Windows Batch Scripting

This book describes and shows how to use the Microsoft-supplied command interpreter cmd.exe and the associated commands, and how to write Windows batch scripts for the interpreter. cmd.exe is the default interpreter on Windows NT, Windows XP, Windows Vista, Windows 7 and later.

Introduction

This book addresses 32-bit Windows commands applicable to modern versions of Windows based on the Windows NT environment. It does not address commands that are specific to DOS environments and to DOS-based operating systems, such as Windows 95, Windows 98, and Windows Me, whose Microsoft-supplied command interpreters are in fact DOS programs, not Win32 programs.

You can find out which version of cmd.exe you are running using the VER command.

This book first describes using the Windows NT command interpreter, how it receives, parses, and processes commands from users. Then it describes various commands available.

To obtain an extensive list of Windows commands and their short summaries, open the command prompt on any Windows computer, and type `help`. To find out about a particular command, type the name of the command followed by "/?".

The subject of this book is also known as "batch programming", even though "batch" refers not only to batch files for MS DOS and Windows command interpreter. Other subject terms include "batch file programming", "batch file scripting", "Windows batch command", "Windows batch file", "Windows command line", "Windows command prompt", and "Windows shell scripting".

Using the Windows command interpreter

How a command line is interpreted

The parsing of a command line into a sequence of commands is complex, and varies subtly from command interpreter to command interpreter. There are, however, four main components:

Variable substitution
: A command line is scanned for variable specifications, and any found are replaced with the contents of those variables.

Quoting
: Special characters can be quoted, to remove their special meanings.

Syntax
: Command lines are developed into a sequence of commands according to a syntax.

Redirection
: Redirection specifications are applied, and removed from the command line, before an individual command in a sequence is executed.

Variable substitution

Command lines can contain variable specifications. These comprise a % character followed by a name, followed by a second % character unless the name is a digit in 0 ... 9 or an asterisk *.

Variable specifications are replaced with values as follows:

- %*varname*%, such as %PATH% or %USERNAME%, is replaced with the value of the named environment variable. For example, %PATH% is replaced by the value of the PATH environment variable.

- %*n* for 0 <= n <= 9, such as %0 or %9, is replaced with is the value of the n-th parameter passed to the batch file when it was invoked, subject to any subsequent modifications by the SHIFT command. For example: %2 is replaced by the value of the second batch file parameter
- %* is replaced with the values of all the command-line parameters except for %0, even those beyond index 9. SHIFT command has no impact on the result of %*. See also Command-line arguments

Special names

Some variable names are not visible using SET command. Rather, they are made available for reading using the % notation. To find out about them, type "help set".

Special variable names and what they expand to:

Name	Replacement Value Used
%CD%	The current directory, not ending in a slash character if it is not in the root directory of the current drive
%TIME%	The system time in HH:MM:SS.mm format.
%DATE%	The system date in a format specific to localization.
%RANDOM%	A generated pseudo-random number between 0 and 32767.
%ERRORLEVEL%	The error level returned by the last executed command, or by the last called batch script.
%CMDEXTVERSION%	The version number of the Command Processor Extensions currently used by cmd.exe.
%CMDCMDLINE%	The content of the command line used when the current cmd.exe was started.

Links:

- Windows Environment Variables at ss64.com
- Command shell overview at Microsoft

Quoting and escaping

You can prevent the special characters that control command syntax from having their special meanings as follows, except for the percent sign (%):

- You can surround a string containing a special character by quotation marks.
- You can place caret (^), an escape character, immediately before the special characters. In a command located after a pipe (|), you need to use three carets (^^^) for this to work.

The special characters that need quoting or escaping are usually <, >, |, &, and ^. In some circumstances, ! and \ may need to be escaped. A newline can be escaped using caret as well.

When you surround the string using quotation marks, they become part of the argument passed to the command invoked. By contrast, when you use caret as an escape character, the caret does not become part of the argument passed.

The percent sign (%) is a special case. On the command line, it does not need quoting or escaping unless two of them are used to indicate a variable, such as %OS%. But in a batch file, you have to use a double percent sign (%%) to yield a single percent sign (%). Enclosing the percent sign in quotation marks or preceding it with caret does not work.

Examples

- echo "Johnson & son"
  - Echoes the complete string rather than splitting the command line at the & character. Quotes are echoed as well
- echo Johnson ^& son
  - As above, but using caret before the special character ampersand. No quotes are echoed.

- echo Johnson & son
    - Does not use an escape character and therefore, "son" is interpreted as a separate command, usually leading to an error message that command son is not found.
- echo A ^^ B
    - Echoes A ^ B. Caret needs escaping as well or else it is interpreted as escaping a space.
- echo > NUL | echo A ^^^^ B
    - Echoes A ^ B. When after a pipe, a caret used for escaping needs to be tripled to work; the fourth caret is the one being escaped.
- if 1 equ 1 ^
  echo Equal &^
  echo Indeed, equal
    - Echoes the two strings. The caret at the end of the line escapes the newlines, leading to the three lines being treated as if they were a single line. The space before the first caret is necessary or else 1 gets joined with the following echo to yield 1echo.
- attrib File^ 1.txt
    - Does not show attributes of file named "File 1.txt" since escaping of space does not work. Using quotes, as in attrib "File 1.txt", works.
- echo The ratio was 47%.
    - If run from a batch, the percent sign is ignored.
- echo The ratio was 47%%.
    - If run from a batch, the percent sign is output once.
- set /a modulo=14%%3
    - If run from a batch, sets modulo variable to 2, the remainder of dividing 14 by 3. Does not work with single %.
- for %%i in (1,2,3) do echo %%i
    - If run from a batch, outputs 1, 2 and 3.
- echo %temp%
    - Outputs the content of temp variable even if run from a batch file. Use of the percent sign in a batch to access environment variables and passed arguments needs no escaping.
- echo ^%temp^%
    - Outputs literally %temp% when run from the command line.
- echo %%temp%%
    - Outputs literally %temp% when run from a batch.
- echo //comment line | findstr \//
    - Command FINDSTR uses backslash (\) for escaping. Unlike caret, this is internal to the command and unknown to the command shell.

Links:

- Syntax : Escape Characters, Delimiters and Quotes at ss64
- Command shell overview at Microsoft
- set at Microsoft

Syntax

Command lines are developed into a sequence of commands according to a syntax. In that syntax, *simple commands* may be combined to form *pipelines*, which may in turn be combined to form *compound commands*, which finally may be turned into *parenthesized commands*.

A simple command is just a command name, a command tail, and some redirection specifications. An example of a simple command is `dir *.txt > somefile`

A pipeline is several simple commands joined together with the "pipe" metacharacter—"|", also known as the "vertical bar". The standard output of the simple command preceding each vertical bar is connected to the standard input of the simple command following it, via a pipe. The command interpreter runs all of the simple commands in the pipeline in parallel. An example of a pipeline (comprising two simple commands) is `dir *.txt | more`.

A compound command is a set of pipelines separated by conjunctions. The pipelines are executed sequentially, one after the other, and the conjunction controls whether the command interpreter executes the next pipeline or not. An example of a compound command (comprising two pipelines, which themselves are just simple commands) is `move file.txt file.bak && dir > file.txt`.

The conjunctions:

- **&** - An unconditional conjunction. The next pipeline is always executed after the current one has completed executing.
- **&&** - A positive conditional conjunction. The next pipeline is executed if the current one completes executing with a zero exit status.
- **||** - A negative conditional conjunction. The next pipeline is executed if the current one completes executing with a non-zero exit status.

A parenthesized command is a compound command enclosed in parentheses (i.e. `(` and `)`). From the point of view of syntax, this turns a compound command into a simple command, whose overall output can be redirected.

For example: The command line `( pushd temp & dir & popd ) > somefile` causes the standard output of the entire compound command `( pushd temp & dir & popd )` to be redirected to *somefile*.

Links:

- Conditional Execution at ss64.com
- Using parenthesis/brackets to group expressions at ss64.com
- Command shell overview at Microsoft

Redirection

Redirection specifications are applied, and removed from the command line, before an individual command in a sequence is executed. Redirection specifications control where the standard input, standard output, and standard error file handles for a simple command point. They override any effects to those file handles that may have resulted from pipelining. (See the preceding section on command syntax.) Redirection signs > and >> can be prefixed with 1 for the standard output (same as no prefix) or 2 for the standard error.

The redirection specifications are:

`< filename`
: Redirect standard input to read from the named file.

`> filename`
: Redirect standard output to write to the named file, overwriting its previous contents.

`>> filename`
: Redirect standard output to write to the named file, appending to the end of its previous contents.

>&*h*
: Redirect to handle *h*, where handle is any of 0—standard input, 1—standard output, 2—standard error, and more.

<&*h*
: Redirect from handle *h*.

Examples:

- dir *.txt >listing.log

    - Redirects the output of the dir command to listing.log file.
- dir *.txt > listing.log
    - As above; the space before the file name makes no difference. However, if you type this into the command window, auto-completion with tab after typing "> l" actually works, while it does not work with ">listing.log".
- dir *.txt 2>NUL
    - Redirects errors of the dir command to nowhere.
- dir *.txt >>listing.log
    - Redirects the output of the dir command to listing.log file, appending to the file. Thereby, the content of the file before the redirected command was executed does not get lost.
- dir *.txt >listing.log 2>&1
    - Redirects the output of the dir command to listing.log file, along with the error messages.
- dir *.txt >listing.log 2>listing-errors.log
    - Redirects the output of the dir command to listing.log file, and the error messages to listing-errors.log file.
- >myfile.txt echo Hello
    - The redirection can precede the command.
- echo Hello & echo World >myfile.txt
    - Only the 2nd echo gets redirected.
- (echo Hello & echo World) >myfile.txt
    - Output of both echos gets redirected.
- type con >myfile.txt
    - Redirects console input (con) to the file. Thus, allows multi-line user input terminated by user pressing Control + Z. See also #User input.
- (for %i in (1,2,3) do @echo %i) > myfile.txt
    - Redirects the entire output of the loop to the file.
- for %i in (1,2,3) do @echo %i > myfile.txt
    - Starts redirection anew each time the body of the loop is entered, losing the output of all but the latest loop iteration.

Links:

- Redirection at ss64.com
- Using command redirection operators at Microsoft

How a command is executed

(...)

Batch reloading

The command interpreter reloads the content of a batch after each execution of a line or a bracketed group.

If you start the following batch and change "echo A" to "echo B" in the batch shortly after starting it, the output will be B.

```
@echo off
ping -n 6 127.0.0.1 >nul & REM wait
echo A
```

What is on a single line does matter; changing "echo A" in the following batch after running it has no impact:

```
@echo off
ping -n 6 127.0.0.1 >nul & echo A
```

Nor have after-start changes have any impact on commands bracketed with (and). Thus, changing "echo A" after starting the following batch has no impact:

```
@echo off
for /L %%i in (1,1,10) do (
  ping -n 2 127.0.0.1 >nul & REM wait
  echo A
)
```

Ditto for any other enclosing, including this one:

```
@echo off
(
ping -n 6 127.0.0.1 >nul & REM wait
echo A
)
```

Environment variables

The environment variables of the command interpreter process are inherited by the processes of any (external) commands that it executes. A few environment variables are used by the command interpreter itself. Changing them changes its operation.

Environment variables are affected by the SET, PATH, and PROMPT commands.

To unset a variable, set it to empty string, such as "set myvar=".

The command interpreter inherits its initial set of environment variables from the process that created it. In the case of command interpreters invoked from desktop shortcuts this will be Windows Explorer, for example.

Command interpreters generally have textual user interfaces, not graphical ones, and so do not recognize the Windows message that informs applications that the environment variable template in the Registry has been changed. Changing the environment variables in Control Panel will cause Windows Explorer to update its own environment variables from the template in the Registry, and thus change the environment variables that any *subsequently invoked* command interpreters will inherit. However, it will not cause command interpreters that *are already running* to update their environment variables from the template in the Registry

COMSPEC

The COMSPEC environment variable contains the full pathname of the command interpreter program file. This is just inherited from the parent process, and is thus indirectly derived from the setting of COMSPEC in the environment variable template in the Registry

PATH

The value of the PATH environment variable comprises a list of directory names, separated by semi-colon characters. This is the list of directories that are searched, in order, when locating the program file of an external command to execute.

PATHEXT

The value of the PATHEXT environment variable comprises a list of filename extensions, separated by semi-colon characters. This is the list of filename extensions that are applied, in order, when locating the program file of an external command to execute.

An example content of PATHEXT printed by "echo %PATHEXT%":

- .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC

By adding ".PL" to the variable, you can ensure Perl programs get run from the command line even when typed without the ".pl" extension. Thus, instead of typing "mydiff.pl a.txt b.txt", you can type "mydiff a.txt b.txt".

Adding ".PL" to the variable in Windows Vista and later:

- setx PATHEXT %PATHEXT%;.PL
  - If you use "set" available in Windows XP, the effect will be temporary and impacting only the current console or process.

Links:

- Windows Environment Variables at ss64
- Making Python scripts run on Windows without specifying ".py" extension at stackoverflow

PROMPT

The PROMPT environment variable controls the text emitted when the command interpreter displays the prompt. The command interpreter displays the prompt when prompting for a new command line in interactive mode, or when echoing a batch file line in batch file mode.

Various special character sequences in the value of the PROMPT environment variable cause various special effects when the prompt is displayed, as in the following table:

Characters	Expansion Result
$$	$ character itself
$A	& symbol AKA ampersand. A convenience, since it is difficult to place a literal & in the value of the PROMPT environment variable using the SET command.
$B	Vertical bar '|' (pipe symbol)
$C	Left parenthesis '('
$D	Current date
$E	ESC (ASCII code 27)
$F	Right parenthesis ')'
$G	Greater-than symbol '>'
$H	Backspace (deletes previous character)
$L	Less-than symbol '<'
$M	Remote name linked to the current drive if it is a network drive; empty string otherwise.
$N	Current drive letter
$P	Current drive letter and full path
$Q	'=' (equals sign)
$S	' ' (space character)
$T	Current system time
$V	Windows version number
$_	<CR> (carriage return character, aka "enter")
$+	As many plus signs (+) as there are items on the pushd directory stack

Links:

- prompt at ss64
- prompt at Microsoft

Switches

Most Windows commands provide switches AKA options to direct their behavior

Observations:

- Switches most often consist of a single-letter; some switches consist of a sequence of multiple letters.

- Switches are preceded with a slash (/) rather than, as in some other operating systems, with a minus sign (-).
- Switches are case-insensitive rather than, as in some other operating systems, case-sensitive.
- If a command from another operating system is ported to Windows (such as grep), it usually retains the option conventions from the original operating system, including the use of minus sign and case-sensitivity.

Examples:

- dir /?
  - Displays the help. This option is provided by many commands.
- dir /b /s
  - Lists all files and folders in the current folder recursively. Two switches are used: b and s.
- dir /bs
  - Does not work; switches cannot be accumulated behind a single slash.
- findstr /ric:"id: *[0-9]*" File.txt
  - Unlike many other commands, findstr allows the accumulation of switches behind a single slash. Indeed, r, i and c are single-letter switches.
- dir/b/s
  - Works. In dir, removing whitespace between the command and the first switch or between the switches does not make a difference; thus, does the same as dir /b /s.
- tree/f/a
  - Does not work, unlike tree /f /a. In tree, separation by whitespace is mandatory. Nor does find/i/v work.
- dir /od
  - The switch letter o is further modified by a single letter specifying that ordering should be by date. The letter d is not a switch by itself. Similar cases include dir /ad and more /t4.
- dir /B /S
  - The switches are case-insensitive, unlike in some other operating systems.
- sort /r file.txt
  - Sorts the file in a reverse order.
- sort /reverse file.txt
  - Sort allows the switch string to be longer than a single-letter.
- sort /reve file.txt
  - Sort allows the specified switch string to be a substring of the complete long name of the switch. Thus, does the same as the above.
- sort /reva file.txt
  - Does not work, since "reva" is not a substring of "reverse".
- taskkill /im AcroRd32.exe
  - Taskkill requires a multiletter switch name for /im; shortening to /i does not work.
- java -version
  - Java, which originated in the environment of another operating system family, uses the minus convention for its switches AKA options.
- grep --help
  - If GNU grep is installed, it requires multi-letter switches to be preceded by two dashes.

Error level

Commands usually set error level at the end of their execution. In Windows NT and later, it is a 32-bit signed integer; in MS DOS, it used to be an integer from 0 to 255. Keywords: return code, exit code, exit status.

The conventional meaning of the error level:

- 0 - success
- not 0 - failure
- The error levels being set are usually positive.
- If the command does not distinguish various kinds of failure, the error level on failure is usually 1.

Uses of the error level:

- It can be tested using && and ||; see also #Syntax.
- It can be tested using IF.
- The value can be accessed from ERRORLEVEL variable.

Examples:

- dir >NUL && echo Success
  - The part after && is executed only if the error level is zero.
- color 00 || echo Failure
  - The part after || is executed only if the error level is non-zero, whether positive or negative.
- color 00 || (
    echo Failure
)
  - Multiline bracketing works as well.
- echo %ERRORLEVEL%
  - Displays the error level without changing it.
- if %errorlevel% equ 0 echo The error level is zero, meaning success.
- if %errorlevel% neq 0 echo The error level is non-zero, meaning failure.
- if errorlevel 1 echo The error level is >= 1, meaning failure via positive error level.
  - Does not cover failure via negative error level. Note the ">=" part: this is not the same as if %errorlevel% equ 1.
- exit /b 1
  - Returns a batch file, setting the error level to 1.
- cmd /c "exit /b 10"
  - In the middle of a batch file or on the command line, sets the error level to 10.
- (cmd /c "exit /b 0" && Echo Success) & (cmd /c "exit /b -1" || Echo Failure)
  - As above, showing the error level is indeed affected.
- (cmd /c "exit /b 0" & cmd /c "exit /b 1") || Echo Failure
  - The error level of a chain created by & is the error level of the last command of the chain.
- cmd /c "exit /b -1" & if not errorlevel 1 echo Would-be success
  - The "if not errorlevel 1" test, which might appear to test for success, passes on negative numbers: it tests on "no error level >= 1", which is "error level <= 0".
- set myerrorlevel=%errorlevel%
  - Remembers the error level for later.
- set errorlevel=0
  - To be avoided: overshadows the built-in errorlevel variable. Ensures that subsequent accesses via %ERRORLEVEL% return 0 rather than the actual error level.
- cmd /c "exit /b 0"
  if 1 equ 1 (cmd /c "exit /b 1" & echo %errorlevel%)
  - Displays 0, since %errorlevel% gets expanded before cmd /c "exit /b 1" gets executed.

Links:

- Error level at ss64

String processing

Getting a substring of a variable:

```
set a=abcdefgh
echo %a:~0,1%   & rem from index 0, length 1; result: a
echo %a:~1,1%   & rem from index 1, length 1; result: b
echo %a:~0,2%   & rem from index 0, length 2; result: ab
echo %a:~1,2%   & rem from index 1, length 2; result: bc
echo %a:~1%     & rem from index 1 to the end; result: bcdefgh
echo %a:~-1%    & rem from index -1 (last char) to the end; result: h
echo %a:~-2%    & rem from index -2 (next-to-last) to the end; result: gh
echo %a:~0,-2%  & rem from index 0 to index -2, excl.; result: abcdef
echo %a:~0,-1%  & rem from index 0 to index -1, excl.; result: abcdefg
echo %a:~1,-1%  & rem from index 1 to index -1, excl.; result: bcdefg
```

Testing substring containment:

- if not "%a:bc=%"=="%a%" echo yes
  - If variable a contains "bc" as a substring, echo "yes".
  - This test is a trick that uses string replacement, discussed below
  - This test does not work if the variable contains a quotation mark.

Testing for "starts with":

```
if %a:~0,1%==a echo yes   & rem If variable a starts with "a", echo "yes".
if %a:~0,2%==ab echo yes  & rem If variable a starts with "ab", echo "yes".
```

String replacement:

```
set a=abcd & echo %a:c=%   & rem replace c with nothing; result: abd
set a=abcd & echo %a:c=e%  & rem replace c with e; result: abed;
set a=abcd & echo %a:*c=%  & rem replace all up to c with nothing; result: d
rem Above, the asterisk (*) only works at the beginning of the sought pattern.
```

See also the help for SET command: set /?.

Splitting a string by any of " ", ",", and ";": ["space", "comma" and "semicolon":]

```
set myvar=a b,c;d
for %%a in (%myvar%) do echo %%a
```

Splitting a string by semicolon, assuming the string contains no quotation marks:

```
@echo off
set myvar=a b;c;d
set strippedvar=%myvar%
:repeat
for /f "delims=;" %%a in ("%strippedvar%") do echo %%a
set prestrippedvar=%strippedvar%
set strippedvar=%strippedvar:*;=%
if not "%prestrippedvar:;=%"=="%prestrippedvar%" goto :repeat
```

Limitations:

- The above string processing does not work with parameter variables (%1, %2, ...).

Links:

- Variables: extract part of a variable (substring) at ss64
- Variable Edit/Replace at ss64

Command-line arguments

The command-line arguments AKA command-line parameters passed to a batch script are accessible as %1, %2, ..., %9. There can be more than nine arguments; to access them, see how to loop over all of them below

The syntax %0 does not refer to a command-line argument but rather to the name of the batch file.

Testing for whether the first command-line argument has been provided:

```
if not -%1-==-- echo Argument one provided
if -%1-==-- echo Argument one not provided  & exit /b
```

A robust looping over all command-line arguments using SHIFT (for each command-line argument, ...):

```
:argactionstart
if -%1-==-- goto argactionend
echo %1 & REM Or do any other thing with the argument
shift
goto argactionstart
:argactionend
```

A robust looping over all command-line arguments using SHIFT without modifying %1, %2, etc.:

```
call :argactionstart %*
echo Arg one: %1 & REM %1, %2, etc. are unmodified in this location
exit /b

:argactionstart
if -%1-==-- goto argactionend
echo %1 & REM Or do any other thing with the argument
shift
goto argactionstart
:argactionend
exit /b
```

Transferring command-line arguments to environment variables:

```
setlocal EnableDelayedExpansion
REM Prevent affecting possible callers of the batch
REM Without delayed expansion, !arg%argno%! used below won't work.
set argcount=0
:argactionstart
if -%1-==-- goto argactionend
set /a argcount+=1
set arg%argcount%=%1
shift
goto argactionstart
:argactionend

set argno=0
:loopstart
set /a argno+=1
if %argno% gtr %argcount% goto loopend
echo !arg%argno%! & REM Or do any other thing with the argument
goto loopstart
:loopend
```

Looping over all command-line arguments, albeit not a robust one:

```
for %%i in (%*) do (
  echo %%i
)
```

This looks elegant but is non-robust, maltreating arguments containing wildcards (*, ?). In particular, the above for command replaces arguments that contain wildcards (*, ?) with file names that match them, or drops them if no files match. Nonetheless, the above loop works as expected as long as the passed arguments do not contain wildcards.

Finding the number of command-line arguments, in a non-robust way:

```
set argcount=0
for %%i in (%*) do set /a argcount+=1
```

Again, this does not work with arguments containing wildcards.

The maximum possible number of arguments is greater than 4000, as empirically determined on a Windows Vista machine. The number can differ on Windows XP and Windows 7.

In passing arguments to a batch script, characters used for argument separation are the following ones:

- space
- comma
- semicolon
- equal sign
- tab character

Thus, the following lines pass the same four arguments:

- test.bat a b c d
- test.bat a,b,c,d
- test.bat a, b, c, d
- test.bat a;b;c;d
- test.bat a=b=c=d
- test.bat a b,c;,;=d

Yes, even the line with "a b,c;,;=d" passes four arguments, since a sequence of separating characters is considered a single separator.

To have a space, comma or semicolon in the argument value, you can pass the value enclosed in quotation marks. However, the quotation marks become part of the argument value. To get rid of the enclosing quotation marks when referring to the argument in the script, you can use %~<number> described in #Percent tilde.

When passing arguments to an invoked command rather than a batch script, you usually need to separate the command from the first argument using a space. However, for internal commands, that separation is not necessary if the first character after the command name is one of a couple of symbols, including .\/, and more:

- echo.
  - Outputs a newline.
- tree.
  - Fails: "tree." not found. tree is an external command.
- dir..
  - Lists the content of the parent directory
- cd..
  - Changes the current directory to the parent one.
- cd\
  - Changes the current directory to the root one.
- start.
  - Opens Windows Explorer from the current directory.
- dir/b/s
  - Lists directory content recursively showing full paths.

Links:

- Parameters / Arguments at ss64
- Escape Characters, Delimiters and Quotes at ss64
- Using batch parameters at Microsoft

Wildcards

Many commands accept file name wildcards--characters that do not stand for themselves and enable matching of a group of filenames.

Wildcards:

- * (asterisk): any sequence of characters
- ? (question mark): a single character other than a period (".") or, if part of a sequence of question marks at the end of a maximum period-free part of a file name, possibly zero number of characters; see examples for clarification

Examples:

- dir *.txt
  - Matches Myfile.txt, Plan.txt and any other file with the .txt extension.
- dir *txt
  - The period does not need to be included. However, this will also match files named without the period convention, such as myfiletxt.
- ren *.cxx *.cpp
  - Renames all files with .cxx extension to have .cpp extension.
- dir a?b.txt
  - Matches files aab.txt, abb.txt, a0b.txt, etc.
  - Does not match ab.txt, since a question mark followed by a character other than a question mark or period cannot match zero characters.
  - Does not match a.b.txt, since a question mark cannot match a period.
- dir ???.txt
  - Matches .txt, a.txt, aa.txt, and aaa.txt, among others, since each question mark in the sequence followed by a period can match zero number of characters.
- dir a???.b???.txt???
  - Matches a.b.txt, among others. While the last question mark sequence is not followed by a period, it is still a sequence at the end of a maximum period-free part of a file name.
- dir ????????.txt & @REM eight question marks
  - Matches the same files as *.txt, since each file also has a short file name that has no more than 8 characters before .txt.

Quirk with short file names: the wildcard matching is performed both on long file names and the usually hidden short 8 chars + period + 3 chars file names. This can lead to bad surprises.

Unlike shells of some other operating systems, the cmd.exe shell does not perform wildcard expansion (replacement of the pattern containing wildcards with the list of file names matching the pattern) on its own. It is the responsibility of each program to treat wildcards as such. This enables such things as "ren *.txt *.bat", since the ren command actually sees the * wildcard rather than a list of files matching the wildcard. Thus, "echo *.txt" does not display files in the current folder matching the pattern but rather literally displays "*.txt". Another consequence is that you can write "findstr a.*txt" without fearing that the "a.*txt" part gets replaced with th names of some files in the current folder. Furthermore, recursive "findstr /s pattern *.txt" is possible, while in some other operating systems, the "*.txt" part would get replaced with the file names found in the current folder, disregarding nested folders.

Commands accepting wildcards include ATTRIB, COPY, DIR, FINDSTR, FOR, REN, etc.

Links:

- Wildcards at ss64
- Using wildcard characters at Microsoft

User input

You can get input from the user using the followig methods:

- SET /P command
- CHOICE command
- Using "type con >myfile.txt", for which the multi-line user input is terminated by user pressing Control + Z.

Percent tilde

When a command-line argument contains a file name, special syntax can be used to get various information about the file.

The following syntaxes expand to various information about the file passed as %1:

Syntax	Expansion Result	Example
%~1	%1 with no enclosing quotation marks	Not provided
%~f1	Full path with a drive letter	C:\Windows\System32\notepad.exe
%~d1	Drive letter	C:
%~p1	Drive-less path with the trailing backslash	\Windows\System32\
%~n1	For a file, the file name without path and extension For a folder, the folder name	notepad
%~x1	File name extension including the period	.exe
%~s1	Modify of f, n and x to use short name	Not provided
%~a1	File attributes	--a------
%~t1	Date and time of last modification of the file	02.11.2006 11:45
%~z1	File size	151040
%~pn1	A combination of p and n	\Windows\System32\notepad
%~dpnx1	A combination of several letters	C:\Windows\System32\notepad.exe
%~$PATH:1	The full path of the first match found in the folders present in the PATH variable, or an empty string in no match.	
%~n0	%~n applied to %0: The extensionless name of the batch	tildetest
%~nx0	%~nx applied to %0: The name of the batch	tildetest.bat
%~d0	%~f applied to %0: The drive letter of the batch	C:
%~dp0	%~dp applied to %0: The folder of the batch with trailing backslash	C:\Users\Joe Hoe\

The same syntax applies to single-letter variables created by FOR command, such as "%%i".

To learn about this subject from the command line, type "call /?" or "for /?".

Links:

- Parameters / Arguments at ss64
- Using batch parameters at Microsoft
- for at Microsoft

Functions

Functions AKA subprograms can be emulated using CALL, labels, SETLOCAL and ENDLOCAL.

An example of a function that determines arithmetic power:

```
@echo off
call :power 2 4
echo %result%
rem Prints 16, determined as 2 * 2 * 2 * 2
goto :eof

rem __Function power______________________
rem Arguments: %1 and %2
:power
setlocal
set counter=%2
set interim_product=%1
:power_loop
if %counter% gtr 1 (
  set /a interim_product=interim_product * %1
  set /a counter=counter - 1
  goto :power_loop
)
endlocal & set result=%interim_product%
goto :eof
```

While the goto :eof at the end of the function is not really needed, it has to be there in the general case in which there is more than one function.

The variable into which the result should be stored can be specified on the calling line as follows:

```
@echo off
call :sayhello result=world
echo %result%
exit /b

:sayhello
set %1=Hello %2
REM Set %1 to set the returning value
exit /b
```

In the example above, exit /b is used instead of goto :eof to the same effect.

Also, remember that the equal sign is a way to separate parameters. Thus, the following items achieve the same:

- call :sayhello result=world
- call :sayhello result world
- call :sayhello result,world
- call :sayhello result;world

(See Command-line arguments as a reminder)

Links:

- Functions at ss64

Calculation

Batch scripts can do simple 32-bit integer arithmetic and bitwise manipulation using SET /a command. The largest supported integer is 2147483647 = 2 ^ 31 - 1. The smallest supported integer is -2147483648 = - (2 ^ 31), assignable with the trick of set /a num=-2147483647-1. The syntax is reminiscent of the C language.

Arithmetic operators include *, /, % (modulo), +, -. In a batch, modulo has to be entered as "%%".

Bitwise operators interpret the number as a sequence of 32 binary digits. These are ~ (complement), & (and), | (or), ^ (xor), << (left shift), >> (right shift).

A logical operator of negation is !: it turns zero into one and non-zero into zero.

A combination operator is ,: it allows more calculations in one set command.

Combined assignment operators are modeled on "+=", which, in "a+=b", means "a=a+b". Thus, "a-=b" means "a=a-b". Similarly for *=, /=, %=, &=, ^=, |=, <<=, and >>=.

The precedence order of supported operators, is as follows:

1. ( )
2. * / % + -
3. << >>
4. &
5. ^
6. |
7. = *= /= %= += -= &= ^= |= <<= >>=
8. ,

Literals can be entered as decimal (1234), hexadecimal (0xffff, leading 0x), and octal (0777, leading 0).

The internal bit representation of negative numbers is two's complement. This provides a connection between arithmetic operations and bit operations. For instance, -2147483648 is represented as 0x80000000, and therefore set /a num=~(-2147483647-1) yields 2147483647, which equals 0x7FFFFFFF (type set /a num=0x7FFFFFFF to check).

As some of the operators have special meaning for the command interpreter, an expression using them needs to be enclosed in quotation marks, such as this:

- set /a num="255^127"
- set /a "num=255^127"
  - Alternative placement of quotation marks.
- set /a num=255^^127
  - Escape ^ using ^ instead of quotation marks.

Examples:

- set n1=40 & set n2=25

  set /a n3=%n1%+%n2%
  - Uses the standard percent notation for variable expansion.
- set n1=40 & set n2=25

  set /a n3=n1+n2
  - Avoids the percent notation around variable names as unneeded for /a.
- set /a num="255^127"
  - Encloses "^" in quotation marks to prevent its special meaning for the command interpreter
- set /a n1 = (10 + 5)/5

    - The spaces around = do not matter with /a. However, getting used to it lends itself to writing "set var = value" without /a, which sets the value of "var " rather than "var".
- if 1==1 (set /a n1=(2+4)*5)
    - Does not work: the arithmetic brackets within grouping brackets cause trouble.
- if 1==1 (set /a n1=^(2+4^)*5)
    - Escaping the arithmetic brackets with caret (^) works, as does enclosing "n1=2+(4*5)" in quotation marks.
- set /a n1=2+3,n2=4*7
    - Performs two calculations.
- set /a n1=n2=2
    - Has the same effect as n1=2,n2=2.
- set n1=40 & set n2=25 & set /a n3=n1+n2
    - Works as expected.
- set /a n1=2,n2=3,n3=n1+n2
    - Works as expected.
- set n1=40 & set n2=25 & set /a n3=%n1%+%n2%
    - Does not work unless n1 and n2 were set previously. The variable specifications "%n1%" and "%n2"% get expanded *before* the first set command is executed. Dropping percent notation makes it work.
- set /a n1=2,n2=3,n3=%n1%+%n2%
    - Does not work unless n1 and n2 were set previously, for the reason stated in the previous example.
- set /a n1=0xffff
    - Sets n1 using hexadecimal notation.
- set /a n1=0777
    - Sets n1 using octal notation.
- set /a n1=%random%
    - A pseudo-random number from 0 to 32767 = 2^15-1.
- set /a n1="%random%>>10"
    - A pseudo-random number from 0 to 31 = 2^5-1. The shift right operator drops 10 out of 15 bits, keeping 5 bits.
- set /a n1=%random%%50
    - A pseudo-random number from 0 to 49. Uses the % modulo operator. In a batch, %% is needed for modulo: set /a n1=%random%%%50. Because of this particular use of the modulo, the result is not perfectly uniform; it is uniform if the 2nd modulo operand--above 50--equals to a power of 2, e.g. 256 = 2^8.
- set /a n1="(%random%<<15)+%random%"
    - A pseudo-random number from 0 to 1073741823 = 2^30 - 1. Combines the two 15-bit random numbers produced by %random% alone to produce a single 30-bit random number
- set /a n1="((%random%<<15)+%random%)%1000000"
    - As above, but again using modulo, this time to achieve the range 0 to 999999.

An example calculation that prints prime numbers:

```
@echo off
setlocal
set n=1
:print_primes_loop
set /a n=n+1
set cand_divisor=1
:print_primes_loop2
set /a cand_divisor=cand_divisor+1
set /a cand_divisor_squared=cand_divisor*cand_divisor
if %cand_divisor_squared% gtr %n% echo Prime %n% & goto :print_primes_loop
set /a modulo=n%%cand_divisor
if %modulo% equ 0 goto :print_primes_loop & REM Not a prime
goto :print_primes_loop2
```

Links:

- set at ss64.com
- set at Microsoft
- Random Numbers at ss64.com

Finding files

Files can be found using #DIR, #FOR, #FINDSTR, #FORFILES, and #WHERE.

Examples:

- dir /b /s *base*.doc*
  - Outputs all files in the current folder and its subfolders such that the file name before the extension contains the word "base" and whose extension starts with "doc", which includes "doc" and "docx". The files are output with ful paths, one file per line.
- dir /b /s *.txt | findstr /i pers.*doc
  - Combines the result of outputting files including their complete paths with the findstr filtering command supporting limited regular expressions, yielding a versatile and powerful combination for finding files by names and the names of their directories.
- for /r %i in (*) do @if %~zi geq 1000000 echo %~zi %i
  - For each file in the current folder and its subfolders that has the size greater than or equal to 1,000,000 bytes, outputs the file size in bytes and the full path of the file. For the syntax in %~zi, see #Percent tilde.
- forfiles /s /d 06/10/2015 /c "cmd /c echo @fdate @path"
  - For each file in the current folder and its subfolders modified on 10 June 2015 or later, outputs the file modification date and full file path. The date format after /d is locale specific. Thus, allows to find most recently modified files.
- (for /r %i in (*) do @echo %~ti :: %i) | findstr 2015.*::
  - Searching the current folder recursively, outputs files whose last modification date is in year 2015. Places the modification date and time, followed by a double colon, before the file name. Works as long as the used version of Windows and locale displays dates in a format that contains four-digit years. The double colon is used to mak sure the findstr command is matching the date and not the file name.
- for /r %i in (*) do @echo %~ti | findstr 2015 >NUL && echo %i
  - As above, outputs files changed in 2015. Unlike the above, only outputs the files, not the modification dates.
- findstr /i /s /m cat.*mat *.txt
  - Finds files by their content. Performs a full text search for regular expression cat.*mat in files with names ending in .txt, and outputs the files names. The /m switch ensures only the file names are output.
- where *.bat
  - Outputs all .bat files in the current directory and in the directories that are in PATH.

Keyboard shortcuts

When using Windows command line from the standard console that appears after typing cmd.exe after pressing Windows + R, you can use multiple keyboard shortcuts, including function keys:

- Tab: Completes the relevant part of the typed string from file names or folder names in the current folder. The relevant part is usually the last space-free part, but use of quotation marks changes that. Generally considers both files and folders for completion, but cd command only considers folders.
- Up and down arrow keys: Enters commands from the command history, one at a time.
- Escape: Erases the current command line being typed.
- F1: Types the characters from the single previously entered command from the command history, one character at a time. Each subsequent press of F1 enters one more character
- F2: Asks you to type a character, and enters the shortest prefix of the previous command from the command history that does not include the typed character. Thus, if the previous command was echo Hello world and you typed o,

enters ech.

- F3: Enters the single previous command from the command history. Repeated pressing has no further effect.
- F4: Asks you to type a character, and erases the part of the currently typed string that starts at the current cursor location, continues to the right, and ends with the character you entered excluding that character. Thus, if you type echo Hello world, place the cursor at H using left arrow key, press F4 and then w, you get echo world.
- F5: Enters previous commands from the command history, one at a time.
- F6: Enters Control+Z character.
- F7: Opens a character-based popup window with the command history, and lets you use arrow key and enter to select a command. After you press enter in the popup, the command is immediately executed.
- F8: Given an already typed string, shows items from the command history that have that string as a prefix, one at a time.
- F9: Lets you enter the number of the command from the command history, and then executes the command.
- Alt + F7: Erases the command history.

The above are also known as command prompt keyboard shortcuts.

The availability of the above shortcuts does not seem to depend on running DOSKEY.

Links:

- Windows Keyboard shortcuts at ss64.com
- doskey at Microsoft

Paths

File and directory paths follow certain conventions. These include the possible use of a drive letter followed by a colon (:), the use of backslash (\) as the path separator, and the distinction between relative and absolute paths.

Forward slash (/) often works when used instead of (\) but not always; it is normally used to mark switches (options). Using forward slash can lead to various obscure behaviors, and is best avoided.

Special device names include NUL, CON, PRN, AUX, COM1, ..., COM9, LPT1, ..., LPT9; these can be redirected to.

Examples:

- attrib C:\Windows\System32\notepad.exe
  - Succeeds if the file exists, as it should. This is an *absolute* path with a drive letter. It is also known as a *fully qualified* path.
- attrib \Windows\System32\notepad.exe
  - Succeeds if the current drive is C:, and if the file exists, as it should. This is an absolute path without a drive letter.
- cd /d C:\Windows & attrib System32\notepad.exe
  - Succeeds if the file exists. The path given to attrib is a *relative* path.
- cd /d C:\Windows\System32 & attrib C:notepad.exe
  - Succeeds if the file exists. The path given to attrib is a *relative* one despite containing a drive letter: there would have to be C:\notepad.exe with a backslash for that to be an absolute path.
- cd /d C:\Windows & attrib .\System32\notepad.exe
  - Succeeds if the file exists. A single period denotes the current folder
- attrib .
  - A single period denotes the current folder.
- cd /d C:\Windows & attrib .\System32\\\notepad.exe
  - Succeeds if the file exists. Piling of backslashes has no impact beyond the first backslash.
- cd /d C:\Windows & attrib .\System32
  - Succeeds if the folder exists.

- cd /d C:\Windows & attrib .\System32\
  - Fails. Folders are usually denoted without the final backslash.
- cd C:\Windows\System32\
  - Succeeds, whyever.
- cd ..
  - A double period denotes the parent folder.
- attrib C:\Windows\System32\..\..\Windows\System32
  - A double period can be used in the middle of the path to navigate to the parent folder, even multiple times.
- attrib \\myserver\myvolume
  - A network UNC path starts with double backslash and no drive letter.
- cd \\myserver\myvolume
  - Does not work; changing to a server folder in this direct manner does not work.
- pushd \\myserver\folder
  - Automatically creates a drive for the folder and changes to it. After you use #POPD, the drive gets unassigned again.
- attrib C:/Windows/System32/notepad.exe
  - Succeeds on multiple versions of cmd.exe. Uses forward slashes.

Links:

- Long filenames, NTFS and legal filename characters at ss64.com
- Naming Files, Paths, and Namespaces at Microsoft
- W:Path (computing)#MS-DOS/Microsoft Windows style, wikipedia.org
- Why does the cmd.exe shell on Windows fail with paths using a forward-slash ('/') *path separator?*, *stackoverflow.com*

Arrays

Arrays can be emulated in the delayed expansion mode using the combination of % and ! to indicate variables. There, %i% is the value of variable i with the *immediate* expansion while !i! is the value of variable i in the *delayed* expansion.

```
@echo off
setlocal EnableDelayedExpansion
for /l %%i in (1, 1, 10) do (
  set array_%%i=!random!
)

for /l %%i in (1, 1, 10) do (
  echo !array_%%i!
)

:: For each item in the array, not knowing the length
set i=1
:startloop
if not defined array_%i% goto endloop
set array_%i%=!array_%i%!_dummy_suffix
echo A%i%: !array_%i%!
set /a i+=1
goto startloop
:endloop
```

Links:

- Arrays, linked lists and other data structures in cmd.exe (batch) script, stackoverflow.com

Perl one-liners

Some tasks can be conveniently achieved with Perl one-liners. Perl is a scripting language originating in the environment of another operating system. Since many Windows computing environments have Perl installed, Perl one-liners are a natural and compact extension of Windows batch scripting.

Examples:

- echo "abcbbc"| perl -pe "s/a.*?c/ac/"
  - Lets Perl act as sed, the utility that supports textual replacements specified using regular expressions.
- echo a b| perl -lane "print $F[1]"
  - Lets Perl act as cut command, displaying the 2nd field or column of the line, in this case b. Use $F[2] to display 3rd field; indexing starts at zero. Native solution FOR /f.
- perl -ne "print if /\x22hello\x22/" file.txt
  - Acts as grep or FINDSTR, outputting the lines in file.txt that match the regular expression after if. Uses the powerful Perl regular expressions, more powerful than those of FINDSTR.
- perl -ne "$. <= 10 and print" MyFile.txt
  - Lets Perl act as head -10 command, outputting the first 10 lines of the file.
- perl -e "sleep 5"
  - Waits for 5 seconds.
- for /f %i in ('perl -MPOSIX -le "print strftime '%Y-%m-%d', localtime"') do @set isodate=%i
  - Gets current date in the ISO format into isodate variable.
- perl -MWin32::Clipboard -e "print Win32::Clipboard->Get()"
  - Outputs the text content of the clipboard. When stored to getclip.bat, yields a handy getclip command to complement CLIP command.
- perl -MText::Diff -e "print diff 'File1.txt', 'File2.txt'"
  - Outputs differences between two files in a format similar to diff command known from other operating systems, including context lines, lines starting with + and lines starting with -.

On the web, Perl one-liners are often posted in the command-line conventions of another operating system, including the use of apostrophe (') to surround the arguments instead of Windows quotation marks. These need to be tweaked for Windows.

Links:

- Perl One-Liners by Peteris Krumins at github.com
- Why doesn't my Perl one-liner work on Windows? at stackoverflow.com
- W:One-liner program#Perl

Limitations

There is no touch command familiar from other operating systems. The touch command would modify the last-modification timestamp of a file without changing its content.

One workaround, with unclear reliability and applicability across various Windows versions, is this:

- copy /b file.txt+,,

Links:

- Windows recursive touch command at superuser.com
- Windows version of the Unix touch command at stackoverflow.com

Built-in commands

These commands are all built in to the command interpreter itself, and cannot be changed. Sometimes this is because they require access to internal command interpreter data structures, or modify properties of the command interpreter process itself.

Overview

Command	Description
ASSOC	Associates an extension with a file type (FTYPE).
BREAK	Sets or clears extended CTRL+C checking.
CALL	Calls one batch program from another.
CD, CHDIR	Displays or sets the current directory.
CHCP	Displays or sets the active code page number.
CLS	Clears the screen.
COLOR	Sets the console foreground and background colors.
COPY	Copies files.
DATE	Displays and sets the system date.
DEL, ERASE	Deletes one or more files.
DIR	Displays a list of files and subdirectories in a directory.
ECHO	Displays messages, or turns command echoing on or off.
ELSE	Performs conditional processing in batch programs when "IF" is not true.
ENDLOCAL	Ends localization of environment changes in a batch file.
EXIT	Quits the CMD.EXE program (command interpreter).
FOR	Runs a specified command for each file in a set of files.
FTYPE	Sets the file type command.
IF	Performs conditional processing in batch programs.
MD, MKDIR	Creates a directory.
MOVE	Moves a file to a new location
PATH	Sets or modifies the PATH environment
PAUSE	Causes the command session to pause for user input.
POPD	Changes to the drive and directory popped from the directory stack
PROMPT	Sets or modifies the string displayed when waiting for input.
PUSHD	Pushes the current directory onto the stack, and changes to the new directory.
RD / RMDIR	Removes the directory.
REM	A comment command. Unlike double-colon (::), the command can be executed.
REN / RENAME	Renames a file or directory
SET	Sets or displays shell environment variables
SETLOCAL	Creates a child-environment for the batch file.
SHIFT	Moves the batch parameters forward.
START	Starts a program with various options.
TIME	Displays or sets the system clock
TITLE	Changes the window title
TYPE	Prints the content of a file to the console.
VER	Shows the command processor, operating system versions.
VERIFY	Verifies that file copy has been done correctly.
VOL	Shows the label of the current volume.

ASSOC

Associates an extension with a file type (FTYPE), displays existing associations, or deletes an association. See also FTYPE.

Examples:

- assoc
    - Lists all associations, in the format "<file extension>=<file type>", as, for example, ".pl=Perl" or ".xls=Excel.Sheet.8".
- assoc | find ".doc"
    - Lists all associations containing ".doc" substring.

Links:

- assoc at ss64.com
- assoc at Microsoft
- Making Python scripts run on Windows without specifying “.py” extension at stackoverflow

BREAK

In Windows versions based on Windows NT, does nothing; kept for compatibility with MS DOS.

Links:

- break at Microsoft

CALL

Calls one batch program from another, calls a subprogram within a single batch program, or, as an undocumented behavior, starts a program. In particular, suspends the execution of the caller, starts executing the callee, and resumes the execution of the caller if and when the callee finishes execution.

For calling a subprogram, see Functions section.

Beware that calling a batch program from a batch without using the *call* keyword results in the execution never returning to the caller once the callee finishes.

The callee inherits environment variables of the caller, and unless the callee prevents that via SETLOCAL, changes made by the callee to environment variables become visible to the caller once it resumes execution.

Examples:

- mybatch.bat
    - If used in a batch, transfers control to mybatch.bat and never resumes the execution of the caller
- call mybatch.bat
- call mybatch
- call mybatch.bat arg1 "arg 2"
- call :mylabel
- call :mylabel arg1 "arg 2"
- cmd /c mybatch.bat
    - Similar to call, but resumes execution even when there are errors. Furthermore, any changes the callee makes to environment variables are not propagated to the caller
- call notepad.exe
    - Launches Notepad, or in general, any other executable. This is apparently not the intended usage of call, and is not officially documented.

See also Functions, CMD amd START.

Links:

- call at ss64.com

- call at Microsoft
- CALL command vs. START with /WAIT option, stackoverflow.com

CD

Changes to a different directory, or displays the current directory. However, if a different drive letter is used, it does not switch to that different drive or volume.

Examples:

- cd
  - Outputs the current directory e.g. C:\Windows\System32.
- cd C:\Program Files
  - No surrounding quotes are needed around paths with spaces.
- cd \Program Files
- cd Documents
- cd %USERPROFILE%
- cd /d C:\Program Files
  - Changes to the directory of the C: drive even if C: is not the current drive.
- C: & cd C:\Program Files.
  - Changes to the directory of the C: drive even if C: is not the current drive.
- cd ..
  - Changes to the parent directory. Does nothing if already in the root directory
- cd ..\..
  - Changes to the parent directory two levels up.
- C: & cd C:\Windows\System32 & cd ..\..\Program Files
  - Uses ".." to navigate through the directory tree up and down
- cd \\myserver\folder
  - Does not work. Changing the directory directly to a network Universal Naming Convention (UNC) folder does not work. Keywords: UNC path.
- subst A: \\myserver\folder && cd /d A:
  - Changes the directory to a server folder with the use of #SUBST command, assuming drive letter A: is free.
- pushd \\myserver\folder
  - Automatically creates a drive for the folder and changes to it. After you use #POPD, the drive gets unassigned again.
- cd C:\W*
  - Changes to C:\Windows, in a typical Windows setup. Thus, wildcards work. Useful for manual typing from the command line.
- cd C:\W**32
  - Changes to C:\Windows\System32, in a typical Windows setup.

Links:

- cd at ss64.com
- cd at Microsoft

CHDIR

A synonym of CD.

CLS

Clears the screen.

COLOR

Sets the console foreground and background colors.

Examples:

- color f9
  - Use white background and blue foreground.
- color
  - Restore the original color setting.

Links:

- color at ss64.com
- color at Microsoft

COPY

Copies files. See also MOVE.

Examples:

- copy F:\File.txt
  - Copies the file into the current directory, assuming the current directory is not F:\.
- copy "F:\My File.txt"
  - As above; quotation marks are needed to surround a file with spaces.
- copy F:\*.txt
  - Copies the files located at F:\ and ending in dot txt into the current directory, assuming the current directory is not F:\.
- copy F:\*.txt .
  - Does the same as the above command.
- copy File.txt
  - Issues an error message, as File.txt cannot be copied over itself.
- copy File1.txt File2.txt
  - Copies File1.txt to File2.txt, overwriting File2.txt if confirmed by the user or if run from a batch script.
- copy File.txt "My Directory"
  - Copies File.txt into "My Directory" directory, assuming "My Directory" exists.
- copy Dir1 Dir2
  - Copies all files directly located in directory Dir1 into Dir2, assuming Dir1 and Dir2 are directories. Does not copy files located in nested directories of Dir1.
- copy *.txt *.bak
  - For each *.txt file in the current folder, makes a copy ending with "bak" rather than "txt".

Links:

- copy at ss64.com
- copy at Microsoft

DEL

Deletes files. *Use with caution, especially in combination with wildcards.* Only deletes files, not directories, for which see RD. For more, type "del /?".

Examples:

- del File.txt
- del /s *.txt
  - Deletes the files recursively including nested directories, but keeps the directories; mercilessly deletes all matching files without asking for confirmation.
- del /p /s *.txt
  - As above, but asks for confirmation before every single file.
- del /q *.txt
  - Deletes without asking for confirmation.

Links:

- del at ss64.com
- del at Microsoft

DIR

Lists the contents of a directory. Offers a range of options. Type "dir /?" for more help.

Examples:

- dir
  - Lists the files and folders in the current folder, excluding hidden files and system files; uses a different manner of listing if DIRCMD variable is non-empty and contains switches for dir
- dir D:
- dir /b C:\Users
- dir /s
  - Lists the contents of the directory and all subdirectories recursively
- dir /s /b
  - Lists the contents of the directory and all subdirectories recursively, one file per line, displaying complete path for each listed file or directory
- dir *.txt
  - Lists all files with .txt extension.
- dir /a
  - Includes hidden files and system files in the listing.
- dir /ah
  - Lists hidden files only.
- dir /ad
  - Lists directories only. Other letters after /A include S, I, R, A and L.
- dir /ahd
  - Lists hidden directories only
- dir /a-d
  - Lists files only, omitting directories.
- dir /a-d-h

    - Lists non-hidden files only omitting directories.
- dir /od
    - Orders the files and folders by the date of last modification. Other letters after /O include N (by name), E (by extension), S (by size), and G (folders first)
- dir /o-s
    - Orders the files by the size descending; the impact on folder order is unclear
- dir /-c /o-s /a-d
    - Lists files ordered by size descending, omitting the thousands separator via /-C, excluding folders.
- dir /s /b /od
    - Lists the contents of the directory and all subdirectories recursively, ordering the files in each directory by the date of last modification. The ordering only happens per directory; the complete set of files so found is not ordered as a whole.
- dir /a /s
    - Lists files recursively including hidden files and system files. Can be used to find out the disk usage (directory size), by considering the final lines of the output.

Links:

- dir at ss64.com
- dir at Microsoft

DATE

Displays or sets the date. The way the date is displayed depends on country settings. Date can also be displayed using "echo %DATE%".

Getting date in the iso format, like "2000-01-28": That is nowhere easy, as the date format depends on country settings.

- If you can assume the format of "Mon 01/28/2000", the following will do:
    - set isodate=%date:~10,4%-%date:~4,2%-%date:~7,2%
- If you have WMIC, the following is locale independent:
    - for /f %i in ('wmic os get LocalDateTime') do @if %i lss a if %i gtr 0 set localdt=%i
      set isodate=%localdt:~0,4%-%localdt:~4,2%-%localdt:~6,2%
    - To use the above in a batch, turn %i into %%i and remove @ from before if.
- If you have Perl installed:
    - for /f %i in ('perl -MPOSIX -le "print strftime '%Y-%m-%d', localtime"') do @set isodate=%i

Links:

- date at ss64.com
- date at Microsoft
- How to get current datetime on Windows command line, in a suitable format for using in a filename?, stackoverflow

ECHO

Displays messages, or turns command echoing on or off.

Examples:

- echo on
- @echo off
- echo Hello
- echo "hello"

    - Displays the quotes too.
- echo %PATH%
    - Displays the contents of PATH variable.
- echo Owner ^& son
    - Uses caret (^) to escape ampersand (&), thereby enabling echoing ampersands.
- echo 1&echo 2&echo 3
    - Displays three strings, each followed by a newline.
- echo.
    - Outputs a newline while the period is not being output. Without the period, outputs "echo off" or "echo on". Adding a space before the period leads to the period being output. Other characters having the same effect as period include :;,/\(=+[].
- echo %random%>>MyRandomNumbers.txt
    - While it seems to output random numbers to MyRandomNumbers.txt, it actually does not do so for numbers 0-9, since these, when placed before >>, indicate which channel is to be redirected. See also #Redirection.
- echo 2>>MyRandomNumbers.txt
    - Instead of echoing 2, redirects standard error to the file.
- (echo 2)>>MyRandomNumbers.txt
    - Echoes even a small number (in this case 2) and redirects the result.
- >>MyRandomNumbers.txt echo 2
    - Another way to echo even a small number and redirect the result.

Displaying a string without a newline requires a trick:

- set <NUL /p=Output of a command:
    - Displays "Output of a command:". The output of the next command will be displayed immediately after ":".
- set <NUL /p=Current time: & time /t
    - Displays "Current time: " followed by the output of "time /t".
- (set <NUL /p=Current time: & time /t) >tmp.txt
    - Like before, with redirecting the output of both commands to a file.

Links:

- echo at ss64.com
- echo at Microsoft

ELSE

An example:

```
if exist file.txt (
  echo The file exists.
) else (
  echo The file does not exist.
)
```

See also IF.

ENDLOCAL

Ends local set of environment variables started using SETLOCAL. Can be used to create subprograms: see Functions.

Links:

- endlocal at ss64.com
- endlocal at Microsoft

ERASE

A synonym of DEL.

EXIT

Exits the DOS console or, with /b, only the currently running batch or the currently executed subroutine. If used without /b in a batch file, causes the DOS console calling the batch to close.

Examples:

- exit
- exit /b

Links:

- exit at ss64.com
- exit at Microsoft

FOR

Iterates over a series of values, executing a command.

In the following examples, %i is to be used from the command line while %%i is to be used from a batch. The index (e.g., %i) must be a single character variable name.

Examples:

- for %%i in (1,2,3) do echo %%i
  - In a batch, echoes 1, 2, and 3. In a batch, the command must use a double percent sign.
  - The remaining examples are intended to be directly pasted into a command line, so they use a single percent sign and include "@" to prevent repetitive display.
- for %i in (1,2,3) do @echo %i
  - From a command line, echoes 1, 2, and 3.
  - The for command tries to interpret the items as file names and as patterns of file names containing wildcards.
  - It does not complain if the items do not match existing file names, though.
- for %i in (1,2,a*d*c*e*t) do @echo %i
  - Unless you happen to have a file matching the third pattern, echoes 1 and 2, discarding the third item.
- for %i in (1 2,3;4) do @echo %i
  - Echoes 1, 2, 3, and 4. Yes, a mixture of item separators is used.
- for %i in (*.txt) do @echo %i
  - Echoes file names of files located in the current folder and having the .txt extension.
- for %i in ("C:\Windows\system32\*.exe") do @echo %i
  - Echoes file names matching the pattern.
- for /r %i in (*.txt) do @echo %i
  - Echoes file names with full paths, of files having the extension .txt located anywhere in the current folder including nested folders.
- for /d %i in (*) do @echo %i
  - Echoes the names of all folders in the current folder

- for /r /d %i in (*) do @echo %i
  - Echoes the names including full paths of all folders in the current folder, including nested folders.
- for /r %i in (*) do @if %~zi geq 1000000 echo %~zi %i
  - For each file in the current folder and its subfolders that has the size greater than or equal to 1,000,000 bytes, outputs the file size in bytes and the full path of the file. For the syntax in %~zi, see #Percent tilde.
- for /l %i in (1,1,10) do @echo %i
  - Echoes the numbers from 1 to 10.
- for /f "tokens=*" %i in (list.txt) do @echo %i
  - For each line in a file, echoes the line.
- for /f "tokens=*" %i in (list1.txt list2.txt) do @echo %i
  - For each line in the files, echoes the line.
- for /f "tokens=*" %i in (*.txt) do @echo %i
  - Does nothing. Does not accept wildcards to match file names.
- for /f "tokens=1-3 delims=:" %a in ("First:Second::Third") do @echo %c-%b-%a
  - Parses a string into tokens delimited by ":".
  - The quotation marks indicate the string is not a file name.
  - The second and third tokens are stored in %b and %c even though %b and %c are not expressly mentioned in the part of the command before "do".
  - The two consecutive colons are treated as one separator; %c is not "" but rather "Third".
  - Does some of the job of the cut command from other operating systems.
- for /f "tokens=1-3* delims=:" %a in ("First:Second::Third:Fourth:Fifth") do @echo %c-%b-%a: %d
  - As above, just that the 4th and 5th items get captured in %d as "Fourth:Fifth", including the separator
- for /f "tokens=1-3* delims=:," %a in ("First,Second,:Third:Fourth:Fifth") do @echo %c-%b-%a: %d
  - Multiple delimiters are possible.
- for /f "tokens=1-3" %a in ("First Second Third,item") do @echo %c-%b-%a
  - The default delimiters are space and tab. Thus, they differ from the separators used to separate arguments passed to a batch.
- for /f "tokens=*" %i in ('cd') do @echo %i
  - For each line of the result of a command, echoes the line.
- for /f "tokens=*" %i in ('dir /b /a-d-h') do @echo %~nxai
  - For each non-hidden file in the current folder, displays the file attributes followed by the file name. In the string "%~nxai", uses the syntax described at #Percent tilde.
- for /f "usebackq tokens=*" %i in (`dir /b /a-d-h`) do @echo %~nxai
  - As above, but using the backquote character (`) around the command to be executed.
- for /f "tokens=*" %i in ('tasklist ^| sort ^& echo End') do @echo %i
  - Pipes and ampersands in the command to be executed must be escaped using caret (^).
- (for %i in (1,2,3) do @echo %i) > anyoldtemp.txt
  - To redirect the entire result of a for loop, place the entire loop inside brackets before redirecting. Otherwise, the redirection will tie to the body of the loop, so each new iteration of the body of the loop will override the results of the previous iterations.
- for %i in (1,2,3) do @echo %i > anyoldtemp.txt
  - An example related to the one above. It shows the consequence of failing to put the loop inside brackets.

Continue: To jump to the next iteration of the loop and thus emulate the continue statement known from many languages, you can use goto provided you put the loop body in a subroutine, as shown in the following:

```
for %%i in (a b c) do call :for_body %%i
exit /b
```

```
:for_body
    echo 1 %1
    goto :cont
    echo 2 %1
  :cont
exit /b
```

If you use goto directly inside the for loop, the use of goto breaks the loop bookkeeping. The following fails:

```
for %%i in (a b c) do (
    echo 1 %%i
    goto :cont
    echo 2 %%i
  :cont
    echo 3 %%i
)
```

Links:

- for at ss64.com
- for at Microsoft

FTYPE

Displays or sets the command to be executed for a file type. See also ASSOC.

Examples:

- ftype
  - Lists all associations of commands to be executed with file types, as, for example, 'Perl="C:\Perl\bin\perl.exe" "%1" %*'
- ftype | find "Excel.Sheet"
  - Lists only associations whose display line contains "Excel.Sheet"

Links:

- ftype at ss64.com
- ftype at Microsoft
- Making Python scripts run on Windows without specifying “.py” extension at stackoverflow

GOTO

Goes to a label.

An example:

```
goto :mylabel
echo Hello 1
REM Hello 1 never gets printed.

:mylabel
echo Hello 2
goto :eof

echo Hello 3
REM Hello 3 never gets printed. Eof is a virtual label standing for the end of file.
```

Goto within the body of a for loop makes cmd forget about the loop, even if the label is within the same loop body

Links:

- goto at ss64.com
- goto at Microsoft

IF

Conditionally executes a command. Documentation is available by entering IF /? to CMD prompt.

Available elementary tests:

- exist <filename>
- <string>==<string>
- <expression1> equ <expression2> -- equals
- <expression1> neq <expression2> -- not equal
- <expression1> lss <expression2> -- less than
- <expression1> leq <expression2> -- less than or equal
- <expression1> gtr <expression2> -- greater than
- <expression1> geq <expression2> -- greater than or equal
- defined <variable>
- errorlevel <number>
- cmdextversion <number>

To each elementary test, "not" can be appliedApparently there are no operators like AND, OR, etc. to combine elementary tests.

The /I switch makes the == and equ comparisons ignore case.

An example:

```
if not exist %targetpath% (
  echo Target path not found.
  exit /b
)
```

Examples:

- if not 1 equ 0 echo Not equal
- if 1 equ 0 echo A & echo B
  - Does nothing; both echo commands are subject to the condition.
- if not 1 equ 0 goto :mylabel
- if not a geq b echo Not greater
- if b geq a echo Greater
- if b geq A echo Greater in a case-insensitive comparison
- if B geq a echo Greater in a case-insensitive comparison
- if 0 equ 00 echo Numerical equality
- if not 0==00 echo String inequality
- if 01 geq 1 echo Numerical comparison
- if not "01" geq "1" echo String comparison
- if 1 equ 0 (echo Equal) else echo Unequal
  - Notice the brackets around the positive then-part to make it work.
- if not a==A echo Case-sensitive inequality
- if /i a==A echo Case-insensitive equality
- if /i==/i echo This does not work
- if "/i"=="/i" echo Equal, using quotation marks to prevent the literal meaning of /i

Links:

- if at ss64.com
- if at Microsoft

MD

Creates a new directory or directories. Has a synonym MKDIR; see also its antonym RD.

Examples:

- md Dir
  - Creates one directory in the current directory
- md Dir1 Dir2
  - Creates two directories in the current directory
- md "My Dir With Spaces"
  - Creates a directory with a name containing spaces in the current directory

Links:

- md at ss64.com
- md at Microsoft

MKDIR

A synonym for MD.

MKLINK

Makes a symbolic link or other type of link. Available since Windows Vista.

Links:

- mklink at ss64.com
- mklink at Microsoft

MOVE

Moves files or directories between directories, or renames them. See also REN.

Examples:

- move File1.txt File2.txt
  - Renames File1.txt to File2.txt, overwriting File2.txt if confirmed by the user or if run from a batch script.
- move File.txt Dir
  - Moves File.txt file into Dir directory assuming File.txt is a file and Dir is a directory; overwrites target file Dir\a.txt if conditions for overwriting are met.
- move Dir1 Dir2
  - Renames directory Dir1 to Dir2, assuming Dir1 is a directory and Dir2 does not exist.
- move Dir1 Dir2
  - Moves directory Dir1 into Dir2, resulting in existence of Dir2\Dir1, assuming both Dir1 and Dir2 are existing directories.
- move F:\File.txt
  - Moves the file to the current directory
- move F:\*.txt
  - Moves the files located at F:\ and ending in dot txt into the current directory, assuming the current directory is not F:\.

Links:

- move at ss64.com
- move at Microsoft

PATH

Outputs or sets the value of the PATH environment variable. When outputing, includes "PATH=" at the beginning of the output.

Examples:

- path
  - Outputs the PATH. An example output:
    - PATH=C:\Windows\system32;C:\Windows;C:\Program Files\Python27
- path C:\Users\Joe Hoe\Scripts;%path%
  - Extends the path with C:\Users\Joe Hoe\Scripts, applying only to the process of the cmd.exe.
- path ;
  - Empties the path.
- echo %path% | perl -pe "s/;/\n/g" | sort
  - Shows the folders in the path sorted if you have perl installed.

Links:

- path at ss64.com
- path at Microsoft

PAUSE

Prompts the user and waits for a line of input to be entered.

Links:

- pause at SS64.com
- pause at Microsoft

POPD

Changes to the drive and directory popped from the directory stack. The directory stack is filled using the PUSHD command.

Links:

- popd at ss64.com
- popd at Microsoft

PROMPT

Can be used to change or reset the cmd.exe prompt. It sets the value of the PROMPT environment variable.

```
C:\>PROMPT MyPrompt$G

MyPrompt>CD
C:\

MyPrompt>PROMPT

C:\>
```

The PROMPT command is used to set the prompt to "MyPrompt>". The CD shows that the current directory path is "C:\". Using PROMPT without any parameters sets the prompt back to the directory path.

Links:

- prompt at ss64.com
- prompt at Microsoft

PUSHD

Pushes the current directory onto the directory stack, making it available for the POPD command to retrieve, and, if executed with an argument, changes to the directory stated as the argument.

Links:

- pushd at ss64.com
- pushd at Microsoft

RD

Removes directories. See also its synonym RMDIR and antonym MD. Per default, only empty directories can be removed. Also type "rd /?".

Examples:

- rd Dir1
- rd Dir1 Dir2
- rd "My Dir With Spaces"
- rd /s Dir1
  - Removes the directory Dir1 including all the files and subdirectories in it, asking for confirmation once before proceeding with the removal. To delete files recursively in nested directories with a confirmation per file, use DEL with /s switch.
- rd /q /s Dir1
  - Like above, but without asking for confirmation.

Links:

- rd at ss64.com
- rd at Microsoft

REN

Renames files and directories.

Examples:

- ren filewithtpyo.txt filewithtypo.txt
- ren *.cxx *.cpp

Links:

- ren at ss64.com
- ren at Microsoft
- How does the Windows RENAME command interpret wildcards?, superuser.com

RENAME

This is a synonym of REN command.

REM

Used for remarks in batch files, preventing the content of the remark from being executed.

An example:

```
REM A remark that does not get executed
echo Hello REM This remark gets displayed by echo
echo Hello & REM This remark gets ignored as wished
:: This sentence has been marked as a remark using double colon.
```

REM is typically placed at the beginning of a line. If placed behind a command, it does not work, unless preceded by an ampersand, as shown in the example above.

Double colon is an alternative to REM. It can cause trouble when used in the middle of sequences in parentheses, like those used in FOR loops. The double colon seems to be just a trick, a label that starts with a colon.

Links:

- rem at ss64.com
- rem at Microsoft
- Which comment style should I use in batch files?, stackoverflow.com

RMDIR

This is a synonym of RD.

SET

Displays or sets environment variables. With /P switch, it asks the user for input, storing the result in the variable. With /A switch, it performs simple arithmetic calculations, storing the result in the variable. With string assignments, there must be no spaces before and after the equality sign; thus, "set name = Peter" does not work, while "set name=Peter" does.

Examples:

- set
  - Displays a list of environment variables
- set HOME
  - Displays the values of the environment variables whose names start with "HOME"
- set MYNUMBER=56
- set HOME=%HOME%;C:\Program Files\My Bin Folder
- set /P user_input=Enter an integer:
- set /A result = 4 * (6 / 3)
  - Sets the result variable with the result of a calculation. See also #Calculation.

Links:

- set at ss64.com
- set at Microsoft

SETLOCAL

When used in a batch file, makes all further changes to environment variables local to the current batch file. When used outside of a batch file, does nothing. Can be ended using ENDLOCAL. Exiting a batch file automatically calls "end local". Can be used to create subprograms: see Functions.

Furthermore, can be used to *enable delayed expansion* like this: "setlocal EnableDelayedExpansion". Delayed expansion consists in the names of variables enclosed in exclamation marks being replaced with their values only after the execution reaches the location c their use rather than at an earlier point.

The following is an example of using delayed expansion in a script that prints the specified number of first lines of a file, providing some of the function of the command "head" known from other operating systems:

```
@echo off

call :myhead 2 File.txt
exit /b

:: Function myhead
:: ===============
:: %1 - lines count, %2 - file name
:myhead
setlocal EnableDelayedExpansion
set counter=1
for /f "tokens=*" %%i in (%2) do (
  echo %%i
  set /a counter=!counter!+1
  if !counter! gtr %1 exit /b
)
exit /b
```

Links:

- setlocal at ss64.com
- EnableDelayedExpansion at ss64.com
- setlocal at Microsoft

SHIFT

Shifts the batch file arguments along, but does not affect %*. Thus, if %1=Hello 1, %2=Hello 2, and %3=Hello 3, then, after SHIFT, %1=Hello 2, and %2=Hello 3, but %* is "Hello 1" "Hello 2" "Hello 3".

Links:

- shift at ss64.com
- shift at Microsoft

START

Starts a program in new window, or opens a document. Uses an unclear algorithm to determine whether the first passed argument is a window title or a program to be executed; hypothesis: it uses the presence of quotes around the first argument as a hint that it is a window title.

Examples:

- start notepad.exe & echo "Done."
  - Starts notepad.exe, proceeding to the next command without waiting for finishing the started one. Keywords: asynchronous.
- start "notepad.exe"
  - Launches a new console window with notepad.exe being its title, apparently an undesired outcome.

- start "" "C:\Program Files\Internet Explorer\iexplore.exe"
  - Starts Internet Explorer. The empty "" passed as the first argument is the window title of a console that actually does not get opened, or at least not visibly so.
- start "C:\Program Files\Internet Explorer\iexplore.exe"
  - Launches a new console window with "C:\Program Files\Internet Explorer\iexplore.exe" being its title, apparently an undesired outcome.
- start /wait notepad.exe & echo "Done."
  - Starts notepad.exe, waiting for it to end before proceeding.
- start /low notepad.exe & echo "Done."
  - As above, but starting the program with a low priority.
- start "" MyFile.xls
  - Opens the document in the program assigned to open it.
- start
  - Starts a new console (command-line window) in the same current folder.
- start .
  - Opens the current folder in Windows Explorer.
- start ..
  - Opens the parent folder in Windows Explorer.
- start "" "mailto:"
  - Starts the application for writing a new email.
- start "" "mailto:joe.hoe@hoemail.com?subject=Notification&body=Hello Joe, I'd like to..."
  - Starts the application for writing a new email, specifying the to, subject and body of the new email.
- start "" "mailto:joe.hoe@hoemail.com?subject=Notification&body=Hello Joe,%0a%0aI'd like to..."
  - As above, with newlines entered as %0a.
- start /b TODO:example-application-where-this-is-useful
  - Starts the application without opening a new console window, redirecting the output to the console from which the start command was called.

Links:

- start at ss64.com
- start at Microsoft
- How to use command line switches to create a pre-addressed e-mail message in Outlook, support.microsoft.com

TIME

Displays or sets the system time.

Links:

- time at ss64.com
- time at Microsoft

TITLE

Sets the title displayed in the console window.

Links:

- title at ss64.com

- title at Microsoft

TYPE

Prints the content of a file or files to the output.

Examples:

- type filename.txt
- type a.txt b.txt
- type *.txt
- type NUL > tmp.txt
  - Create an empty file (blank file).

Links:

- type at ss64.com
- type at Microsoft

VER

Shows the command processor or operating system version.

```
C:\>VER

Microsoft Windows XP [Version 5.1.2600]

C:\>
```

Some version strings:

- Microsoft Windows [Version 5.1.2600]
  - For Windows XP
- Microsoft Windows [Version 6.0.6000]
  - For Windows Vista
- ...

The word "version" appears localized.

Links:

- ver at ss64.com
- ver at Microsoft
- Operating System Version at Microsoft
- List of Microsoft Windows versions, wikipedia.org
- Windows Build Numbers, eddiejackson.net
- mxpv/windows_build_numbers.txt, gist.github.com

VERIFY

Sets or clears the setting to verify whether COPY files etc. are written correctly.

Links:

- verify at ss64.com
- verify at Microsoft

VOL

Displays volume labels.

Links:

- vol at ss64.com
- vol at Microsoft

External commands

External commands available to Windows command interpreter are separate executable program files, supplied with the operating system by Microsoft, or bundled as standard with the third-party command interpreters. By replacing the program files, the meanings and functions of these commands can be changed.

Many, but not all, external commands support the "/?" convention, causing them to write on-line usage information to their standard output and then to exit with a status code of 0.

ARP

Displays or changes items in the address resolution protocol cache, which maps IP addresses to physical addresses.

Links:

- arp at ss64.com
- at arp Microsoft

AT

Schedules a program to be run at a certain time. See also SCHTASKS.

Links:

- at at ss64.com
- at at Microsoft

ATTRIB

Displays or sets file attributes. With no arguments, it displays the attributes of all files in the current directory. With no attribute modification instructions, it displays the attributes of the files and directories that match the given search wildcard specifications. Similar to chmod of other operating systems.

Modification instructions:

- To add an attribute, attach a '+' in front of its letter.
- To remove an attribute, attach a '-' in front of its letter
- Attributes:
  - A - Archived
  - H - Hidden
  - S - System
  - R - Read-only
  - ...and possibly others.

Examples:

- attrib

    - Displays the attributes of all files in the current directory.
- attrib File.txt
    - Displays the attributes of the file.
- attrib +r File.txt
    - Adds the "Read-only" attribute to the file.
- attrib -a File.txt
    - Removes the "Archived" attribute from the file.
- attrib -a +r File.txt
    - Removes the "Archived" attribute and adds the "Read-only" attribute to the file.
- attrib +r *.txt
    - Acts on a set of files.
- attrib /S +r *.txt
    - Acts recursively in subdirectories.

For more, type "attrib /?".

Links:

- attrib at ss64.com
- attrib at Microsoft

BCDEDIT

(Not in XP). Edits Boot Configuration Data (BCD) files. For more, type "bcdedit /?".

Links:

- bcdedit at ss64.com
- at Microsoft

CACLS

Shows or changes discretionary access control lists (DACLs). See also ICACLS. For more, type "cacls /?".

Links:

- cacls at ss64.com
- cacls at Microsoft

CHCP

Displays or sets the active code page number. For more, type "chcp /?".

Links:

- chcp at ss64.com
- chcp at Microsoft

CHKDSK

Checks disks for disk problems, listing them and repairing them if wished. For more, type "chkdsk /?".

Links:

- chkdsk at ss64.com
- chkdsk at Microsoft

CHKNTFS

Shows or sets whether system checking should be run when the computer is started. The system checking is done using Autochk.exe. The "NTFS" part of the command name is misleading, since the command works not only with NTFS file system but also with FAT and FAT32 file systems. For more, type "chkntfs /?".

Links:

- chkntfs at ss64.com
- chkntfs at Microsoft

CHOICE

Lets the user choose one of multiple options by pressing a single key, and sets the error level as per the chosen option. Absent in Windows 2000 and Windows XP, it was reintroduced in Windows Vista, and has remained in Windows 7 and 8.

Examples:

- choice /m "Do you agree"
  - Presents the user with a yes/no question, setting the error level to 1 for yes and to 2 for no. If the user presses Control + C, the error level is 0.
- choice /c rgb /m "Which color do you prefer"
  - Presents the user with a question, and indicates the letters for the user. Responds to user pressing r, g or b, setting the error level to 1, 2 or 3.

An alternative is "set /p"; see SET.

Links:

- choice at ss64.com
- choice at Microsoft

CIPHER

Shows the encryption state, encrypts or decrypts folders on a NTFS volume.

Links:

- cipher at ss64.com
- cipher at Microsoft

CLIP

(Not in XP, or make a copy from Server 2003) Places the piped input to the clipboard.

Examples:

- set | clip
  - Places the listing of environment variables to the clipboard.
- clip < File1.txt
  - Places the content of File1.txt to the clipboard.

Links:

- clip at ss64.com
- clip at Microsoft

CMD

Invokes another instance of Microsoft's CMD.

Links:

- cmd at ss64.com
- cmd at Microsoft

COMP

Compares files. See alsoFC.

Links:

- comp at ss64.com
- comp at Microsoft

COMPACT

Shows or changes the compression of files or folders on NTFS partitions.

Links:

- compact at Microsoft

CONVERT

Converts a volume from FAT16 or FAT32 file system to NTFS file system.

Links:

- convert at ss64.com
- convert at Microsoft

DEBUG

Allows to interactively examine file and memory contents in assembly language, hexadecimal or ASCII. Available in 32-bit Windows including Windows 7; the availability in 64-bit Windows is unclear. In modern Windows, useful as a quick hack to view hex content of a file. Keywords: hex dump, hexdump, hexadecimal dump, view hex, view hexadecimal, disassembler

Debug offers its own command line. Once on its ommand like, type "?" to find about debug commands.

To view hex of a file, invoke debug.exe with the file name as a parameter, and then repeatedly type "d" followed by enter on the debug command line.

Limitations:

- Being a DOS program, debug chokes on long file names. Use dir /x to find the 8.3 file name, and apply debug on that one.
- Debug cannot view larger files.

Links:

- Debug at technet.microsoft.com
- Debug for MS-DOS at technet.microsoft.com
- W:Debug (command)

DISKCOMP

Compares the content of two floppies.

Links:

- diskcomp at ss64.com
- diskcomp at Microsoft

DISKCOPY

Copies the content of one floppy to another.

Links:

- diskcopy at ss64.com
- diskcopy at Microsoft

DISKPART

Shows and configures the properties of disk partitions.

Links:

- diskpart at ss64.com
- diskpart at Microsoft, for XP
- diskpart at Microsoft

DOSKEY

Above all, creates macros known from other operating systems as aliases. Moreover, provides functions related to command history, and enhanced command-line editing. Macros are an alternative to very short batch scripts.

Macro-related examples:

- doskey da=dir /s /b
  - Creates a single macro called "da"
- doskey np=notepad $1
  - Creates a single macro that passes its first argument to notepad.
- doskey /macrofile=doskeymacros.txt
  - Loads macro definitions from a file.
- doskey /macros
  - Lists all defined macros with their definitions.
- doskey /macros | find "da"
  - Lists all macro definitions that contain "da" as a substring; see also FIND.

Command history-related examples:

- doskey /history
  - Lists the complete command history
- doskey /history | find "dir"
  - Lists each line of command history that contains "dir" as a substring
- doskey /listsize=100
  - Sets the size of command history to 100.

To get help on doskey from command line, type "doskey /?".

Links:

- doskey at ss64.com
- doskey at Microsoft

DRIVERQUERY

Shows all installed device drivers and their properties.

Links:

- driverquery at ss64.com
- driverquery at Microsoft

EXPAND

Extracts files from compressed .cab cabinet files. See also #MAKECAB.

Links:

- expand at ss64.com
- expand at Microsoft

FC

Compares files, displaying the diferences in their content in a peculiar way

Examples:

- fc File1.txt File2.txt >NUL && Echo Same || echo Different or error
  - Detects difference using the error level of fc.The error level of zero means the files are the same; non-zero can mean the files differ but also that one of the files does not exist.

Links:

- fc at ss64.com
- fc at Microsoft

FIND

Searches for a string in files or input, outputting matching lines. Unlike FINDSTR, it cannot search folders recursively, cannot search for a regular expression, requires quotation marks around the sought string, and treats space literally rather than as a logical or

Examples:

- find "(object" *.txt

- dir /S /B | find "receipt"
- dir /S /B | find /I /V "receipt"
  - Prints all non-matching lines in the output of the dir command, ignoring letter case.
- find /C "inlined" *.h
  - Instead of outputting the matching lines, outputs their count. If more than one file is searched, outputs one count number per file preceded with a series of dashes followed by the file name; does not output the total number of matching lines in all files.
- find /C /V "" < file.txt
  - Outputs the number of lines AKA line count in "file.txt". Does the job of "wc -l" of other operating systems. Works by treating "" as a string not found on the lines. The use of redirection prevents the file name from being output before the number of lines.
- type file.txt | find /C /V ""
  - Like the above, with a diferent syntax.
- type *.txt 2>NUL | find /C /V ""
  - Outputs the sum of line counts of the files ending in ".txt" in the current folder. The "2>NUL" is a redirection of standard error that removes the names of files followed by empty lines from the output.
- find "Schönheit" *.txt
  - If run from a batch file saved in unicode UTF-8 encoding, searches for the search term "Schönheit" in UTF-8 encoded *.txt files. For this to work, the batch file must not contain the byte order mark written by Notepad when saving in UTF-8. Notepad++ is an example of a program that lets you write UTF-8 encoded plain text files without byte order mark. While this works with find command, it does not work with #FINDSTR.
- find "Copyright" C:\Windows\system32\a*.exe
  - Works with binary files no less than text files.

Links:

- find at ss64.com
- find at Microsoft

FINDSTR

Searches for regular expressions or text strings in files. Does some of the job of "grep" command known from other operating systems, but is much more limited in the regular expressions it supports.

Treats space in a regular expression as a disjunction AKA logical or unless prevented with /c option.

Examples:

- findstr /s "[0-9][0-9].*[0-9][0-9]" *.h *.cpp
  - Searches recursively all files whose name ends with dot h or dot cpp, printing only lines that contain two consecutive decimal digits followed by anything followed by two consecutive decimal digits.
- findstr "a.*b a.*c" File.txt
  - Outputs all lines in File.txt that match any of the *two* regular expressions separated by the space. Thus, the effect is one of *logical or* on regular expressions.
- echo world | findstr "hello wo.ld"
  - Does not match. Since the 1st item before the space does not look like a regex, findstr treats the whole search term as a plain search term.
- echo world | findstr /r "hello wo.ld"
  - Matches. The use of /r forces regex treatment.
- findstr /r /c:"ID: *[0-9]*" File.txt
  - Outputs all lines in File.txt that match the single regular expression containing a space. The use of /c prevents the space from being treated as a logical or. The use of /r switches the regular expression treatment on, which

was disabled by default by the use of /c. To test this, try the following:

    - echo ID: 12|findstr /r /c:"ID: *[0-9]*$"
        - Matches.
    - echo ID: 12|findstr /c:"ID: *[0-9]*$"
        - Does not match, as the search string is not interpreted as a regular expression.
    - echo ID: abc|findstr "ID: *[0-9]*$"
        - Matches despite the output of echo failing to match the complete regular expression: the search is interpreted as one for lines matching "ID:" *or* "*[0-9]*$".
- findstr /ric:"id: *[0-9]*" File.txt
    - Does the same as the previous example, but in a case-insensitive manner
    - While findstr enables this sort of accumulation of switches behind a single "/", this is not possible with any command. For instance, "dir /bs" does not work, while "dir /b /s" does.
    - To test this, try the following:
        - echo ID: 12|findstr /ric:"id: *[0-9]*$"
        - echo ID: ab|findstr /ric:"id: *[0-9]*$"
- findstr /msric:"id: *[0-9]*" *.txt
    - Like above, but recursively for all files per /s, displaying only matching files rather than matching lines per /m.
- echo hel lo | findstr /c:"hel lo" /c:world
    - /c switch can be used multiple times to create logical or
- echo \hello\ | findstr "\hello\"
    - Does not match. Backslash before quotation marks and multiple other characters acts as an escape; thus, \" matches ".
- echo \hello\ | findstr "\\hello\\"
    - Matches. Double backslash passed to findstr stands for a single backslash.
- echo \hello\ | findstr \hello\
    - Matches. None of the single backslashes passed to findstr is followed by a character on which the backslash acts as an escape.
- echo ^"hey | findstr \^"hey | more
    - To search for a quote (quotation mark), you need to escape it two times: once for the shell using caret (^), and once for findstr using backslash (\).
- echo ^"hey | findstr ^"\^"hey there^" | more
    - To search for a quote and have the search term enclosed in quotes as well, the enclosing quotes need to be escaped for the shell using caret (^).
- echo //comment line | findstr \//
    - If forward slash (/) is the 1st character in the search term, it needs to be escaped with a backslash (\). The escaping is needed even if the search term is enclosed in quotes.
- findstr /f:FileList.txt def.*():
    - Search in the files stated in FileList.txt, one file per line. File names in FileList.txt can contain spaces and do not need to be surrounded with quotation marks for this to work.
- findstr /g:SearchTermsFile.txt *.txt
    - Search for the search terms found in SearchTermsFile.txt, one search term per line. A space does not serve to separate two search terms; rather, each line is a complete search term. A line is matched if at least one of the search terms matches. If the first search term looks like a regex, the search will be a regex one, but if it looks like a plain search term, the whole search will be a plain one even if 2nd or later search terms look like regex.
- findstr /xlg:File1.txt File2.txt
    - Outputs set intersection: lines present in both files.
- findstr /xlvg:File2.txt File1.txt
    - Outputs set difference: File1.txt - File2.txt.

- findstr /m Microsoft C:\Windows\system32\*.com
  - Works with binary files no less than text files.

Limitations of the regular expressions of "findstr", as compared to "grep":

- No support of groups -- "\(", "\)".
- No support of greedy iterators -- "*?".
- No support of "zero or one of the previous" -- "?".
- And more.

Other limitations: There is a variety of limitations and strange behaviors as documented at What are the undocumented features and limitations of the Windows FINDSTR command?

Bugs:

- echo bb|findstr "bb baaaa"
  - Does not find anything in multiple Windows versions, but it should.

Also consider typing "findstr /?".

Links:

- findstr at ss64.com
- findstr at Microsoft
- What are the undocumented features and limitations of the Windows FINDSTR command? at StackOverflow

FORFILES

Finds files by their modification date and file name pattern, and executes a command for each found file. Is very limited, especially compared to the find command of other operating systems. Available since Windows Vista. For more, type "forfiles /?".

Examples:

- forfiles /s /d 06/10/2015 /c "cmd /c echo @fdate @path"
  - For each file in the current folder and its subfolders modified on 10 June 2015 or later, outputs the file modification date and full file path. The date format after /d is locale specific. Thus, allows to find most recently modified files. Keywords: most recently changed files.
- forfiles /m *.txt /s /d 06/10/2015 /c "cmd /c echo @fdate @path"
  - As above, but only for files ending in .txt.

Links:

- forfiles at ss64.com
- forfiles at Microsoft

FORMAT

Formats a disk to use Windows-supported file system such as FAT, FAT32 or NTFS, thereby overwriting the previous content of the disk. To be used with great caution.

Links:

- format at ss64.com
- format at Microsoft

FSUTIL

A powerful tool performing actions related to FAT and NTFS file systems, to be ideally only used by powerusers with an extensive knowledge of the operating systems.

Links:

- fsutil at ss64.com
- fsutil at Microsoft
  - Fsutil: behavior
  - Fsutil: dirty
  - Fsutil: file
  - Fsutil: fsinfo
  - Fsutil: hardlink
  - Fsutil: objectid
  - Fsutil: quota
  - Fsutil: reparsepoint
  - Fsutil: sparse
  - Fsutil: usn
  - Fsutil: volume

GPRESULT

Displays group policy settings and more for a user or a computer

Links:

- gpresult at ss64.com
- gpresult at Microsoft
- Wikipedia:Group Policy

GRAFTABL

Enables the display of an extended character set in graphics mode. Fore more, type "graftabl /?".

Links:

- graftabl at Microsoft

HELP

Shows command help.

Examples:

- help
  - Shows the list of Windows-supplied commands.
- help copy
  - Shows the help for COPY command, also available by typing "copy /?".

Links:

- help at ss64.com
- help at Microsoft

ICACLS

(Not in XP) Shows or changes discretionary access control lists (DACLs) of files or folders. See also CACLS. Fore more, type "icacls /?".

Links:

- icacls at ss64.com
- icacls at Microsoft

IPCONFIG

Displays Windows IP Configuration. Shows configuration by connection and the name of that connection (i.e. Ethernet adapter Local Area Connection) Below that the specific info pertaining to that connection is displayed such as DNS suffix and ip address and subnet mask.

Links:

- ipconfig at ss64.com
- ipconfig at Microsoft

LABEL

Adds, sets or removes a disk label.

Links:

- label at ss64.com
- label at Microsoft

MAKECAB

Places files into compressed .cab cabinet file. See also #EXPAND.

Links:

- makecab at ss64.com
- makecab at Microsoft

MODE

A multi-purpose command to display device status, configure ports and devices, and more.

Examples:

- mode
  - Outputs status and configuration of all devices, such as com3 and con.
- mode con
  - Outputs status and configuration of con device, the console in which the command interpreter is running.
- mode con cols=120 lines=20
  - Sets the number of columns and lines for the current console, resulting in window resizing, and clears the screen. The setting does not affect new console instances. Keywords: wide screen, wide window, screen size, window size, resize screen, resize window.
- mode 120, 20
  - As above: Sets the number of columns (120) and lines (20), resulting in window resizing, and clears the screen.
- mode con cols=120

    - Sets the number of columns for the current console, resulting in window resizing, and clears the screen. It seems to change the number of visible lines as well, but the total lines count of the console buffer seems unchanged.
- mode 120
    - As above: Sets the number of columns.
- mode con cp
    - Outputs the current code page of the console.
- mode con cp select=850
    - Sets the current code page of the console. For a list of code pages, see the linked Microsoft documentation below.
- mode con rate=31 delay=1
    - Sets the rate and delay for repeated entry of a character while a key is held pressed, of the console. The lower the rate, the fewer repetitions per second.

Links:

- mode at ss64.com
- mode at Microsoft

MORE

Displays the contents of a file or files, one screen at a time. When redirected to a file, performs some conversions, also depending on the used switches.

Examples:

- more Test.txt
- more *.txt
- grep -i sought.*string Source.txt | more /p >Out.txt
    - Taking the output of a non-Windows grep command that produces line breaks consisting solely of LF character without CR character, converts LF line breaks to CR-LF line breaks. CR-LF newlines are also known as DOS line breaks, Windows line breaks, DOS newlines, Windows newlines, and CR/LF line endings, as opposed to LF line breaks used by some other operating systems.
    - In some setups, seems to output gibberish if the input contains LF line breaks and tab characters at the same time.
    - In some setups, for the conversion, /p may be unneeded. Thus, "more" would convert the line breaks even without /p.
- more /t4 Source.txt >Target.txt
    - Converts tab characters to 4 spaces.
    - In some setups, tab conversion takes place automatically, even without the /t switch. If so, it is per default to 8 spaces.

Switch /e:

- The online documentation for "more" in Windows XP and Windows Vista does not mention the switch.
- The switch /e is mentioned in "more /?" at least in Windows XP and Windows Vista.
- Per "more /?", the switch is supposed to enable extended features listed at the end of "more /?" help such as showing the current row on pressing "=". However, in Windows XP and Windows Vista, that seems to be enabled by default even without /e.
- Hypothesis: In Windows XP and Windows Vista, /e does not do anything; it is present for compatibility reasons.

Links:

- more at ss64.com
- more at Microsoft, Windows XP
- more at Microsoft, Windows Server 2008, Windows Vista

NET

Provides various network services, depending on the command used. Available variants per command:

- net accounts
- net computer
- net config
- net continue
- net file
- net group
- net help
- net helpmsg
- net localgroup
- net name
- net pause
- net print
- net send
- net session
- net share
- net start
- net statistics
- net stop
- net time
- net use
- net user
- net view

Links:

- net at ss64.com
- net services overview at Microsoft, Windows XP
  - net computer at Microsoft
  - net group at Microsoft
  - net localgroup at Microsoft
  - net print at Microsoft
  - net session at Microsoft
  - net share at Microsoft
  - net use at Microsoft
  - net user at Microsoft
  - net view at Microsoft

OPENFILES

Performs actions pertaining to open files, especially those opened by other users over the network. The actions involve querying, displaying, and disconnecting. For more, type "openfiles /?".

Links:

- openfiles at ss64.com
- openfiles at Microsoft

PING

Syntax:

- PING /?

- PING address
- PING hostname

Send ICMP/IP "echo" packets over the network to the designated address (or the first IP address that the designated hostname maps to via name lookup) and print all responses received.

Examples:

- ping en.wikibooks.org
- ping 91.198.174.192
- ping http://en.wikibooks.org/
  - Does not work.

Links:

- ping at ss64.com
- ping at Microsoft

RECOVER

Recovers as much information as it can from damaged files on a defective disk.

Links:

- recover at ss64.com
- recover at Microsoft

REG

Queries or modifies Windows registry.

The first argument is one of the following commands: query, add, delete, copy, save, load, unload, restore, compare, export, import, and flags. To learn more about a command, follow it by /?, like reg query /?.

Links:

- reg at ss64.com
- reg at Microsoft

REPLACE

Replaces files in the destination folder with same-named files in the source folder.

Links:

- replace at ss64.com
- replace at Microsoft

ROBOCOPY

(Not in XP) Copies files and folders. See also XCOPY and COPY.

Links:

- robocopy at ss64.com
- robocopy at Microsoft

RUNDLL32

Runs a function available from a DLL. The available DLLs and their functions differ among Windows versions.

Examples:

- rundll32 sysdm.cpl,EditEnvironmentVariables
  - In some Windows versions, opens the dialog for editing environment variables.

Links:

- rundll32 at ss64.com
- at Microsoft
- rundll at robvanderwoude.com
- dx21.com - lists rundll32 examples

SC

Controls Windows services, supporting starting, stopping, querying and more. Windows services are process-like things. A Windows service is either hosted in its own process or it is hosted in an instance of svchost.exe process, often with multiple services in the same instance. Processor time use of a particular service can be found using freely downloadable Process Explorer from Sysinternals, by going to properties of a service and then Threads tab. Another command capable of controlling services is NET. TASKLIST can list hosted services using /svc switch.

Examples:

- sc start wuauserv
  - Starts wuauserv service.
- sc stop wuauserv
- sc query wuauserv
- sc query
  - Outputs information about all services.
- sc config SysMain start= disabled
  - Make sure SysMain service is disabled after start. SysMain is the SuperFetch service, causing repeated harddrive activity by trying to guess which programs to load into RAM in case they will be used, and loading them. Notice the mandatory lack of space before = and the mandatory space after =.

Links:

- sc at ss64.com
- Windows 7 Services at ss64.com
- sc at Microsoft

SCHTASKS

Schedules a program to be run at a certain time, more powerful than AT.

Links:

- schtasks at ss64.com
- schtasks at Microsoft

SETX

Like SET, but affecting the whole machine rather than the current console or process. Not available in Windows XP; available in Windows Vista and later.

Links:

- setx at ss64.com
- setx at Microsoft, Windows Server 2008, Windows Vista

SHUTDOWN

Shuts down a computer, or logs off the current user.

Links:

- shutdown at ss64.com
- shutdown at Microsoft

SORT

Sorts alphabetically, from A to Z or Z to A, case insensitive. Cannot sort numerically: if the input contains one integer per line, "12" comes before "9".

Examples:

- sort File.txt
  - Outputs the sorted content of File.txt.
- sort /r File.txt
  - Sorts in reverse order, Z to A.
- dir /b | sort

Links:

- sort at ss64.com
- sort at Microsoft

SUBST

Assigns a drive letter to a local folder, displays current assignments, or removes an assignment.

Examples:

- subst p: .
  - Assigns p: to the current folder
- subst
  - Shows all assignments previously made using subst.
- subst /d p:
  - Removes p: assignment.

Links:

- subst at ss64.com
- subst at Microsoft

SYSTEMINFO

Shows configuration of a computer and its operating system.

Links:

- systeminfo at ss64.com
- systeminfo at Microsoft

TASKKILL

Ends one or more tasks.

Examples:

- taskkill /im AcroRd32.exe
  - Ends all process with the name "AcroRd32.exe"; thus, ends all open instances of Acrobat Reader. The name can be found using tasklist.
- taskkill /f /im AcroRd32.exe
  - As above, but *forced*. Succeeds in ending some processes that do not get ended without /f.
- tasklist | find "notepad"

  taskkill /PID 5792
  - Ends the process AKA task with process ID (PID) of 5792; the assumption is you have found the PID using tasklist.

Links:

- taskkill at ss64.com
- taskkill at Microsoft

TASKLIST

Lists tasks, including task name and process id (PID).

Examples:

- tasklist | sort
- tasklist | find "AcroRd"
- tasklist | find /C "chrome.exe"
  - Displays the number of tasks named "chrome.exe", belonging to Google Chrome browser
- tasklist /svc | findstr svchost
  - Outputs Windows services hosted in svchost.exe processes alongside the usual information abot the process.

Links:

- tasklist at ss64.com
- tasklist at Microsoft

TIMEOUT

Waits a specified number of seconds, displaying the number of remaining seconds as time passes, allowing the user to interrupt the waiting by pressing a key. Also known as delay or sleep. Available in Windows Vista and later.

Examples:

- timeout /t 5

    - Waits for five seconds, allowing the user to cancel the waiting by pressing a key
- timeout /t 5 /nobreak
    - Waits for five seconds, ignoring user input other than Control + C.
- timeout /t 5 /nobreak >nul
    - As above, but with no output.

Workaround in Windows XP:

- ping -n 6 127.0.0.1 >nul
    - Waits for five seconds; the number after -n is the number of seconds to wait plus 1.

Perl-based workaround in Windows XP, requiring Perl installed:

- perl -e "sleep 5"
    - Waits for 5 seconds.

Links:

- timeout at ss64.com
- timeout at Microsoft
- How to wait in a batch script? at stackoverflow.com
- Sleeping in a batch file at stackoverflow.com

TREE

Displays a tree of all subdirectories of the current directory to any level of recursion or depth. If used with /F switch, displays not only subdirectories but also files.

Examples:

- tree
- tree /f
    - Includes files in the listing, in addition to directories.
- tree /f /a
    - As above, but uses 7-bit ASCII characters including "+", "-" and \" to draw the tree.

A snippet of a tree using 8-bit ASCII characters:

```
├───winevt
│   ├───Logs
│   └───TraceFormat
├───winrm
```

A snippet of a tree using 7-bit ASCII characters:

```
+---winevt
|   +---Logs
|   \---TraceFormat
+---winrm
```

Links:

- tree at Microsoft

WHERE

Outputs one or more locations of a file or a file name pattern, where the file or pattern does not need to state the extension if it listed in PATHEXT, such as .exe. Searches in the current directory and in the PATH by default. Does some of the job of "which" command of some other operating systems, but is more flexible.

Available on Windows 2003, Windows Vista, Windows 7, and later; not available on Windows XP. An alternative to be used with Windows XP is in the examples below

Does not find internal commands, as there are no dot exe files for them to match.

Examples:

- where find
  - Outputs the location of the find command, possibly "C:\Windows\System32\find.exe". The .exe extension does not need to be specified as long as it is listed in PATHEXT, which it is by default.
  - If there are more find commands in the path, outputs paths to both. In some situations, it can output the following:

    C:\Windows\System32\find.exe

    C:\Program Files\GnuWin32\bin\find.exe
- for %i in (find.exe) do @echo %~$PATH:i
  - Outputs the location of "find.exe" on Windows XP. The name has to include ".exe", unlike with the where command.
- where /r . Tasks*
  - Searches for files whose name matches "Task*" recursively from the current folder. Similar to "dir /b /s Tasks*". The /r switch disables search in the folders in PATH.
- where *.bat
  - Outputs all .bat files in the current directory and in the directories that are in PATH. Thus, outputs all .bat files that you can run without entering their full path.
- where ls*.bat
  - As above, constraining also the beginning of the name of the .bat files.
- where ls*
  - As above, but with no constraint on the extension. Finds lsdisks.bat, lsmice.pl, and lsmnts.py if in the current directory or in the path.
- where *.exe *.com | more
  - Displays countless .exe and .com files in the path and in the current folder, including those in C:\Windows\System32.
- where $path:*.bat
  - Outputs .bat files in the path but not those in the current folder unless the current folder is in PATH. Instead of path, another environment variable containing a list of directories can be used.
- where $windir:*.exe
  - Outputs .exe files found in the folder stated in WINDIR environment variable.
- where $path:*.bat $windir:*.exe
  - A combination is possible. Outputs all files matching either of the two queries.
- where /q *.bat && echo Found
  - Suppresses both standard and error output, but sets the error level, enabling testing on it. The error level is set either way, with or without /q.

Links:

- where at ss64.com
- where at Microsoft
- Is there an equivalent of 'which' on windows?

WMIC

Starts Windows Management Instrumentation Command-line (WMIC), or with arguments given, passes the arguments as commands to WMIC. Not in Windows XP Home. For more, type "wmic /?".

Examples:

- wmic logicaldisk get caption,description
  - Lists drives (disks) accessible under a drive letter, whether local hard drives, CD-ROM drives, removable flash drives, network drives or drives created using #SUBST.
- wmic
  Control + C
  - Enters wmic and then interrupts it. A side effect is that the console buffer becomes very wide, and the screen becomes horizontally resizable with the mouse as a consequence. This is the result of wmic setting a high number of columns of the console, which you can verify using *mode con*. You can achieve a similar result by typing *mode 1500*. See also #MODE.

Links:

- wmic at ss64.com
- wmic at Microsoft

XCOPY

Copies files and directories in a more advanced way than COPY, deprecated in Windows Vista and later. Type xcopy /? to learn more, including countless options.

Examples:

- xcopy C:\Windows\system
  - Copies all files, but not files in nested folders, from the source folder ("C:\Windows\system") to the current folder
- xcopy /s /i C:\Windows\system C:\Windows-2\system
  - Copies all files and folders to any nesting depth (via "/s") from the source folder ("C:\Windows\system") to "C:\Windows-2\system", creating "Windows-2\system" if it does not exist (via "/i").
- xcopy /s /i /d:09-01-2014 C:\Windows\system C:\Windows-2\system
  - As above, but copies only files changed on 1 September 2014 or later. Notice the use of the month-first convention even if you are on a non-US locale of Windows.
- xcopy /L /s /i /d:09-01-2014 C:\Windows\system C:\Windows-2\system
  - As above, but in a test mode via /L (list-only, output-only, display-only). Thus, does not do any actual copying, merely lists what would be copied.

Links:

- xcopy at ss64.com
- xcopy at Microsoft

External links

- Windows XP - Command-line reference A-Z at microsoft.com
- Windows Server 2008R2 - Command-Line Reference at microsoft.com
- Windows Server 2012R2 - Command-Line Reference at microsoft.com
- Windows Server 2016 - Windows Commands at microsoft.com
- Windows CMD Commands at ss64.com -- licensed under Creative Commons Attribution-Non-Commercial-Share Alike 2.0 UK: England & Wales[1], and thus incompatible with CC-BY-SA used by Wikibooks
- The FreeDOS HTML Help at fdos.org -- a hypertext help system for FreeDOS commands, written in 2003/2004, available under the GNU Free Documentation License

- Category:Batch File, rosettacode.org

Retrieved from "https://en.wikibooks.org/w/index.php?title=Windows_Batch_Scripting&oldid=3476953"