# Commands

## Prompt Control

### cls

**Usage**: Clear the screen.

**Syntax**: cls

### explorer

**Usage**: Open the UI explorer from the Prompt.

**Syntax**: explorer <any-directory>

### pause

**Usage**: Suspend the program and wait for the user to press any key to continue.

**Syntax**: pause

**Example**:

@echo off

echo Hello World

pause

Output:

C:\Users\ADMIN\Desktop\batch\_test>test.cmd

Hello World

Press any key to continue . . .

### exit

**Usage**: Terminate and exit the console.

**Syntax**: exit

### echo

**Usage:**

* **Displays messages** to the Prompt.
* Is added at the beginning of a bat file to either **show or hide the commands** from being printed at the Prompt.

**Syntax**: echo <msg-content>

**Examples**:

*1. @echo on*

@echo on

echo One

echo Two

Output:

C:\Users\ADMIN\Desktop\batch\_test>echo One

One

C:\Users\ADMIN\Desktop\batch\_test>echo Two

Two

C:\Users\ADMIN\Desktop\batch\_test>

*2. @echo off*

@echo off

echo One

echo Two

Output:

C:\Users\ADMIN\Desktop\batch\_test>test.cmd

One

Two

**Note**:

* By default, @echo on
* If the message following the echo command is embraced in a double quote, the output will also include that double quote. For example:

C:\Users\ADMIN\Desktop\batch\_test>echo "Hello World"

"Hello World"

**Tips**:

* To create a **new line**, use echo:

### title

**Usage**: Set the title displayed in the Prompt window.

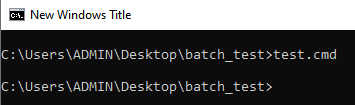
**Syntax**: title <title-name>

**Example**:

@echo off

title New Windows Title

Output:



### choice

**Usage**: Provide a list of options to the user.

**Syntax**: choice /c [choices] /m [message]

Example:

@echo off

choice /C yn /m "Do you want to continue?"

Output:

C:\Users\ADMIN\Desktop\batch\_test>test.cmd

Do you want to continue? [Y,N]?

The console will wait for the user input, and once you enter your answer ("y" or "n") it will continue running other scripts. If your answer is neither "y" nor "n", nothing happens.

For how to set the choice to variable, check [here](https://www.computerhope.com/issues/ch001674.htm).

## Command Helper

### help

**Usage**: Show command guideline.

**Syntax**:

* Shows the list of Windows-supplied commands: help
* Show guideline on how to use a particular command: help <any-command>

### rem (or ::)

**Usage**: Remarks in batch files. Lines following rem are not executed by the OS.

**Syntax:**

rem <any-content>

or:

:: <any-content>

## Directories and Files

### dir

**Usage**: List a list of names of files and subdirectories in a directory.

**Syntax**: dir [options] [directory]

**Common options**:

|  |  |
| --- | --- |
| /ah | Show hidden files and directories. |
| /o  <short-order> | Sort by:   * /on: name (alphabetic) * /os: size (smallest first) * /oe: extension (alphabetic) * /od: date/time (oldest first) * /og: Group directories first   **Tip:**   * Prefix to reverse order is "-". * To controls which time field displayed or used for sorting, add:   /tc: Creation  /ta: Last access  /tw: Last written |
| /s | List files and subdirectories recursively. |

**Example:**

C:\Users\ADMIN\Desktop>dir /og

Volume in drive C has no label.

Volume Serial Number is 8CFB-CB27

Directory of C:\Users\ADMIN\Desktop

05/30/2020 05:12 PM <DIR> .

05/30/2020 05:12 PM <DIR> ..

01/30/2020 11:18 PM <DIR> Folder 1

02/17/2020 09:20 PM <DIR> Folder 2

12/15/2017 05:49 PM 2,752 My Excel.xlsx

01/22/2020 08:17 PM 5,423,284 My PDF.pdf

### copy

**Usage**: Make copies of files (not directories). Wildcards may be used to copy multiple files.

**Syntax**: copy [options] <file> <destination-dir>

**Example**:

1. copy picture.jpg picture-02.jpg

-> Make a copy of picture.jpg, the new file is named as picture-02.jpg. Both files now exist in the same working directory.

2. copy "D:\pictures\my picture.jpg" D:\backup\

-> Make a copy of my picture.jpg in D:\pictures\. The new file is also named my picture.jpg but located in D:\backup.

**Common options**:

|  |  |
| --- | --- |
| /y | Silently override the existing destination file without waiting for your confirm. |

**Tip:** The command copy can concatenate files. For example, copy file1 + file2 file3 creates a file (file3) which contains file1's and file2's contents.

**Note**: The command copy only deals with file. For copying directories, use command [xcopy](#_xcopy).

### xcopy

**Usage**: Make copies of files (by default) or directories (with option). This is the advanced version of the command copy.

**Syntax**: xcopy [options] <file-or-dir> <destination-dir>

**Common options**:

|  |  |
| --- | --- |
| /y | Silently override the existing destination file without waiting for your confirm. |
| /s | Copies directories and subdirectories except empty ones. |
| /e | Copies directories and subdirectories, including empty ones. |
| /h | Copies hidden and system files also. |
| /w | Prompts you to press a key before copying. |
| /EXCLUDE:file1[+file2][+file3]... | Specifies a list of files containing strings. Each string should be in a separate line in the files. When any of the strings match any part of the absolute path of the file to be copied, that file will be excluded from being copied.  For example, specifying a string like \obj\ or obj\ will exclude all files underneath the directory obj, or .obj will exclude all files with the .obj extension.  **Note**:   * After /EXCLUDE, it **has to be a path to a text file or list of text files**. Harding paths is not allowed. * That path has to be **absolute**. Cannot be relative path. So use %cd% to get the current directory. |

**Differences between xcopy and copy:**

* xcopy can deal with folders, while copy can only deal with files.
* xcopy automatically creates folders (and sub-folders) if they don't exist, while copy outputs error.

C:\Users\ADMIN\Desktop\batch\_test>**copy list.txt a**

1 file(s) copied.

🡪 File 'a' is created with the same content as file 'list.txt'

C:\Users\ADMIN\Desktop\batch\_test>**copy list.txt a\**

The system cannot find the path specified.

0 file(s) copied.

🡪 Folder 'a' cannot be created automatically

C:\Users\ADMIN\Desktop\batch\_test>**xcopy list.txt a\**

C:list.txt

1 File(s) copied

🡪 Folder 'a' is created with a copy version of 'list.txt'

**Note:**

In some cases (I don't know exactly which cases), xcopy might prompt the message below:

C:\Users\ADMIN\Desktop\batch\_test>xcopy \e \y list.txt dest

Does dest specify a file name

or directory name on the target

(F = file, D = directory)?

To pick up an answer automatically, use:

* File: echo F | xcopy \e \y list.txt dest
* Directory: xcopy \e \y list.txt dest\

### move

**Usage**: Move or rename files and directories. Wildcards may be used to move multiple files.

**Syntax**: move [options] <source-dir-or-file> <destination-dir-or-file>

**Examples**:

1. Move file myfile.txt from the current directory into C:\Downloads: move myfile.txt C:\Downloads

2. Rename directory temp to non-temp: move D:\temp D:\non-temp

**Common options**:

|  |  |
| --- | --- |
| /y | Silently override the existing destination file without waiting for your confirm. |

### ren

**Usage**: Rename files and directories.

**Syntax**: ren [options] <source-dir-or-file> <destination-dir-or-file>

### del

**Usage**: Delete files (not directories). Wildcards may be used to delete multiple files.

**Syntax**: del [options] <file>

**Common options**:

|  |  |
| --- | --- |
| /p | Prompts for confirmation before deleting each file. |
| /f | Force deleting of read-only files. |
| /q | Quiet mode, do not ask if ok to delete on global wildcard |
| \* | Delete all files (not directories) in the working directory. |

**Note**: The command del D:\temp deletes all files in the directory temp, but not the temp itself. To delete directories, use command [rd](#_rd).

**Caution**: There is **NO Recycle Bin for del**. So cannot recover the deleted files. For alternative for del, but support Recycle Bin, check [here](https://stackoverflow.com/a/1646493).

### rd

**Usage**: Delete empty directories (by default), or directories and its subdirectories and files (with option).

**Syntax**: rd [options] <dir>

**Common options**:

|  |  |
| --- | --- |
| /s | Removes all subdirectories and files in the specified directory in addition to the directory itself. |
| /q | Quiet mode, do not ask if ok to delete on global wildcard |

### tree

**Usage**: List the contents of directories (and files) in a tree-like format. It's a really neat and useful to view the structure of directories.

**Syntax**: tree [options] [dir]

**Common options**:

|  |  |
| --- | --- |
| /f | Display the names of the files in each directory. By default, only directories are listed. |

## Directories

### cd

**Usage**:

* Stand for "change directory". It changes the current working directory.
* Show full path of the current directory (when used without any parameter).

**Syntax**: cd [dir]

**Common options**:

|  |  |
| --- | --- |
| .. | Return you to the parent directory |
| / | Return you to the root directory (C: \) |

**Tip**: To get the current directory, use environment variable %cd%.

### md (or mkdir)

**Usage**: Create a new directory.

**Syntax**: md [options] <dir>

**Tip:**

Without any more option, the md command can create intermediate directories in the path, if needed. For example, assume \a does not exist then:

md \a\b

is the same as:

md \a

cd \a

md b

## Files

### type

**Usage**: Display the contents of a file.

**Syntax**: type [options] <file>

**Example**: Assume test.cmd has following content:

@echo off

echo Hello World

pause

Running type test.cmd gives following output:

C:\Users\ADMIN\Desktop\batch\_test>more test.cmd @echo off

echo Hello World

pause

### more

**Usage**: Display the contents of a file or files, one screen at a time. Compared to the type command, the more command is more advanced.

**Syntax**: more [options] <file>

**Example**: Assume test.cmd has following content:

@echo off

echo Hello World

pause

Running more test.cmd gives following output:

C:\Users\ADMIN\Desktop\batch\_test>more test.cmd @echo off

echo Hello World

pause

**Common Options**:

|  |  |
| --- | --- |
| /c | Clears the screen before displaying a page. |

## Text Manipulation

### find

**Usage**: Search for a string in files or input, outputting matching lines.

**Syntax**: find <search-pattern> [destination]

**Example**:

1. Find the word "Application" in list.txt: find "Application" C:\tp\lists.txt

2. Display only lines containing the word "Path" in the output of the command set:

C:\Users\ADMIN\Desktop\batch\_test>set | find "Path"

Path=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Git\cmd

PSModulePath=C:\Program Files\WindowsPowerShell\Modules

**Common options**:

|  |  |
| --- | --- |
| /i | Ignores the case of characters when searching for the string. |
| /v | Displays all lines NOT containing the specified string. |

## Process Management

### start

**Usage**: Run a specified program or command in a separate window.

**Syntax**: start <file-or-program>

**Common options**:

|  |  |
| --- | --- |
| /wait | Wait until the opened window is closed. |

**Example**:

1. Start Notepad: start notepad

2. Start a text file with the default text editor: start test.txt (assume test.txt is located in your working directory). To specify which text editor will be used, e.g. Notepad, run start notepad test.txt.

### call

**Usage**: Run a specified program or command in the same window.

**Syntax**: call <file-or-program>

**Example**:

**Note**: Differences between start and call: <https://stackoverflow.com/a/13258451>

### tasklist

**Usage**: List all the running tasks in the console.

**Syntax**: tasklist [option]

**Example**:

Running tasklist gives following output:

C:\Users\ADMIN\Desktop\batch\_test>tasklist

Image Name PID Session Name Session# Mem Usage

========================= ======== ================ =========== ============

System Idle Process 0 Services 0 8 K

System 4 Services 0 20 K

Registry 96 Services 0 37,704 K

smss.exe 348 Services 0 332 K

csrss.exe 540 Services 0 2,208 K

wininit.exe 616 Services 0 1,088 K

services.exe 688 Services 0 7,108 K

lsass.exe 708 Services 0 9,652 K

svchost.exe 812 Services 0 672 K

fontdrvhost.exe 832 Services 0 524 K

svchost.exe 852 Services 0 19,888 K

svchost.exe 76 Services 0 14,044 K

svchost.exe 504 Services 0 4,808 K

svchost.exe 1076 Services 0 4,228 K

svchost.exe 1220 Services 0 5,668 K

svchost.exe 1228 Services 0 9,352 K

svchost.exe 1300 Services 0 5,452 K

svchost.exe 1312 Services 0 2,212 K

svchost.exe 1456 Services 0 5,404 K

svchost.exe 1616 Services 0 64,584 K

svchost.exe 1624 Services 0 1,388 K

memory Compression 1668 Services 0 243,524 K

…

### taskkill

**Usage**: Terminate tasks by process id (PID) or image name.

**Syntax**: taskkill [option]

**Common options**:

|  |  |
| --- | --- |
| /pid | Specifies the PID of the process to be terminated.  Use taskList to get the PID. |
| /im | Specifies the image name of the process to be terminated.  Use taskList to get the PID.  Wildcard can be used to specify all tasks or image names. |
| /t | Terminates the specified process and any child processes which were started by it. |
| /f | Forcefully terminate the process(es). |

## System

### set

**Usage**: Display, set, or remove cmd.exe environment variables on the current system.

**Syntax**:

* Display list of current environment variables: set
* Set an environment variable to a specific value: set [option] <variable=[string]>

Note: There must be **NO space around the '=' notation**. Else, syntax error!

**Common options**:

|  |  |
| --- | --- |
| /a | set /a <expression>  The /a switch specifies that the string to the right of the equal sign is a **numerical** expression that is evaluated. The expression evaluator supports the following [operations](#_Operators), in decreasing order of precedence:  () grouping  \* / % arithmetic operators  + - arithmetic operators  << >> logical shift  & bitwise and  ˆ bitwise exclusive or  | bitwise or  = \*= /= %= += -=  &= ˆ= |= <<= >>= assignment  , expression separator |
| /p | set /p variable=[prompt-string]  The /p switch allows you to set the value of a variable to a line of **input entered by the user**.  Note: If nothing is entered, and only the Enter key is pressed, variable will remain unchanged instead of being cleared.  Example:  @echo off  echo Enter your name:  set /p name=  echo Your name is:  echo %name%  Output:  Enter your name:  Nhan Tri  Your name is:  Nhan Tri |

**Tip**:

* How to add a path only for a batch file executing? Example:

set devenvPath="C:\Program Files (x86)\Microsoft Visual Studio\2019\Professional\Common7\IDE\"

set path=%devenvPath%;%path%

...

devenv /Rebuild Debug myProject\proj.sln

### systeminfo

**Usage**: Show computer configuration and its OS.

**Syntax**: systeminfo [options]

**Example**:

C:\Users\ADMIN\Desktop\batch\_test>systeminfo

Host Name: NHANTRI

OS Name: Microsoft Windows 10 Pro

OS Version: 10.0.18362 N/A Build 18362

OS Manufacturer: Microsoft Corporation

OS Configuration: Standalone Workstation

OS Build Type: Multiprocessor Free

Registered Owner: Windows User

Registered Organization:

Product ID: 00331-10000-00001-AA779

Original Install Date: 1/27/2020, 11:52:51 PM

System Boot Time: 5/27/2020, 10:42:04 PM

System Manufacturer: ASUSTeK COMPUTER INC.

System Model: X550CC

System Type: x64-based PC

Processor(s): 1 Processor(s) Installed.

[01]: Intel64 Family 6 Model 58 Stepping 9 GenuineIntel ~1801 Mhz

BIOS Version: American Megatrends Inc. X550CC.213, 6/26/2013

Windows Directory: C:\WINDOWS

System Directory: C:\WINDOWS\system32

Boot Device: \Device\HarddiskVolume2

System Locale: en-us;English (United States)

Input Locale: en-us;English (United States)

Time Zone: (UTC+07:00) Bangkok, Hanoi, Jakarta

Total Physical Memory: 3,982 MB

Available Physical Memory: 1,171 MB

Virtual Memory: Max Size: 7,694 MB

Virtual Memory: Available: 3,657 MB

Virtual Memory: In Use: 4,037 MB

Page File Location(s): C:\pagefile.sys

Domain: WORKGROUP

Logon Server: \\NHANTRI

Hotfix(s): 2 Hotfix(s) Installed.

[01]: KB4528759

[02]: KB4528760

Network Card(s): 3 NIC(s) Installed.

[01]: TAP-Windows Adapter V9

Connection Name: Ethernet 3

Status: Media disconnected

[02]: Realtek PCIe GbE Family Controller

Connection Name: Ethernet 2

Status: Media disconnected

[03]: Qualcomm Atheros AR9485WB-EG Wireless Network Adapter

Connection Name: Wi-Fi 2

DHCP Enabled: Yes

DHCP Server: 192.168.1.1

IP address(es)

[01]: 192.168.1.5

[02]: fe80::2cb7:70a9:a5fb:8874

[03]: 2001:ee0:4c76:5250:5986:abf6:7c93:e695

[04]: 2001:ee0:4c76:5250:2cb7:70a9:a5fb:8874

Hyper-V Requirements: VM Monitor Mode Extensions: Yes

Virtualization Enabled In Firmware: Yes

Second Level Address Translation: Yes

Data Execution Prevention Available: Yes

### shutdown

**Usage**: Shut down the computer.

**Syntax**: shutdown [options]

**Common options**:

|  |  |
| --- | --- |
| /s | Shut down the computer. |
| /r | Shut down and restart the computer. |
| /h | Hibernate the computer. |
| /l | Log off the computer. |

## Network

### ipconfig

**Usage**: Displays information about IP configuration on the current machine

**Syntax**: ipconfig

**Example**:

C:\Users\ADMIN\Desktop\batch\_test>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet 2:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Wireless LAN adapter Local Area Connection\* 1:

Media State . . . . . . . . . . . : Media disconnected

Connection-specific DNS Suffix . :

Wireless LAN adapter Wi-Fi 2:

Connection-specific DNS Suffix . : Home

IPv6 Address. . . . . . . . . . . : 2001:ee0:4c76:5250:2cb7:70a9:a5fb:8874

Temporary IPv6 Address. . . . . . : 2001:ee0:4c76:5250:5986:abf6:7c93:e695

Link-local IPv6 Address . . . . . : fe80::2cb7:70a9:a5fb:8874%11

IPv4 Address. . . . . . . . . . . : 192.168.1.5

Subnet Mask . . . . . . . . . . . : 255.255.255.0

Default Gateway . . . . . . . . . : fe80::a6f4:c2ff:fec2:3b0a%11

192.168.1.1

### ping

**Usage**: Check whether the network connection is available or not

**Example**: Use ping [www.google.com](http://www.google.com/) to check if the computer can access to Google.

# Variables

## The 'set' Command

First take a look at [this section](#_set).

### Working with Strings

For example:

@echo off

set message=Hello World

echo %message%

Output:

Hello World

**Note**: To display the value of the variable, we need to enclose the variable in two % signs.

### Working with Numeric Values

For example:

@echo off

set /a a = 5

set /a b = 10

set /a c = %a% + %b%

echo %c%

Output:

15

## Local vs Global

**By default, variables are global** throughout the entire script. To make variables local to the scope of your script, enclose them in the setlocal and endlocal pair.

For example:

@echo off

set /a globalVar = 1

setlocal

set /a localVar = 2

echo %localVar%

echo %globalVar%

endlocal

echo %localVar%

Output:

2

1

ECHO is off.

## Environment Variables

If you have variables that would be used across batch files, then it is always preferable to use environment variables. Once the environment variable is defined, it can be accessed via the % sign. The following example shows how to see the JAVA\_HOME defined on a system. This variable is a key component that is normally used by a wide variety of applications.

@echo off

echo %JAVA\_HOME%

The output would show the JAVA\_HOME directory which would depend from system to system. Following is an example of an output.

C:\Program Files (x86)\Java\jre1.8.0\_211\lib

# Strings

<https://www.tutorialspoint.com/batch_script/batch_script_strings.htm>

# Arrays

<https://www.tutorialspoint.com/batch_script/batch_script_arrays.htm>

# Operators

## Comparison

### Strings

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | equal to  It runs a string comparison of the two arguments on equality, i.e. use strcmp with condition being true on strcmp returning 0. |
| not | not equal to  In combination with == to invert the result of the string comparison on equality, i.e. the condition is true if the two compared strings are not equal. |

**Example:**

The command:

if "19"=="3" echo My computer doesn't know Maths

runs strcmp with the strings "19" and "3", which means the compared byte streams are hexadecimal 22 31 39 22 00 and 22 33 22 00. The double quotes are not removed before running the string comparison. The quotes are included in the string comparison.

**Tip**: There is the option /I to compare the two arguments **case-insensitive** using stricmp instead of strcmp. For example:

if /I not "%~1" == "/I" echo First argument is neither /i nor /I.

### Numbers

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

In addition, the keyword not is used to negate a condition.

**Notes**:

* An integer comparison is done if that is successful for both argument strings because the two compared strings are:
* **Decimal numbers** with first character being optionally - or + and all other characters are decimal digits 0123456789 with first digit not being 0 like -2147483648, -200, +10, 32, 2147483647, or
* **Hexadecimal numbers** with first character being optionally - or + and next with 0x or 0X and all other characters are hexadecimal digits 0123456789ABCDEFabcdef like -0x80000000, -0XC8, +0x0a, 0x20, 0x7fffFFFF, or
* **Octal numbers** with first character being optionally - or + and next 0 and all other characters are octal digits 01234567 like -020000000000, -0310, +012, 040, 017777777777.

For examples, two below conditions are true:

if 014 EQU 12 echo Octal number 014 is equal decimal number 12.

if 0x0C EQU 12 echo Hexadecimal number 0C is equal decimal number 12.

* The angle brackets < and > are used on Windows command line as redirection operators. So they **can't be used as comparison operators** on an if condition. Also, the exclamation mark ! is not available as operator because it means begin/end of an environment variable reference on having delayed environment variable expansion enabled.

# Conditions

## 'if'

**Syntax**:

if <condition> (

...

)

**Note**:

* The evaluation is **case-sensitive**.
* The keyword 'if' and the sign '(' must be **on the same line**. Otherwise, syntax error!

**Example**:

@echo off

set /a num=5

set str=hello

if %num%==5 (

    echo The value of variable num is 5

)

if %str%==hello (

    echo The value of variable str is "hello"

)

Output:

The value of variable num is 5

The value of variable str is "hello"

## 'if … else'

**Syntax**:

if <condition> (

...

) else (

...

)

**Note**:

* The sign ')', the keyword 'else' and the sign '(' must be **on the same line**. Otherwise, syntax error!

## Special Cases

### 'if defined'

**Usage**: Test for the existence of a variable.

**Example**:

@echo off

set str1=String1

if defined str1 (

    echo Variable str1 is defined

)

if defined str2 (

    echo Variable str2 is defined

) else (

    echo Variable str2 is not defined

)

**Output**:

Variable str1 is defined

Variable str2 is not defined

### 'if exists'

**Usage**: Test for the existence of a file.

**Example**:

@echo off

if exist C:\Windows\explorer.exe (

    echo File exists

)

if exist C:\myfile.txt (

    echo File exists

) else (

    echo File does not exist

)

Output:

File exists

File does not exist

### 'if errorlevel'

**Usage**: Test the exit codes of the last command that was run. Various commands issue integer exit codes to denote the status of the command. Generally, commands pass 0 if the command was completed successfully and 1 if the command failed.

**Example**:

# Arguments

The arguments can be called from the batch files through the variables %1, %2, %3, and so on.

For example:

@echo off

echo first arg %1

echo second arg %2

echo third arg %3

Output:

C:\Users\Admin\Desktop> test.bat 4 5 6

first arg 4

second arg 5

third arg 6

**Note**: If we were to run the batch as test.bat 4 5 6 7. The output will still remain the same as above because the fourth argument will be ignored.

# Operators

<https://www.tutorialspoint.com/batch_script/batch_script_operators.htm>

# Loops

For usage guide, run help for.

## 'for' (default)

**Usage**: Iterate over a list of files.

**Syntax:**

for %%variable in (set\_of\_files) do (

...

)

Where:

* set\_of\_files: Set of files which are separated by standard delimiters (can be a comma, a space or a semicolon). These files need to be in the same disk drive.
* variable: Name of variable (must have **one character**).

**Example**:

@echo off

for %%f in (E:\test\file1.data E:\test\file2.txt) do (

   echo Copying %%f

   copy %%f E:\backup

)

pause

Output:

Copying E:\test\file1.data

1 file(s) copied.

Copying E:\test\file2.txt

1 file(s) copied.

Press any key to continue ...

## 'for' /r

**Usage**: Iterate over a list of files, including files in subdirectories. It is called *recurse* loop.

**Syntax:**

for /r [path] %%variable in (set\_of\_file\_filters) do (

...

)

Where:

* path: The root folder. If it’s not described, the folder containing executable script file or current folder will be considered as root folder.
* set\_of\_file\_filters: List of file filters. For example \*.txt, \*.bat, or dot (.) means all.
* variable: Name of variable and must have one unique character.

**Example**: Print the list of all \*.txt or \*.log files in C:/Windows/System32 (including subdirectories).

@echo off

for /r "C:\Windows\System32" %%f in (\*.txt \*.log) do (

   echo %%f

)

pause

Output:

C:\Windows\System32\WindowsCodecsRaw.txt

C:\Windows\System32\NetSetupMig.log

C:\Windows\System32\catroot2\dberr.txt

C:\Windows\System32\drivers\gmreadme.txt

C:\Windows\System32\DriverStore\FileRepository\igdlh64.inf\_amd64\_0d5be4a8324f3703\ocl\_cpu\_llvm\_release\_license.txt

C:\Windows\System32\DriverStore\FileRepository\igdlh64.inf\_amd64\_0d5be4a8324f3703\ocl\_cpu\_readme.txt

C:\Windows\System32\LogFiles\setupcln\setupact.log

C:\Windows\System32\LogFiles\setupcln\setuperr.log

C:\Windows\System32\Macromed\Flash\FlashInstall64.log

C:\Windows\System32\MailContactsCalendarSync\LiveDomainList.txt

C:\Windows\System32\wbem\WMIMigration.log

C:\Windows\System32\WindowsPowerShell\v1.0\en-US\default.help.txt

Press any key to continue . . .

## 'for' /d

**Usage**: Iterate over a list of directories which are subdirectories of current directory.

**Syntax:**

for /d [/r] %%variable in (set\_of\_directory\_filters) do (

...

)

Where:

* set\_of\_directory\_filters: List of directory filters. For example, en\*, fr\*, etc.
* [/r]: Not mandatory. If it is available, subdirectories will be involved in the loop.
* variable: Name of variable and must have a unique character.

**Example**: List all subdirectories of C:/Windows with the name started by "media" or "Microsoft.NET":

@echo off

cd C:/Windows

for /d /r %%d in (media\* Microsoft.NET\*) do (

    echo %%d

)

pause

Output:

C:\Windows\Media

C:\Windows\Microsoft.NET

C:\Windows\SystemApps\Microsoft.Windows.CloudExperienceHost\_cw5n1h2txyewy\media

C:\Windows\SystemApps\Microsoft.Windows.CloudExperienceHost\_cw5n1h2txyewy\webapps\inclusiveOobe\media

C:\Windows\SystemApps\Microsoft.Windows.CloudExperienceHost\_cw5n1h2txyewy\webapps\scoobe\media

Press any key to continue . . .

## 'for' /l

**Usage**: Iterate over a range of numbers.

**Syntax:**

for /l %%variable in (start, step, end) do (

...

)

Where:

* start: The first value of variable
* step: After each iteration, variable’s value will be added a 'step'.
* end: The last value of variable.

**Example**:

1. Increasing order

@echo off

for /l %%d in (1 2 8) do (

   echo %%d

)

pause

Output:

1

3

5

7

Press any key to continue . . .

2. Decreasing order

@echo off

for /l %%d in (8 -2 1) do (

   echo %%d

)

pause

Output:

8

6

4

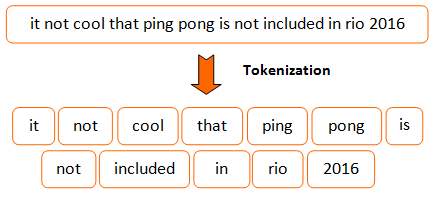
2

Press any key to continue . . .

## 'for' /f

**Usage**:

* It’s a complex but powerful loop. It reads a file or a few files and, then analyzes their content. The content of a file is a text; it is split into several small pieces of text, each of which is called a **token**. The default rule for separating a text is based on whitespace. However, you can customize the delimiter rule by ["delims = xxx"] parameter.
* It's also used to analyze the contents of a string, or to execute a set of commands.



**Syntax:**

for /f ["options"] %%variable in (set\_of\_filenames) do (

...

)

for /f ["options"] %%variable in ("Text string to process") do (

...

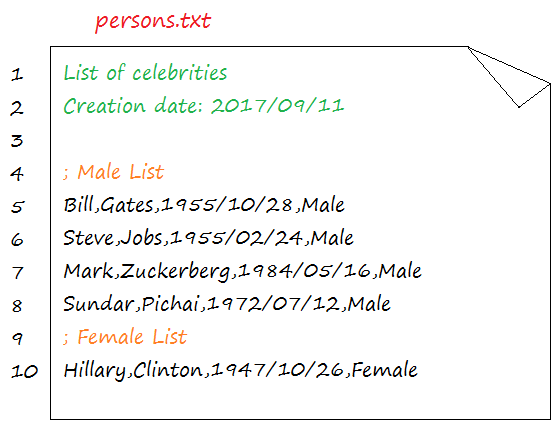
)

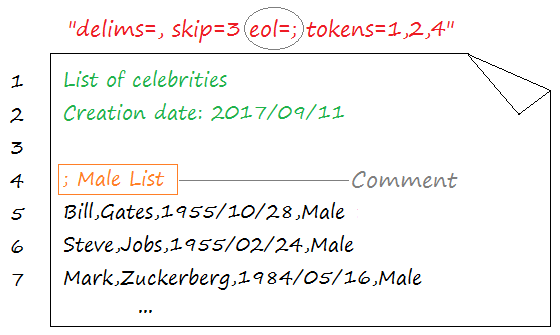
Where:

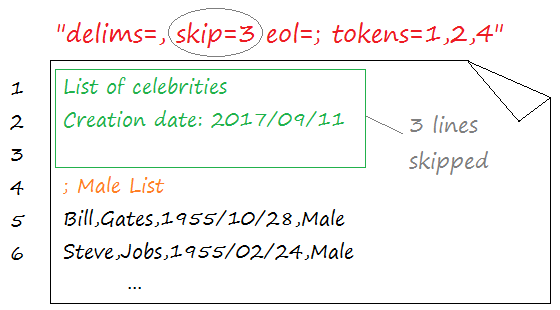
* set\_of\_filenames: List of files.

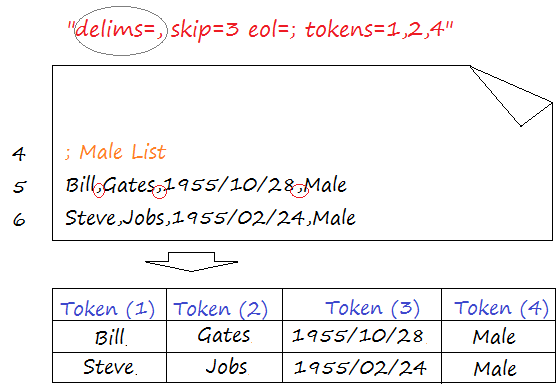
|  |  |
| --- | --- |
| Option | Description |
| delims=xxx | Delimiter character(s). By default, it's a space. |
| skip=n | The first line number will be ignored in file. By default, skip = 0​​​​​​​. |
| eol=; | End of line. Specifies a special character, which is put at the beginning of a line to mark this line as comment line. Comment lines will be ignored by the program. By default, it's a semicolon character (;). |
| tokens=n1,n2,n3 | Defines the positions cared about (n1, n2, n3, ..). By default, tokens=1.  tokens=1,2,4 The indices such as 1, 2, 4 are interested.  tokens=2-8 The indices of 2 to 8 are interested  tokens=3,\* The indices such as 3, 4,5, ... are interested  tokens=\* All indices are interested |
| usebackq |  |

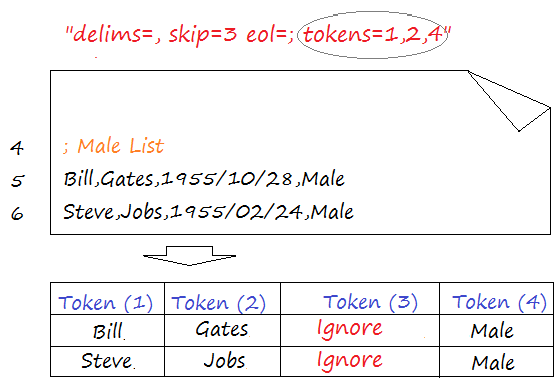
To be easy to understand, let's analyze the following file:











Example of using the for /f loop to read *persons.txt*file:

persons.txt:

List of celebrities

Creation date: 2017/09/11

; Male List

Bill,Gates,1955/10/28,Male

Steve,Jobs,1955/02/24,Male

Mark,Zuckerberg,1984/05/16,Male

Sundar,Pichai,1972/07/12,Male

; Female List

Hillary,Clinton,1947/10/26,Female

Code:

@echo off

for /f "delims=, skip=3 eol=; tokens=1,2,4" %%i in ( person.txt ) do (

   echo Full Name: %%i %%j   Gender: %%k

)

pause

Output:

Full Name: Bill Gates Gender: Male

Full Name: Steve Jobs Gender: Male

Full Name: Mark Zuckerberg Gender: Male

Full Name: Sundar Pichai Gender: Male

Full Name: Hillary Clinton Gender: Female

**Note**: **%%i** variable is explicitly declared on the loop. **%%j** and **%%k** variables are implicitly declared (names of implicit variables are the next characters of explicit variable names).

## Substitution of 'for' variable references

You can use the following optional syntax:

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

# Aliases

<https://stackoverflow.com/a/39459404>

# Redirection Operators

### Output Redirection (>)

**Usage**: The > operator is used to send standard **output** (stdout) to a file. To append content to an existing file instead of overriding it, use the >> operator.

**Examples**:

* The output of command diris re-directed to the file list.txt instead of the screen:

> dir > list.txt

> more list.txt

06/11/2020 11:03 PM <DIR> .

06/11/2020 11:03 PM <DIR> ..

05/31/2020 04:40 PM <DIR> folder1

05/31/2020 06:46 PM <DIR> folder2

06/11/2020 12:08 AM 174 test.cmd

### Input Redirection (<)

**Usage**: The < symbol is used to read standard input from a file.

**Examples**:

* To find the folder named "folder1" in the above list.txt, use:

> find "folder1" < list.txt

05/31/2020 04:40 PM <DIR> folder1

### Error Output Redirection (2>)

**Usage**: The 2> symbol is used to send standard **error** (stderr) to a file. To append content to an existing file instead of overriding it, use the 2>> operator.

**Examples**:

* With the command myprogram 2> errorsfile.txt, we are executing a program named myprogram while re-directing its error output to a file named errorfile.txt. Thus, program output is not cluttered with errors.

**Tip**: You can even combine the stdout and stderr streams using the file number and the '&' prefix. For example: dir C:\ > lists.txt 2>&1

### Pipe Redirection (|)

**Usage**: The pipe (|) is used to redirect a stream from one program to another.

When a program's standard output is sent to another through a pipe, the first program's data, which is received by the second program, will not be displayed on the terminal. Only the filtered data returned by the second program will be displayed.

**Example**:

* Display only lines containing the word "Path" in the output of the command set:

> set | find "Path"

Path=C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Git\cmd

PSModulePath=C:\Program Files\WindowsPowerShell\Modules

# Errorlevel

Almost all applications and utilities will set an Exit Code when they complete/terminate.

The exit codes that are set do vary, in general a code of 0 (false) will indicate successful completion.

By default, SCCM will only consider 0 a success, but commands like Robocopy may return success Exit Codes from 0 to 7.

## Detecting Errorlevels

There are two different methods of checking an errorlevel:

**1.** Via IF ERRORLEVEL ...

This syntax provides compatibility with old .bat batch files from the era of MS-DOS

IF ERRORLEVEL n statements should be read as IF Errorlevel >= number

Examples:

IF ERRORLEVEL 0 will return TRUE whether the errorlevel is 0, 1 or 5 or 64

IF ERRORLEVEL 1 will return TRUE whether the errorlevel is 1 or 5 or 64

IF NOT ERRORLEVEL 3 means if ERRORLEVEL is less than 3 ( 2, 1, 0 or a negative number).

To check for a specific error level N, you can use the following construct:

IF ERRORLEVEL N IF NOT ERRORLEVEL N+1 COMMAND

This is not very readable or user friendly and does not account for negative error numbers.

**2.** Via %ERRORLEVEL% variable

This is a preferred method of checking Errorlevels.

Examples:

IF %ERRORLEVEL% NEQ 0 ECHO An error was found

IF %ERRORLEVEL% EQU 0 ECHO No error found

## Old Style .bat files vs .cmd files

There is a key difference between the way .CMD and .BAT batch files set errorlevels:

* A .BAT batch script running the 'new' internal commands: APPEND, ASSOC, PATH, PROMPT, FTYPE and SET will only set ERRORLEVEL if an error occurs. So, if you have two commands in the batch script and the first fails, the ERRORLEVEL will remain set even after the second command succeeds.
* A .CMD batch script is more consistent and will set ERRORLEVEL after every command that you run.

So as a thumb of rule, **always use errorlevels with .CMD format**.

**Notes:**

Even though a CMD batch script should set or reset ERRORLEVEL after every command, there are a few exceptions:

* Commands that do NOT affect the ERRORLEVEL: BREAK, ECHO, ENDLOCAL, FOR, IF, PAUSE, REM, RD/RMDIR, TITLE
* Commands that will set but not clear an ERRORLEVEL: CLS, GOTO, KEYS, POPD, SHIFT
* Commands that set an Exit Code but not the ERRORLEVEL: RD/RMDIR
* Commands that set an ERRORLEVEL but not the Exit Code (SO explanation): MD/MKDIR