FREE eBook

LEARNING batch-file

Free unaffiliated eBook created from
Stack Overflow contributors.

#batch-file

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: batch-file

It is an unofficial and free batch-file ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official batch-file.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with batch-file

Remarks

From Microsoft Technet:

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

Batch File Names and Extensions

Extension	Remarks
.bat	This extension runs with MS-DOS and all versions of Windows
.cmd	Used for batch files in Windows NT family
.btm	The extension used by 4DOS and 4NT

To understand the difference between `.cmd` and `.bat` please see here.

Avoid names which are already the name of built-in commands. like `tracert`. There is a utility called `tracert.exe`. So, avoid naming a batch file `tracert.bat`

Running Batch File

The easiest way to run a batch file is simply double-clicking its icon. Or paste the file full path into a command prompt, or just its name if command Prompt was started from the batch file directory, then enter.

Example:

```
C:\Foo\Bar>test.bat
C:\Foo\Bar>C:\Foo\Bar\Baz\test.bat
```

Examples

Opening a Command Prompt

The command prompt comes pre-installed on all Windows NT, Windows CE, OS/2 and eComStation operating systems, and exists as `cmd.exe`, typically located in `C:\Windows\system32\cmd.exe`

On Windows 7 the fastest ways to open the command prompt are:

- Press `Win ⊞`, type "cmd" and then press `Enter`.
- Press `Win ⊞`+`R`, type "cmd" then then press `Enter`.
- If you have an explorer window open, type "cmd" in the address bar to open a prompt in the currently selected directory.
- Right-click a folder in Explorer while holding `Shift` and select "Open command window here".

It can also be opened by navigating to the executable and double-clicking on it.

In some cases you might need to run `cmd` with elevated permissions, in this case right click and select "Run as administrator". This can also be achieved by pressing `Control`+`Shift`+`Enter` instead of `Enter` when using way 1 of the points above.

Editing and Viewing Batch Files

Any ASCII editor can edit batch files. A list of editors that can syntax highlight batch syntax can be found here. You can also use the default notepad shipped with windows to edit and view a batch file, although it does not offer syntax highlighting.

To open notepad:

- Press `Win ⊞`+`R`, type `notepad` and then press `Enter`.

Alternatively, the most "primitive" way to create a batch file is to redirect output from the command line to a file, eg.

```
echo echo hello world > first.bat
```

which writes `echo hello world` to the file `first.bat`.

You can edit a batch file by right clicking the file and selecting "Edit" from the context menu.

To view the contents of a batch file from within a command prompt, run the following command:

```
type first.bat
```

You can also start editing your batch file with notepad from the command prompt by typing

```
notepad first.bat
```

Getting Help

To get help on a batch file command you can use the built-in help.

Open a command prompt (whose executable is `cmd.exe`) and enter `help` to see all available

commands.

To get help for any of these commands, type `help` followed by the name of the command.

For example:

```
help help
```

Will display:

```
Provides help information for Windows commands.

HELP [command]

    command - displays help information on that command.
```

Some commands will also display help if followed by `/?`.

Try:

```
help /?
```

Note:

`Help` will only display the help for **internal** commands.

Read Getting started with batch-file online: https://riptutorial.com/batch-file/topic/1277/getting-started-with-batch-file

Chapter 2: Add delay to Batch file

Introduction

This topic will teach you one of the many useful things to know in the scripting language, batch file; Adding a delay/pause/timeout to your batch file.

Examples

Timeout

Timeout

The simplest way to make a delay or pause for a certain amount of time, is with the standard command `TIMEOUT`. To make a timeout that lasts exactly one minute we type:

```
timeout /t 60
```

Now what is going on here?

First off we use the command `TIMEOUT` with the parameter `/T` (which simply means timeout) then we specify the amount of seconds to wait. In this case... `60` seconds.

Timeout with the parameter /NOBREAK

If we take the example from before and run that in a BATCH file: `timeout /t 60` then while waiting those 60 seconds, you are actually able to break the timeout by pressing any key on your keyboard. To prevent this we simply add the parameter `/NOBREAK` to the end of it.

```
timeout /t 60 /nobreak
```

By doing this it will timeout for 60 seconds, and if you want to break the timeout you will have to press (CTRL-C) on your keyboard.

Silent timeout

When it's doing a timeout it will display:

```
Waiting for X seconds, press a key to continue ...
or
Waiting for X seconds, press CTRL+C to quit ... [This is with the /NOBREAK parameter]
```

To hide the message use the `NUL` argument (For explanation of `NUL`: **Click Here**)

```
timeout /t 60 > nul
or
timeout /t 60 /nobreak > nul
```

Pause

To make your script pause simply use the `PAUSE` command.

```
PAUSE
```

This will display the text `Press any key to continue . . .`, then add a newline on user input.

Let's say we want to create a "Hello World" program and after we click something on our keyboard, we want it to exit the program with the `EXIT` command.

```
echo Hello World
pause
exit
```

Here it uses the `ECHO` command to say "Hello World". Then we use the `PAUSE` command which displays `Press any key to continue . . .` and then we use the `EXIT` command to terminate the current BATCH script.

When it's pausing it will display:

```
Press any key to continue . . .
```

Hide the "Press any key to continue... prompt

To hide the message we redirect the output to a special device called `nul`. This isn't actually a real device, but whatever we send to it is thrown away.

```
pause > nul
```

Ping

Ping

One of the most used command to delay for a certain amount of time is `ping`.

Basic usage

```
PING -n 1 -w 1000 1.1.1.1

REM the -n 1 flag means to send 1 ping request.
REM the -w 1000 means when the IP(1.1.1.1) does not respond, go to the next command
REM 1.1.1.1 is an non-existing IP so the -w flag can ping a delay and go to next command
```

This would output the following on your batch file/console:

```
C:\Foo\Bar\Baz>ping -n -w 1000 1.1.1.1

Pinging 1.1.1.1 (Using 32 bytes of data)
Request timed out

Ping statistics for 1.1.1.1
    Packets: Sent = 2,Received = 0, Lost = 1(100% loss)
```

Hide the text echoed out

Just add `>nul` at the back of the command to redirect it to null.

```
ping -n w 1000 1.1.1.1 >nul
```

This would output nothing.

Sleep

Sleep

On older Windows system, `timeout` is not available. However, we can use the `sleep` command.

Usage

```
sleep 1
```

Very self-explanatory; sleep for 1 second. However, `sleep` is a deperacted command and should be replaced by timeout.

Availability

This command is available on old Windows system. Also `SLEEP.exe` is included in 2003 Resource Kit.

To use `sleep.exe`, put the executable file to `%Windir%\System32` folder. Then you can use it as normal command.

Read Add delay to Batch file online: https://riptutorial.com/batch-file/topic/9123/add-delay-to-batch-file

Chapter 3: Batch and JScript hybrids

Introduction

`JScript` is actually the superset of `Javascript` (it's 1.8.1 version - so some newer features are not available), and they can be embedded into a `batch` script for extending `batch` script's functions. Usually, techniques of embedding are using the JScript directives (not part of the official Javascript standard) in order to separate the batch and JScript code. JScript allows you to work with Com/ActiveX objects, as well as with WMI objects in addition to the standard Javascript.

Examples

Embedded JScript In a Batch File

This following example is created by user Michael Dillon from this answer.

Consider the following script:

```
@set @junk=1 /*
@echo off
cscript //nologo //E:jscript %0 %*
goto :eof
*/

//JScript aka Javascript here
```

This script snippet does:

- Execute the `cscript` command which calls itself with all the arguments provided.
- As the part after `@set @junk=1` is commented(`/*` and `*/` Are valid JScript comment),
- JScript will ignore them.
- **Note: We need the `@set @junk=1` part because the batch file does not recognize `/*` as a command, but a `set` statement will be a workaround. JScript will recognize `/*` as a comment so the other `batch` file will not be executed by JScript engine.**

You can add your JScript after `*/` and start extending your batch file scripting!

Run JScript With Temporary Files

As mentioned here, the old-school method to run another script is by using temporary files. Simple `echo` it into a file and then run it(and remove it optionally).

Here's the basic concept:

```
@echo off
echo //A JS Comment > TempJS.js
echo //Add your code>>TempJS.js

cscript //nologo //e:cscript.exe TempJS.js

del /f /s /q TempJS.js
```

But this would require lots of `echo` statements to create a relatively large JScript. Here's a better method by Aacini.

```
@echo off
setlocal

rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > TempJS.js

cscript //nologo //e:cscript.txt TempJS.js
del /f /s /q TempJS.js

goto :EOF

<resource>
JScript
JScript
JScript
```

Read Batch and JSCript hybrids online: https://riptutorial.com/batch-file/topic/10578/batch-and-jscript-hybrids

Chapter 4: Batch and VBS hybrids

Introduction

`Batch` are capable of running with `VBS` functionality further increasing their reliability. For example, `VBS` can deal with decimals, spaces, and some other advanced operations that cannot be done in `batch`. Also is capable of working with WMI and ActiveX objects.

Examples

Run VBS with temporary file(s)

The old-school method for running another script from `batch` is to `echo` the script into another location, and then run it.

This method can be represented like this:

```
@echo off
rem VBS below
    echo your vbs > TempVBS.vbs
    echo other vbs>>TempVBS.vbs
rem End of VBS

cscript //nologo TempVBS.vbs
del /f /s /q TempVBS.vbs
```

The method above would require lots of `echo (vbs) >> TempVBS.vbs`, so here's a way to shorten it. *(code by Aacini)*

```
@echo off
setlocal

rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > Program.vbs

cscript //nologo Program.vbs
del /f /s /q Program.vbs
exit /b

<resource>
your vbs
another line of vbs
```

The last method is by using `streams`. A file can have a few streams. And every stream can contain different information.

```
@echo off
```

```
    echo vbs >%0:stream1
    rem This command redirect "vbs" into the stream1 of this script, then we can call it later

cscript %0:stream1 //nologo
   rem if you like, you can clear the stream1 of this file by:
   type nul>%0:stream1
```

Embed vbscript code into batch file without using temporary files

Here's an example with the technique(hack) invented by the dostips forums' user Liviu:

```
@echo off
echo Printed by CMD.EXE
cscript //nologo "%~f0?.wsf" //job:JS //job:VBS

exit /b %errorlevel%

----END OF BATCH CODE---
<package>
  <job id="JS">
    <script language="VBScript">

        WScript.Echo("Printed by VBScript"):

    </script>
  </job>
  <job id="VBS">
    <script language="JScript">

        WScript.Echo("Printed by JScript");

    </script>
  </job>
</package>
```

As running `wsf` file with windows script host is extension sensitive you can run a file with any extension by adding `?.wsf` at the end of the file (which is the core of the hack). While the Liviu's example is probably more robust the above code is more simplified version. As wsh does not care much about the things outside the `<package>` node you are not obligated to put everything in xml comments. Though it's to be careful with redirection symbols (`<` and `>`)

Read Batch and VBS hybrids online: https://riptutorial.com/batch-file/topic/10061/batch-and-vbs-hybrids

Chapter 5: Batch file command line arguments

Examples

Command line arguments supplied to batch files

Batch file command line arguments are parameter values submitted when starting the batch. They should be enclosed in quotes if they contain spaces. In a running batch file, the arguments are used for various purposes, i.e. redirection to `:labels`, setting variables, or running commands.

The arguments are referred to in the batch file using `%1, %2, ..., %9`.

```
@echo off
setlocal EnableDelayedExpansion
if not "%1"=="" (
    set "dir=%~1" & set "file=%~2"
    type !dir!\!file! | find /n /i "True" >nul^
     && echo Success! || echo Failure
)
exit /b

C:\Users\UserName> test.bat "C:\Temp\Test Results" "Latest.log"
Success!
```

Notes:

- In the above example, double quotes are removed by using the argument modifier `%~1`.
- Long strings are split to several lines using `^`, and there is a space before the character on the next line.

Batch files with more than 9 arguments

When more than 9 arguments are supplied, the `shift [/n]` command can be used, where `/n` means start at the nth argument, n is between zero and eight.

Looping through arguments:

```
:args
set /a "i+=1"
set arg!i!=%~1
call echo arg!i! = %%arg!i!%%
shift
goto :args
```

Note, in the above example delayed expansion variable `i` is used to assign argument values to variables array. The `call` command allows to display such variable values inside the loop.

Counting arguments:

```
for %%i in (%*) do (set /a ArgCount+=1)
echo %ArgCount%
```

Set a variable to n'th argument:

```
set i=5
call set "path%i%=%%~i"
```

Shifting arguments inside brackets

Lets have the following `example.bat` and call it with arguments `1` ,`2` and `3`:

```
@echo off

(
    shift
    shift
    echo %1
)
```

As the variable expansion will change after the the end brackets context is reached the output will be:

    1

As this might be an issue when shifting inside brackets to access the argument you'll need to use call:

```
@echo off

(
    shift
    shift
    call echo %%1
)
```

now the output will be `3`. As `CALL` command is used (which will lead to additional variable expansion) with this technique the arguments accessing can be also parametrized:

```
@echo off

set argument=1

    shift
    shift
    call echo %%%argument%
```

with delayed expansion:

```
@echo off
setlocal enableDelayedExpansion
set argument=1

    shift
    shift
    call echo %%!argument!
```

the output will be

    3

Read Batch file command line arguments online: https://riptutorial.com/batch-file/topic/4981/batch-file-command-line-arguments

Chapter 6: Batch file macros

Introduction

In a command prompt, you can use DOSKEY for creating macros. In a batch file you can define a variable that can be called as a piece of code and even pass arguments to it.

Examples

Basic Macro

Using `DOSKEY`, we can create macros to simplify typing many commands in command prompt. Take a look at the following example.

```
DOSKEY macro=echo Hello World
```

Now if you type `macro` in the command prompt, it would return `Hello World`.

Comments

Unfortunately, `DOSKEY` macro doesn't support comment, but there's a workaround.

```
;= Comment
;= Comment
;= Remember to end your comment with ;=
;=
```

$ Character Usages

There are 3 usages of the `$` character in a `DOSKEY` macro.

Command separator

`$T` is the equivalent of `&` in a batch script. One can join commands together like so.

```
DOSKEY test=echo hello $T echo world
```

Command-line arguments

Like `bash`(not `batch`), we use `$` to indicate command-line argument.

`$1` refers to the first command-line argument

`$2` refers to second command-line argument, etc..

`$*` refers to all command-line argument

Macros In Batch Script

`DOSKEY` macros don't work in a batch script. However, we can use a little workaround.

```
set DOSKEYMacro=echo Hello World
%DOSKEYMacro%
```

This script can simulate the macro function. One can also use ampersands(`&`) to join commands, like `$T` in `DOSKEY`.

If you want a relatively large "macro", you may try a simple function or take a look at other function topics here.

Read Batch file macros online: https://riptutorial.com/batch-file/topic/10791/batch-file-macros

Chapter 7: Batch files and Powershell hybrids

Examples

Run Powershell with Temporary Files

This has been mentioned in other hybrid topics again and again. The old-school, but easy method to run Powershell is by:

- `echo`ing the Powershell script into a temporary script
- Execute the temporary script
- Optionally remove the temporary script

This is a sample script.

```
@echo off
echo powershell-command>Temp.ps1
echo another line>>Temp.ps1
    rem echo the script into a temporary file

powershell -File Temp.ps1
    rem execute the temporary script

del Temp.ps1
    rem Optionally remove the temporary script
```

The method above requires tons of `echo` statement if a long script is required, here is a better method suggest by @Aacini

```
@echo off
setlocal

    rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

    rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > Temp.ps1

powershell -File Temp.ps1
del /f /s /q Temp.ps1

goto :EOF

<resource>
PS
Powershell script
```

Use POWERSHELL Command To Execute 1-line Powershell Command

Using the `POWERSHELL` command, we can execute a 1-line command directly from a batch script,

without any temporary file.

Here's the syntax.

```
powershell.exe -Command <yourPowershellCommandHere>
```

You may also want to include other flags, like `-Nologo` to improve the actual outcome.

Powershell/batch hybrid without temp files

This is the approach proposed by the stackoverflow's user rojo which also can handle the command line arguments :

```
<# : batch portion
@echo off & setlocal

(for %%I in ("%~f0";%*) do @echo(%%~I) | ^
powershell -noprofile "$argv = $input | ?{$_}; iex (${%~f0} | out-string)"

goto :EOF
: end batch / begin powershell #>

"Result:"
$argv | %{ "`$argv[{0}]: $_" -f $i++ }
```

called like this:

    psbatch.bat arg1 "This is arg2" arg3

will produce:

```
Result:
$argv[0]: C:\Users\rojo\Desktop\test.bat
$argv[1]: arg1
$argv[2]: This is arg2
$argv[3]: arg3
```

Read Batch files and Powershell hybrids online: https://riptutorial.com/batch-file/topic/10711/batch-files-and-powershell-hybrids

Chapter 8: Best Practices

Introduction

This topic will focus on the things that one should *(not mandatory)* do in a batch file. Using these "best practices" can enhance the effect and the function of a batch file.

Examples

Quotes

Most online batch scripts come with a lot of quote issues.

Examples and Solutions

Example A

```
if %var%==abc echo Test
```

This code works - when the content of `%var%` does not contains space or other special characters. Now let's assume `%var%` contains 1 whitespace. Now `cmd.exe` sees:

```
if  ==abc echo Test
```

This would cause a failure because `cmd.exe` doesn't understand this syntax.

Solution A

```
if "%var%"=="abc" echo Test
```

Using quotes, `cmd.exe` sees the entire `%var%`(including space and special characters) as only one normal string. Yet this is not the safest comparison method. The safest one uses `echo`, `pipe`, and `findstr`.

Example B

```
cd C:\User\Spaced Name\Spaced FileName.txt
```

`cd` would only change directory to `C:\User\Spaced`, as `cd` only accepts one path argument.

Solution B

Simply by adding quotes around the path, the issue would be solved.

```
cd "C:\User\Spaced Name\Spaced FileName.txt"
```

There are also a few examples that work better using quotes, like the `set /a` statement, etc. But, when one works on strings that contain spaces or special characters, it is usually much safe to use quotes.

Spaghetti Code

Spaghetti code means a code snippet that uses many, and often confusing structures. Such as `GOTO`s, exceptions and inconsistent code.

Examples and Solutions

Example A

```
@echo off
set /a counter=0

:Loop
set /a counter=%counter% + 1
echo %counter%

if %counter% equ 10 goto :exit
goto :Loop

:exit
```

This program comes with plenty of jumps, making us harder to know what exactly the script is doing.

Solution A

```
@echo off
for /l %%G in (0,1,10) echo %%G
```

Using less `GOTO`s, we reduced the amount of code greatly, and we can focus on the actual code.

Example B

Consider the following statements.

```
:endGame
if %player1Score% gtr %player2Score% goto :player1wins
if %player1Score% lss %player2Score% goto :player2wins
goto :tie

:player1wins
echo player 1 wins
goto :eof

:player2wins
echo player 2 wins
goto :eof

:tie
echo tie
goto :eof
```

This snippet requires lots of `goto` statements and can be confusing to debug. To simplify these statements, we can use `call` command. Here is the above script at a better condition.

```
:endGame
if %player1Score% gtr %player2Score% call :message player 1 wins
if %player1Score% lss %player2Score% call :message player 2 wins
if %player1Score% equ %player2Score% call :message tie

goto :eof

:message
echo %*
goto :eof
```

Both scripts output the exact same result, but the new script is much shorter and clearer.

Read Best Practices online: https://riptutorial.com/batch-file/topic/10746/best-practices

Chapter 9: Bugs in cmd.exe processor

Introduction

This topic will focus on errors caused by the processor bugs. Here are the things we would focus on the cause and the solution of the issue.

Remarks

In the example DEL File Extension, user X. Liu notices that this bug will **not** occurs when the file extension in the `DEL` command is less than 3 characters.

Examples

Parentheses Confusion

From this website, the OP has noticed a problem.

Cause

Consider the following code snippet.

```
if 1==1 (
    set /a result = 2*(3+4)
)
```

At your first glance, you may think `CMD.exe` would process it like so:

- The condition is true, execute code block
- Set variable result's value to `14`
- Continue

However, `CMD.exe` process like so:

- The condition is true, execute code block
- **Calculate `2*(3+4`, the `)` after 4 is processed at the end of `if` code block**
- A random `)` has appeared!

The second step would return `Unbalanced parentheses` error.

Solution

According to a German CMD.exe's `set /?`, we would need to quote arithmetic operations. Here's an example.

Previous	Result
`set /a result=2+5*4`	`set /a result="2+5*4"`

By the way, according to an English CMD.exe `set /?`, quotes are required if logical or modulus operators are present in the expression(although this is not a *must-do* step).

Improper Escape Character

In this Stack Overflow question, user txtechhelp found an issue with the `^` character which could cause a security issue.

Cause

```
anyInvaildCommand ^
```

Note: Make sure the caret(`^`) is the last character! Any extra `CR\LF` won't work at all!

The caret looks for the next character to escape. However, the are no more character available to escape, so `cmd` loops infinitely, looking for a character to escape. In this "loop" process, `cmd.exe` will consume your computer memory. And gradually eating all memory, bringing the computer to knees.

This issue can lead to more serious security worries as one could just enter the code into the one's unlocked computer.

Solutions

- Use codepage **UTF-16** could solve this problem. Only **UTF-8** or **ASCII** would cause the bug.
- Make sure there is an extra `CR\LF` in the file, or just simply don't use caret at the end of the file.

Extra

This bug seems to be solved in Windows 10.

DEL File Extension

This bug was reported by steve2916 from this Microsoft Windows Forum thread.

Cause

Consider a folder with such files.

```
TestA.doc
TestB.doc
TestC.docx
TestD.docx
```

If we want to remove all `.doc` file in this directory, we usually would do:

```
del *.doc
```

However, this command also removes the `.docx` files. The same happens on file extensions with this pattern.

File A	File B
Anyname.**abc**	AnotherName.**abc**d

As we can see, as long as the file extension string contains the string used in the `del` command, the file will be deleted. Note that this bug only occurs when the extension string used in the `del` command has at least three characters. For instance, `del *.do` doesn't delete `A.doc` or `A.docx`.

Solution

In the thread, user MicroCompsUnltd noticed that using `Powershell` would solve the issue.

Read Bugs in cmd.exe processor online: https://riptutorial.com/batch-file/topic/10694/bugs-in-cmd-exe-processor

Chapter 10: Bypass arithmetic limitations in batch files

Introduction

Batch files allows only 32bit integer calculations , though this can be bypassed with different approaches.

Examples

Using powershell

As the powershell is installed by default on every windows system from 7/2008 and above it can be used for more complex calculations:

```
@echo off
set "expression=(2+3)*10/1000"
for /f %%# in ('"powershell %expression%"') do set result=%%#
echo %result%
```

Mind the additional double quotes in the `for /f` which prevent brackets conflicts with the for command syntax.

Potential issue is that powershell is much slower than using wsh/vbscript/jscript due to the loading of the .net framerwork

Using jscript

`WSH/JScript` is installed on every windows system since NT so using it for more complex calculations makes it pretty portable. JScript is easier for combining it with batch file :

```
@if (@codesection==@batch) @then
@echo off

set "expression=2*(2+3)/1000"
for /f %%# in ('cscript //nologo //e:jscript "%~f0" "%expression%"') do set
result=%%#
echo %result%
:: more batch code

exit /b %errorlevel%
@end
WScript.Echo(eval(WScript.Arguments(0)));
```

With this approach you can put your whole code in a single file.It is faster than using powershell. Here and here more advanced scripts can be found (which can be used as external files).

Emulating pen and paper calculations, math functions implementations

1. ***Here*** can be found the most comprehensive math library that emulates pen and paper calculations and allows working with bigger numbers.
2. Here are another examples of pen and paper emulations: ADD , Comparison , Multiply
3. Some math functions implementations can be found here.

Read Bypass arithmetic limitations in batch files online: https://riptutorial.com/batch-file/topic/10702/bypass-arithmetic-limitations-in-batch-files

Chapter 11: Changing Directories and Listing their Contents

Syntax

- echo %cd% - displays the current path of the directory
- cd "C:\path\to\some\directory" -changes the path of the directory
- cd "%variable_containing_directory_path%" - also changes the path of the directory
- cd /d E: - change to E: drive from a different drive
- cd/ - changes directory back to current drive
- echo `%__CD__%` - displays the current path of the directory with trailing backslash (undocumented)
- echo %=C:% - The current directory of the C: drive (undocumented)
- echo %=D:% - The current directory of the D: drive if drive D: has been accessed in the current CMD session (undocumented)

Remarks

Why is it important and what are they uses and advantages:

- to open file or application in a directory using batch
- to create andwrite and read files in a directory using batch
- to know and list out all folders
- to know where your batch file is running

Examples

To display the current directory

Format and Usage:

```
echo %cd%
```

`%cd%` is a system variable that contains the current directory path

To change the current directory (without changing drives)

Format:

```
cd "<path>"
```

Example:

```
 cd "C:\Program Files (x86)\Microsoft Office"
```

`cd` is an abbreviation for `chdir` and the two commands behave in the exact same way. For the sake of consistency, `cd` will be used throughout this topic.

To navigate to the directory one level above the current directory, specify the system directory `..`

```
cd ..
```

To navigate to a directory that is inside of the current directory, simply `cd` to the folder name without typing the full path (wrapping the directory name in quotes if it contains spaces).

For example, to enter `C:\Program Files (x86)\Microsoft Office` while in the `C:\Program Files (x86)` directory, the following syntax may be used:

```
cd "Microsoft Office"
```

or

```
cd "C:\Program Files (x86)\Microsoft Office"
```

Navigating to a directory on a different drive

`cd` by itself will not allow a user to move between drives. To move to a different drive, the `/d` option must be specified.

e.g. Moving from `C:\Users\jdoe\Desktop` to `D:\Office Work`

```
cd /d "D:\Office Work"
```

How to show all folders and in files in a directory

Usage to list all folders and files in the current directory: `dir`

A target directory can also be specified: `dir C:\TargetPath`

When specifying a path with spaces, it must be surrounded by quotes: `dir "C:\Path With Spaces"`

Changing drive without CD /D

```
Pushd "D:\Foo"
Dir
```

```
Popd
```

`Pushd` will change the directory to the directory following (in this case D:\Foo. `Popd` returns back to the original directory.

To change the current directory to the root of the current drive

Format:

```
cd/
```

`cd/` is set to change the current directory back to the root of the current drive

Read Changing Directories and Listing their Contents online: https://riptutorial.com/batch-file/topic/7132/changing-directories-and-listing-their-contents

Chapter 12: Comments in Batch Files

Introduction

Comments are used to show information in a batch script.

Syntax

- REM
- &REM
- ::
- &::
- Goto :Label

```
    Comments. You can also use |>< ,etc.
```

:Label

Examples

Using REM for Comments

```
REM This is a comment
```

- `REM` is the official comment command.

Using Labels as Comments

```
::This is a label that acts as a comment
```

The double-colon `::` comment shown above is not documented as being a comment command, but it is a special case of a label that acts as a comment.

Caution: when labels are used as comments within a bracketed code block or `for` command, the command processor expects every label to be followed by at least one command, so when a jump is made to the label it will have something to execute.

The `cmd` shell will try to execute the second line even if it is formatted as a label (and **this causes an error**):

```
(
```

```
echo This example will fail
:: some comment
)
```

When working within bracketed code blocks it is definitely safer to use REM for all comment lines.

Using Variables as Comments

It is also possible to use variables as comments. This can be useful to conditionally prevent commands being executed:

```
@echo off
setlocal
if /i "%~1"=="update" (set _skip=) Else (set _skip=REM)
%_skip% copy update.dat
%_skip% echo Update applied
...
```

When using the above code snippet in a batch file the lines beginning with `%_skip%` are only executed if the batch file is called with `update` as a parameter.

Block Comments

The batch file format does not have a block comment syntax, but there is an easy workaround for this.

Normally each line of a batch file is read and then executed by the parser, but a `goto` statement can be used to jump past a block of plain text (which can be used as a block comment):

```
@echo off
goto :start

A multi-line comment block can go here.
It can also include special characters such as | >

:start
```

Since the parser never sees the lines between the `goto :start` statement and `:start` label it can contain arbitrary text (including control characters without the need to escape them) and the parser will not throw an error.

Comment on the code's line

To comment on the same line as the code you can use `&::` or `&rem`. You can also use `&&` or `||` to replace `&`.

Example :

```
@echo off
echo This is a test &::This is a comment
echo This is another test &rem This is another comment
```

```
pause
```

A curiosity: `SET` command allows *limited* inline comments without `&rem`:

```
set "varname=varvalue"    limited inline comment here
```

Limitations:

- syntax with double quotes `set "varname=varvalue"` or `set "varname="`,
- an inline comment **may not** contain any double quote,
- any `cmd` poisonous characters `| < > &` **must be** properly escaped as `^| ^< ^> ^&`,
- parentheses `( )` **must be** properly escaped as `^( ^)` within a bracketed code block.

Batch and WSF Hybrid Comment

```
<!-- : Comment
```

This works with both batch script and WSF. The closing tag(`-->`), only works in WSF.

Code	Sucessful in both batch and WSF?
`<!--: Comment`	True
`<!--: Comment -->`	False - The closing tag only works for WSF
`-->`	False

Read Comments in Batch Files online: https://riptutorial.com/batch-file/topic/3152/comments-in-batch-files

Chapter 13: Creating Files using Batch

Introduction

One useful feature of batch files is being able to create files with them. This section shows how to create files using batch code.

Syntax

- echo (type here whatever you want in the to be) >> (filename)
- echo (variable name) >> (filename)

Remarks

If a file exists, `>` will overwrite the file and `>>` will append to the end of the file. If a file does not exist, both will create a new file.

Also, the `echo` command automatically adds a newline after your string.

So

```
echo 1 > num.txt
echo 1 > num.txt
echo 2 >> num.txt
```

will create the following file:

```
1
2
```

Not this:

```
1 1 2
```

or

```
1 2
```

Furthermore, you cannot just modify a single line in a text file. You have to read the whole file, modify it in your code and then write to the whole file again.

Examples

Redirection

Format:

```
[command] [> | >>] [filename]
```

`>` *saves* the output of [command] into [filename].

`>>` *appends* the output of [command] into [filename].

Examples:

1. `echo Hello World > myfile.txt` saves "Hello World" into myfile.txt
2. `echo your name is %name% >> myfile.txt` appends "your name is xxxx" into myfile.txt
3. `dir C:\ > directory.txt` saves the directory of C:\ to directory.txt

Echo to create files

Ways to create a file with the echo command:

```
ECHO. > example.bat (creates an empty file called "example.bat")

ECHO message > example.bat (creates example.bat containing "message")
ECHO message >> example.bat (adds "message" to a new line in example.bat)
(ECHO message) >> example.bat (same as above, just another way to write it)
```

If you want to create a file via the `ECHO` command, in a specific directory on your computer, you might run into a problem.

```
ECHO Hello how are you? > C:\Program Files\example.bat

(This will NOT make a file in the folder "Program Files", and might show an error message)
```

But then how do we do it? Well it's actually extremely simple... When typing a path or file name that has a space included in it's name, then remember to use "quotes"

```
ECHO Hello how are you? > "C:\Program Files\example.bat"
(This will create "example.bat" in the folder "Program Files")
```

But what if you want to make a file that outputs a new file?

```
ECHO message > file1.bat > example.bat

(example.bat is NOT going to contain "message > file1.bat")
example.bat will just contain "message"... nothing else
```

Then how do we do this? Well for this we use the `^` symbol.

```
ECHO message ^> file1.bat > example.bat
```

```
(example.bat is going to contain "message > file1.bat")
```

Same goes for other stuff in batch

The next step requires you to have some knowledge about variables and statements:

click here to learn about variables | click here to learn about if and else statements

Variables:

```
SET example="text"
ECHO %example% > file.bat
(This will output "text" to file.bat)
```

if we don't want it to output "text" but just plain %example% then write:

```
ECHO ^%example^% > file.bat
(This will output "%example%" to file.bat)
```

IF/ELSE statements:

```
ELSE statements are written with "pipes" ||

IF ^%example^%=="Hello" ECHO True || ECHO False > file.bat

(example.bat is going to contain "if %example%=="Hello" echo True")
[it ignores everything after the ELSE statement]
```

to output the whole line then we do the same as before.

```
IF ^%example^%=="Hello" ECHO True ^|^| ECHO False > file.bat

This will output:
IF %example%=="Hello" ECHO True || ECHO False
```

If the variable is equal to "Hello" then it will say "True", ELSE it will say "False"

Saving the output of many commands

Using many `ECHO` commands to create a batch file:

```
(
  echo echo hi, this is the date today
  echo date /T
  echo echo created on %DATE%
  echo pause >nul
)>hi.bat
```

The command interpreter treats the whole section in parenthesis as a single command, then saves all the output to `hi.bat`.

`hi.bat`

now contains:

```
echo hi, this is the date today
date /T
echo created on [date created]
pause >nul
```

Read Creating Files using Batch online: https://riptutorial.com/batch-file/topic/7129/creating-files-using-batch

Chapter 14: Deprecated batch commands and their replacements

Examples

DEBUG

DEBUG command was used to create binaries/executabled from batch file. The command is still available in 32bit versions of windows , but is able to create binaries only with 16bit instructions with makes it unusable for 64bit machines. Now CERTUTIL is used for that purpose which allows to decode/encode binary/media files from HEX or BASE64 formats.Example with a file that creates icon file:

```
@echo off

del /q /f pointer.jpg >nul 2>nul
certutil -decode "%~f0" pointer.jpg
hh.exe pointer.jpg
exit /b %errorlevel%

-----BEGIN CERTIFICATE-----
/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja3kAAQAEAAAAMgAA/+4ADkFkb2Jl
AGTAAAAAAf/bAIQACAYGBgYGCAYGCAwIBwgMDgoICAoOEA0NDg0NEBEMDg0NDgwR
DxITFBMSDxgYGhoYGCMiIiIjJycnJycnJycnJwEJCAgJCgkLCQkLDgsNCw4RDg4O
DhETDQ0ODQ0TGBEPDw8PERgWFxQUFBcWGhoYGBoaISEgISEnJycnJycnJycn/8AA
EQgACgAKAwEiAAIRAQMRAf/EAFsAAQEBAAAAAAAAAAAAAAAAAAAGBwEBAQAAAAAA
AAAAAAAAAAAAAAEQAAIBAwQDAAAAAAAAAAAAAAAAEDAgARBSExIwQSIhMRAQEBAAAA
AAAAAAAAAAAAAAAARIf/aAAwDAQACEQMRAD8A13PZ5eIX3gO8ktKZfFPksvQ8r4uL
ecJmx1BMD8A13PZ5eIX3gO8ktKZfFPksvQ8r4uL
-----END CERTIFICATE-----
```

Here's the same with hex format:

```
@echo off

(echo 0000    ff d8 ff e0 00 10 4a 46  49 46 00 01 02 00 00 64)   >pointer.hex
(echo 0010    00 64 00 00 ff ec 00 11  44 75 63 6b 79 00 01 00)   >>pointer.hex
(echo 0020    04 00 00 00 32 00 00 ff  ee 00 0e 41 64 6f 62 65)  >>pointer.hex
(echo 0030    00 64 c0 00 00 00 01 ff  db 00 84 00 08 06 06 06)   >>pointer.hex
(echo 0040    06 06 08 06 06 08 0c 08  07 08 0c 0e 0a 08 08 0a)   >>pointer.hex
(echo 0050    0e 10 0d 0d 0e 0d 0d 10  11 0c 0e 0d 0d 0e 0c 11)   >>pointer.hex
(echo 0060    0f 12 13 14 13 12 0f 18  18 1a 1a 18 18 23 22 22)   >>pointer.hex
(echo 0070    22 23 27 27 27 27 27 27  27 27 27 27 01 09 08 08)   >>pointer.hex
(echo 0080    09 0a 09 0b 09 09 0b 0e  0b 0d 0b 0e 11 0e 0e 0e)   >>pointer.hex
(echo 0090    0e 11 13 0d 0d 0e 0d 0d  13 18 11 0f 0f 0f 0f 11)   >>pointer.hex
(echo 00a0    18 16 17 14 14 14 17 16  1a 1a 18 18 1a 1a 21 21)   >>pointer.hex
(echo 00b0    20 21 21 27 27 27 27 27  27 27 27 27 27 ff c0 00)   >>pointer.hex
(echo 00c0    11 08 00 0a 00 0a 03 01  22 00 02 11 01 03 11 01)   >>pointer.hex
(echo 00d0    ff c4 00 5b 00 01 01 01  00 00 00 00 00 00 00 00)   >>pointer.hex
(echo 00e0    00 00 00 00 00 00 06 07  01 01 01 00 00 00 00 00)   >>pointer.hex
(echo 00f0    00 00 00 00 00 00 00 00  00 00 01 10 00 02 01 03)   >>pointer.hex
(echo 0100    04 03 00 00 00 00 00 00  00 00 00 00 01 03 02 00)   >>pointer.hex
```

```
(echo 0110    11 05 21 31 23 04 12 22  13 11 01 01 01 00 00 00)   >>pointer.hex
(echo 0120    00 00 00 00 00 00 00 00  00 00 00 11 21 ff da 00)   >>pointer.hex
(echo 0130    0c 03 01 00 02 11 03 11  00 3f 00 d7 73 d9 e5 e2)   >>pointer.hex
(echo 0140    17 de 03 bc 92 d2 99 7c  53 e4 b2 f4 3c af 8b 8b)   >>pointer.hex
(echo 0150    79 c2 66 c7 50 4c 49 b9  bc 0f a5 15 29 57 20 ff)   >>pointer.hex
(echo 0160    d9                                              )   >>pointer.hex

certutil -decodehex "pointer.hex" pointer.jpg
hh.exe pointer.jpg
exit /b %errorlevel%
```

As you can see hex requires additional temp file and hex format expansions is bigger

APPEND

`APPEND` was command in msdos that allowed to use resource/media files like they are in the same directory.The command is still available in 32bit versions of windows but seems is not working. In some sources (including microsofts') it is pointed that the command is replaced by DPATH ,but it is not entirely true. Despite the DPATH help message points to APPEND command it's syntnax is the same as `PATH`.The directories listed in DPATH can be used with input redirection or type command :

```
@echo off
dpath %windir%

set /p var=<win.ini
echo using dpath with input redirection:
echo %var%
echo.
echo using dpath with type command:
type win.ini
```

BITSADMIN

`BITSADMIN` was used to transfer documents, download website, download files from websites, etc... This command is a deprecated command, and may be removed on next Windows updates. To prevent this issue, use the new Powershell `BIT cmdlet`.

Here is a sample code utilizing `BITSADMIN`.

```
@echo off
Bitsadmin /create /download Stackoverflow.com
rem download the website StackOverflow.com
```

Read Deprecated batch commands and their replacements online: https://riptutorial.com/batch-file/topic/10109/deprecated-batch-commands-and-their-replacements

Chapter 15: Differences between Batch (Windows) and Terminal (Linux)

Introduction

Batch and bash are quite different. Batch flags are indicated with a `/`, while bash flags use a `-`. Capitalization matters in bash, but (almost) not at all in batch. Batch variable names can contain spaces, bash variable names can not. Ultimately, both are ways of manipulating and interacting with the command line. Not surprisingly, there is a reasonably-sized amount of overlap between the capabilities of the two systems.

Remarks

- `bitsadmin` is deprecated in favor of the PowerShell cmdlet BITS but still works as of Windows 10 version 1607
- `certutil` separates pairs of hexadecimal digits with a space, so `md5sum` will return an example value of `d41d8cd98f00b204e9800998ecf8427e`, while `certutil` displays the same value as `d4 1d 8c d9 8f 00 b2 04 e9 80 09 98 ec f8 42 7e`
- To `cd` to another drive (for example, from C: to D:) the `/d` flag must be used
- `del` can not delete folders, use `rm` instead
- `grep` is so much more powerful than `find` and `findstr`, it's almost not fair to compare them; `find` has no regex capabilities and `findstr` has extremely limited regex capabilities (`[a-z]{2}` is not valid syntax, but `[a-z][a-z]` is)
- `for` loops on the Windows command prompt can only use single-character variable names; this is the only time batch variable names are case-sensitive
- `for` loops on the command prompt also use the variable form `%A` instead of `%A%` - `for`loops in batch scripts use the variable form `%%A`
- `md` automatically creates any necessary parent directories, while `mkdir` needs the `-p` flag to do so
- `rem` may not be used as an inline comment character unless it is preceded by a `&`
- `::` may not be used as an inline comment at all, and should also not be used inside of a code block (set of parentheses)

- Note that some Windows command like `ping` still uses `-` as flags

Examples

Batch Commands and Their Bash Equivalents

Batch	Bash	Description
`command /?`	`man command`	Shows the help for *command*
`bitsadmin`	`wget` or `curl`	Downloads a remote file

Batch	Bash	Description
`certutil -hashfile file_name MD5`	`md5sum file_name`	Gets the MD5 checksum of *file_name*
`cd`	`pwd`	Displays the current directory
`cd directory`	`cd directory`	Changes the current directory to the specified one
`cls`	`clear`	Clears the screen
`copy`	`cp`	Copies a file or files from a source path to a target path
`date`	`date`	Displays the date or sets it based on user input
`del`	`rm`	Deletes a file or files
`dir`	`ls`	displays a list of files and directories in the current directory
`echo`	`echo`	Displays text on the screen
`exit`	`return`	Exits a script or subroutine
`exit`	`logout`	Closes the command prompt or terminal
`fc`	`diff`	Compares the contents of two files
`find "string" file_name`	`grep "string" file_name`	Searches *file_name* for *string*
`findstr "string" file_name`	`grep "string" file_name`	Searches *file_name* for *string*
`for /F %A in (fileset*) do something`	`for item in fileset*; do; something; done`	Do something for every file in a set of files
`for /F %A in ('command') do something`	`` `command` ``	Returns the output of a command
`for /L %A in (first,increment,last) do something`	`for item in `seq first increment last`; do; something; done`	Starts at *first* and counts by *increment* until it reaches *last*
`forfiles`	`find`	Searches for files that match

Batch	Bash	Description
		a certain criteria
`if "%variable%"=="value" (`	`if [ "variable"="value" ]; then`	Compares two values
`ipconfig`	`ifconfig`	Displays IP information
`md`	`mkdir`	Creates new folders
`mklink`	`ln -s`	Creates a symbolic link
`more`	`more`	Displays one screen of output at a time
`move`	`mv`	Moves a file or files from a source path to a target path
`pause`	`read -p "Press any key to continue"`	Pauses script execution until the user presses a button
`popd`	`popd`	Removes the top entry from the directory stack and goes to the new top directory
`pushd`	`pushd`	Adds the current directory to the directory stack and goes to the new top directory
`ren`	`mv`	Renames files
`rem` or `::`	`#`	Comments a line of code
`rd`	`rmdir`	Removes empty directories
`rd /s`	`rm -rf`	Removes directories regardlesss of whether or not they were empty
`set variable=value`	`variable=value`	Sets the value of *variable* to *value*
`set /a variable=equation`	`variable=$((equation))`	Performs math (batch can only use 32-bit integers)
`set /p variable=promptstring`	`read -p "promptstring" variable`	Gets user input and stores it in *variable*
`shift`	`shift`	Shifts arguments by 1 (or n if provided)

Batch	Bash	Description
`sort`	`sort`	Sorts output alphabetically by line
`tasklist`	`ps`	Shows a list of running processes
`taskkill /PID processid`	`kill processid`	Kills the process with PID *processid*
`time /t`	`date`	Displays the current time
`type`	`cat`	Displays the contents of a file
`where`	`which`	Searches the current directory and the PATH for a file or command
`whoami`	`id`	Displays the name and group of the current user

Batch Variables and Their Bash Equivalent

Batch	Bash	Description
`%variable%`	`$variable`	A regular variable
`!variable!`	`$variable`	A variable inside of a code block when `setlocal enabledelayedexpansion` is on
`%errorlevel%` or `ERRORLEVEL`	`$?`	Returns the status of the previous command: 0 if successful, 1 (or something else) if not
`%1`, `%2`, `%3`, etc.	`$1`, `$2`, `$3`, etc.	The parameters given to a script
`%*`	`$*`	All parameters given to a script

Read Differences between Batch (Windows) and Terminal (Linux) online: https://riptutorial.com/batch-file/topic/8362/differences-between-batch--windows--and-terminal--linux-

Chapter 16: Directory Stack

Syntax

- PUSHD [path]
- POPD

Parameters

Parameter	Details
path	The directory to navigate to

Remarks

- Using `pushd` without parameters will print the stack.
- The `popd` command will overwrite the current Current Directory value.

Examples

Delete Text Files

The following example shows how you can use the `pushd` command and the `popd` command in a batch program to change the current directory from the one in which the batch program was run and then change it back:

```
@echo off
rem This batch file deletes all .txt files in a specified directory
pushd %1
del *.txt
popd
cls
echo All text files deleted in the %1 directory
```

Sourced from https://technet.microsoft.com/en-us/library/cc771180%28v=ws.11%29.aspx

Print Directory Stack

To print the directory stack, use the command `pushd` without any parameters:

```
@echo off

cd C:\example\
pushd one
pushd ..\two
```

```
pushd ..\..


pushd
echo Current Directory: %cd%

echo:
popd
pushd three

pushd
echo Current Directory: %cd%
```

Output:

```
C:\example\two
C:\example\one
C:\example
Current Directory: C:\

C:\example\two
C:\example\one
C:\example
Current Directory: C:\example\two\three
```

Read Directory Stack online: https://riptutorial.com/batch-file/topic/4288/directory-stack

Chapter 17: Echo

Introduction

`echo` can be used to control and produce output.

Syntax

- ECHO [ON | OFF]
- ECHO message
- ECHO(message
- ECHO(

Parameters

Parameter	Details
ON | OFF	Can either be `ON` or `OFF` (case insensitive)
message	Any string (except `ON` or `OFF` when used without `(`)

Remarks

- `echo.` will also display an empty string. However, this is slower than `echo(` as `echo.` will search for a file named "echo". Only if this file does not exist will the command work, but this check makes it slower.
- `echo:` will behave just like `echo(`, unless `message` looks like a file path, e.g. `echo:foo\..\test.bat`. In this case, the interpreter will see `echo:foo` as a folder name, strip `echo:foo\..\` (because it appears just to enter the directory `echo:foo` then leave it again) then execute `test.bat`, which is not the desired behaviour.

Examples

Displaying Messages

To display "Some Text", use the command:

```
echo Some Text
```

This will output the string `Some Text` followed by a new line.

To display the strings `On` and `Off` (case insensitive) or the empty string, use a `(` instead of white-space:

```
echo(ON
echo(
echo(off
```

This will output:

```
ON

off
```

It is also common to use `echo.` to output a blank line, but please see the remarks for why this is not the best idea.

To display text without including a CR/LF, use the following command:

```
<nul set/p=Some Text
```

This command will attempt to set the variable called the empty string to the user input following a prompt. The `nul` file is redirected to the command with `<nul`, so the command will give up as soon as it tries to read from it and only the prompt string will be left. Because the user never typed a new line, there is no linefeed.

Echo Setting

The echo setting determines whether command echoing is on or off. This is what a sample program looks like with command echoing **on** (default):

```
C:\Windows\System32>echo Hello, World!
Hello, World!

C:\Windows\System32>where explorer
C:\Windows\System32\explorer.exe

C:\Windows\System32>exit
```

This is what it looks like with echo **off**:

```
Hello, World!
C:\Windows\System32\explorer.exe
```

Getting and Setting

To get (display) the echo setting, use `echo` with no parameters:

```
> echo
ECHO is on.
```

To set the echo setting, use `echo` with `on` or `off`:

```
> echo off

> echo
ECHO is off.

> echo on

> echo
ECHO is on.
```

Note that with these examples, the prompt has been represented by `>`. When changing the echo setting, the prompt will appear and disappear, but that makes for unclear examples.

Note that it is possible to prevent a command from being echoed even when echo is on, by placing an `@` character before the command.

Echo outputs everything literally

Quotes will be output as-is:

```
echo "Some Text"
```

```
"Some Text"
```

Comment tokens are ignored:

```
echo Hello World REM this is not a comment because it is being echoed!
```

```
Hello World REM this is not a comment because it is being echoed!
```

However, `echo` will still output variables - see Variable Usage - and special characters still take effect:

```
echo hello && echo world
```

```
hello
world
```

Echo output to file

Ways to create a file with the echo command:

```
echo. > example.bat (creates an empty file called "example.bat")

echo message > example.bat (creates example.bat containing "message")
echo message >> example.bat (adds "message" to a new line in example.bat)
(echo message) >> example.bat (same as above, just another way to write it)
```

Output to path

A little problem you might run into when doing this:

```
echo Hello how are you? > C:\Users\Ben Tearzz\Desktop\example.bat

(This will NOT make a file on the Desktop, and might show an error message)
```

But then how do we do it? Well it's actually extremely simple... When typing a path or file name that has a space included in it's name, then remember to use "quotes"

```
echo Hello how are you? > "C:\Users\Ben Tearzz\Desktop\example.bat"
(This will make a file on MY Desktop)
```

But what if you want to make a file that outputs a new file?

```
echo message > file1.bat > example.bat

(This is NOT going to output:
"message > file1.bat" to example.bat
```

Then how do we do this?

```
echo message ^> file1.bat > example.bat

(This will output:
"message > file1.bat" to example.bat
```

Same goes for other stuff in batch

If you haven't learned what variables and statements are, then you most likely won't understand the following: click here to learn about variables | click here to learn about "if" statements

Variables:

```
set example="text"
echo %example% > file.bat
(This will output "text" to file.bat)
```

if we don't want it to output "text" but just plain %example% then write:

```
echo ^%example^% > file.bat
(This will output "%example%" to file.bat)
```

else statements:

```
else = ||
if ^%example^%=="Hello" echo True || echo False > file.bat

(This will output:
if %example%=="Hello" echo True
```

to output the whole line we write:

```
if ^%example^%=="Hello" echo True ^|^| echo False > file.bat

This will output:
if %example%=="Hello" echo True || echo False
```

If the variable is equal to "Hello" then it will say "True", else it will say "False"

@Echo off

`@echo off` prevents the prompt and contents of the batch file from being displayed, so that only the output is visible. The `@` makes the output of the `echo off` command hidden as well.

Turning echo on inside brackets

In the following example `echo on` will take effect after the end of the brackets context is reached:

```
@echo off
(
    echo on
    echo ##
)
echo $$
```

In order to "activate" echo on in a brackets context (including `FOR` and `IF` commands) you can use FOR /f macro :

```
@echo off
setlocal

:: echo on macro should followed by the command you want to execute with echo turned on
set "echo_on=echo on&for %%. in (.) do"

(
  %echo_on% echo ###
)
```

Read Echo online: https://riptutorial.com/batch-file/topic/3981/echo

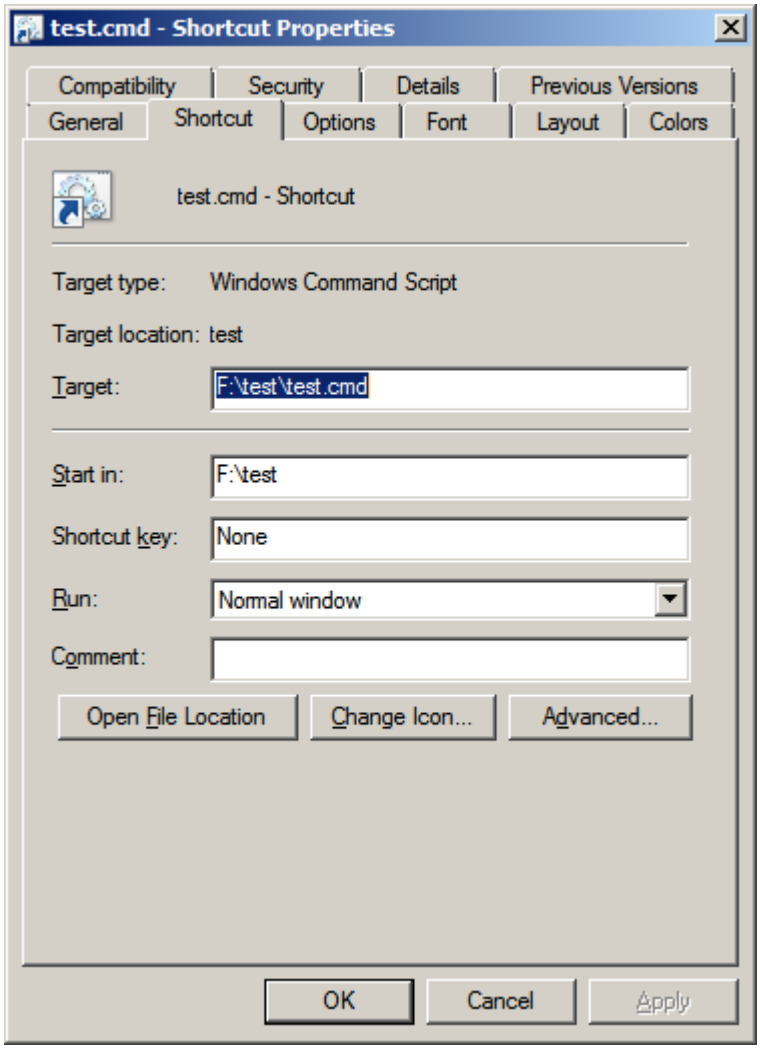
Chapter 18: Elevated Privileges in Batch Files

Examples

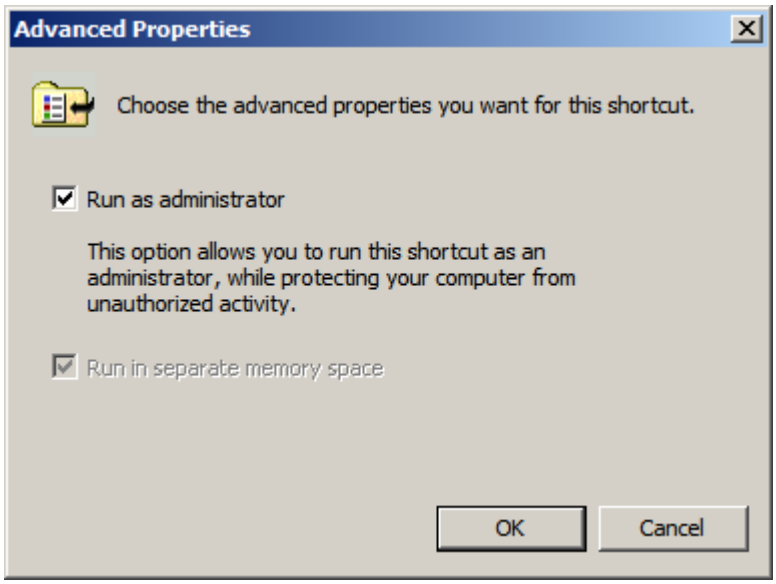
Requesting Elevate Privileges in a Shortcut

Many tasks require elevated privileges. You can elevate user privileges to Administrator level for your batch runtime using a shortcut:

1. Press `alt`+ and the batch file to a selected folder to create a shortcut.
2. Right click and select "Properties".
3. Select the "Shortcut" tab.
4. Click "Advanced".



5. Enable "Run as Administrator".



6. Click "OK" twice.

Requesting Elevated Privileges at Runtime

The following batch file will popup a UAC Prompt allowing you to accept elevated Administrator privileges for the batch session. Add your tasks code to `:usercode` section of the batch, so they run with elevated privileges.

```
@echo off
setlocal EnableDelayedExpansion
:: test and acquire admin rights
cd /d %~dp0 & echo/
if not "%1"=="UAC" (
    >nul 2>&1 net file && echo Got admin rights || (echo No admin rights & ^
MSHTA "javascript: var shell = new ActiveXObject('shell.application');
shell.ShellExecute("%~snx0", 'UAC', '', 'runas', 1);close();"))
:: re-test admin rights
echo/ & >nul 2>&1 net file && (echo Got admin rights & echo/) || (echo No admin rights.
Exiting... & goto :end)

:usercode
:: add your code here
echo Performing admin tasks
echo Hello >C:\test.txt

:end
timeout /t 5 >nul
exit /b
```

Requesting runtime elevated privileges without UAC prompt

As previous example, this script request elevation if needed. We ask the user for credentials avoiding the UAC prompt.

```
@echo off

cls & set "user=" & set "pass="
```

```
:: check if we have administrative access
:: ---------------------------------------------------------------------
whoami /groups | find "S-1-5-32-544">NUL 2>&1 && goto:gotAdmin


echo/
echo/ Testing administrative privileges
echo/
echo/ Please enter administrative credentials
echo/

:: get user
:: ---------------------------------------------------------------------
set/P user="*     User:: "
if "%user%" EQU "" exit/B 1
:: get password
:: ---------------------------------------------------------------------
<NUL set/p=* password& call:getPass pass || exit/B 1
if "%pass%" EQU "" exit/B 1

:: check if credential has administrative privileges
:: this call can be removed
:: ---------------------------------------------------------------------
call:askIsAdmin "%user%", "%pass%" || goto:notAdmin

:: run elevated without UAC prompt
:: with the previous call enabled, we are sure credentials are right
:: without it, this will fail if credentials are invalid
:: ---------------------------------------------------------------------
call:elevateScript "%user%", "%pass%"

exit/B 0

:gotAdmin
echo(
echo( You have administrative rights.
echo(
timeout /T 7
exit/B 0

:notAdmin
echo(
echo( Invalid credential or non administrative account privileges
echo(
timeout /T 7
exit/B 1

:: invoke powershell to get the password
:: ---------------------------------------------------------------------
:getPass
SetLocal
set "psCmd=powershell -Command "$pwd = read-host ':' -AsSecureString;
$BSTR=[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($pwd);
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)""
for /F "usebackq delims=" %%# in (`%psCmd%`) do set "pwd=%%#"
if "%pwd%" EQU "" EndLocal & exit/B 1
EndLocal & set "%1=%pwd%"
doskey /listsize=0 >NUL 2>&1 & doskey /listsize=50 >NUL 2>&1        & rem empty keyboard
buffer
exit/B 0
```

```
:: check if credential has administrative privileges
:: -----------------------------------------------------------------------
:askIsAdmin
SetLocal & set "u=%~1" & set "p=%~2" & set/A ret=1
set "psCmd=powershell -Command "$p='%p%'^|convertto-securestring -asplaintext -force;$c=new-
object -typename system.management.automation.pscredential('%u%',$p^);start-process
'powershell' '-Command "write-host ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent(^)^).IsInRole([Security.Principal.WindowsBuiltInRole]
-credential $c -passthru -wait;""
for /F "usebackq delims=" %%# in (`%psCmd%`) do @set "result=%%#"
echo %result% | find /I "true">NUL 2>&1 && set/A ret=0
EndLocal & exit/B %ret%
exit/B 1

:: run elevated without UAC prompt
:: -----------------------------------------------------------------------
:elevateScript
SetLocal & set "u=%~1" & set "p=%~2"
set "_vbs_file_=%TEMP%\runadmin.vbs"
echo set oWS ^= CreateObject^("wScript.Shell"^)>"%_vbs_file_%"
echo strcmd="C:\Windows\system32\runas.exe /user:%COMPUTERNAME%\%u% " +
"""%~dpnx0""">>"%_vbs_file_%"
echo oWS.run strcmd, 2, false>>"%_vbs_file_%"
echo wScript.Sleep 100>>%_vbs_file_%
echo oWS.SendKeys "%p%{ENTER}">>%_vbs_file_%
if exist "%TEMP%\runadmin.vbs" (set "_spawn_=%TEMP%\runadmin.vbs") else (set
"_spawn_=runadmin.vbs")
ping 1.1.1.1 -n 1 -w 50 >NUL
start /B /WAIT cmd /C "cls & "%_spawn_%" & del /F /Q "%_spawn_%" 2>NUL"
EndLocal & set "pass="
exit/B 0
```

Read Elevated Privileges in Batch Files online: https://riptutorial.com/batch-file/topic/4898/elevated-privileges-in-batch-files

Chapter 19: Escaping special characters

Introduction

In all `cmd.exe` and `DOS` version, some characters are reserved for specific usage(e.g. command redirection). This topic will talk about how to use the special characters without issues.

Examples

Escape using caret(^)

Most special characters can be escaped using the caret(`^`). Take a look at the following example.

```
echo > Hi
echo ^> Hi
```

This first command would not output `> Hi` because `>` is a special character, which means redirect output to a file. In this case, the file is named "Hi"

However in the second command, `> Hi` would be outputted without any issue because the caret(`^`) tells the `>` to stop functioning as "redirect output to file" command, now `>` is just a normal character.

Here's a list of special characters that can be escaped(taken, and edited from Rob van der Woude's page)

Character	Escaped Result	Remarks
^	^^	
&	^&	
<	^<	
>	^>	
|	^|	
\	^\	
!	^^!	Only required when DelayedExpansion is on

Escaping the caret

Carets can be stacked up to the escape other carets, consider the following example.

Input	Output
^&	&
^^^&	^&
^^^^^&	^^&

Note: The carets in bold form are escaped.

Security issue

A bit off topic here, but this is very important! An unwanted caret escape at the end of the file could cause a memory leak!

```
any-invalid-command-you-like-here ^
```

This command would leak all the memory, **rendering the system completely unusable**! See here for more information.

FIND and FINDSTR Special Characters

In `find` and `findstr`, there are some special characters that require some caution on it.

FIND

There is only one character that needs escaping - `"` quote. To escape it, simply add another quote next to it. So `"` becomes `""`. Pretty simple.

FINDSTR

`Findstr` comes with plenty of characters to escape, so please be very cautious. Using `\`, we can escape special characters. Here's a list of special characters to escape

Character	Escaped Result
\	\\
[	\[
]	\]

Character	Escaped Result
"	\"
.	\.
*	*
?	\?

FOR /F Special Characters

FOR /F

In a `FOR /F` statement, some characters needs escaping, here a list(taken and edited from Rob van der Woude's page)

Character	Escaped Result	Remarks
'	^'	Only needed in `FOR /F`'s brackets, unless `usebackq` is specified.
`	^`	Only needed in `FOR /F`'s brackets, when `usebackq` is specified
,	^,	┐
;	^;	┃
=	^=	┣ Must be escaped in `FOR /F`'s brackets, even if it is double-quoted
(	^(	┃
)	^)	┛

Extra Special Characters

Here is a list of other special character(s), that require(s)/may need escaping, but not mentioned above.

Character	Escaped Result	Remarks
%	%%	
[LF]	^[LF]	This trick is metioned by Mark Stang in the `alt.msdos.batch` news group.

Escaping through the pipeline

When there's an expression with a pipe the `cmd` starts two threads on both sides of the pipe and the expression is parsed twice (for each side of the pipe) so carets need to be doubled.

On the left side:

```
echo ^^^&|more
```

On the right side:

```
break|echo ^^^&
```

Read Escaping special characters online: https://riptutorial.com/batch-file/topic/10693/escaping-special-characters

Chapter 20: File Handling in batch files

Introduction

In this topic you will learn how to create, edit, copy, move, and delete files in batch.

Examples

Creating a File in Batch

There may be multiple reason why you want to create a text file in batch. But whatever the reason may be, this is how you do it.

If you want to overwrite an existing text file use `>`. Example:

```
@echo off
echo info to save > text.txt
```

But if you want to append text to an already existing text file use `>>`. Example:

```
@echo off
echo info to save >> text.txt
```

If you need to save multiple lines of text to a file use `()>text.txt` Example:

```
@echo off
(echo username
 echo password)>text.txt
```

How to Copy Files in Batch

You may want to copy files from one place to another. In this example we'll teach you.

You can use the command `xcopy`. The syntax is `xcopy c:\From C:\To`

Example:

```
@echo off
xcopy C:\Folder\text.txt C:\User\Username\Desktop
```

There are also switches you can use. If you want to view them open up command prompt by `Start Menu -> Accessories -> Command Prompt` and then type `xcopy /?`

Moving Files

Using the move command, you can move files:

```
@echo off
cd C:\Foo\Bat\Baz
move /Y Meow.bat "Meow Folder" >nul
```

`Meow.bat` stands for which file to move. The "Meow Folder" moves `Meow.bat` to the `Meow Folder`. `/Y` says to not prompt for confirmation. Replacing that with `/-Y` makes the batch file prompt for confirmation. The `>nul` hides the command output. If it didn't have `>nul`, it would output:

```
    1 File Moved
```

Deleting Files

Using the `DEL`(alias for `ERASE`) command, one can remove files.

```
@echo off
del foo.ext
```

This command will delete `foo.ext` from the current directory. One can also specify path and file, such as:

```
del C:\Foo\Bar\Baz.ext
```

But it is always ideal to put quotes (`"`) around paths, see here for the reason.

There are a few flags available for `DEL`.

Flag	Function
`/P`	Prompts user before deleting file(s)
`/F`	Forcefully remove read-only file(s)
`/S`	Remove file(s) in subdirectories
`/Q`	Prevents the user prompt
`/A`	Filter: Only remove specific attributed file,
	using the `-` character means **not** attributed as that type.

Copy Files Without xcopy

In this example, user BoeNoe showed how to use the command `xcopy` to copy files. There is also an extra command called `copy`.

Here is a simple example:

```
copy foo.ext bar.ext
```

This copies `foo.ext` to `bar.ext`, and create `bar.ext` when it doesn't exist. We can also specify paths to the file, but it is always ideal to put quotes (`"`) around paths, see here for the reason.

There are also many flags available for `copy`, see `copy /?` or `help copy` on a command prompt to see more.

Editing Nth Line of a File

Batch file does not come with a built-in method for replacing `n`th line of a file except `replace` and `append`(`>` and `>>`). Using `for` loops, we can emulate this kind of function.

```
@echo off
set file=new2.txt

call :replaceLine "%file%" 3 "stringResult"

type "%file%"
pause
exit /b

:replaceLine <fileName> <changeLine> <stringResult>
setlocal enableDelayedExpansion

set /a lineCount=%~2-1

for /f %%G in (%~1) do (
    if !lineCount! equ 0 pause & goto :changeLine
    echo %%G>>temp.txt
    set /a lineCount-=1
)

:changeLine
echo %~3>>temp.txt

for /f "skip=%~2" %%G in (%~1) do (
    echo %%G>>temp.txt
)

type temp.txt>%~1
del /f /q temp.txt

endlocal
exit /b
```

- The main script calls the function `replaceLine`, with the filename/ which line to change/ and the string to replace.
- Function receives the input
    - It loops through all the lines and `echo` them to a temporary file before the replacement line

    - It `echo`es the replacement line to the file
    - It continues to output to rest of the file
    - It copies the temporary file to the original file
    - And removes the temporary file.

- The main script gets the control back, and `type` the result.

Read File Handling in batch files online: https://riptutorial.com/batch-file/topic/9253/file-handling-in-batch-files

Chapter 21: For Loops in Batch Files

Syntax

- for /l %%p in (startNumber, increment, endNumber) do command
- for /f %%p in (filename) do command
- for /f %%p in ("textStrings") do command
- for /f %%p in ('command') do command
- for /r drive:\path %%p in (set) do command
- for /d %%p in (directory) do command

Remarks

The `for` command accepts options when the `/f` flag is used. Here's a list of options that can be used:

- `delims=x` Delimiter character(s) to separate tokens
- `skip=n` Number of lines to skip at the beginning of file and text strings
- `eol=;` Character at the start of each line to indicate a comment
- `tokens=n` Numbered items to read from each line or string to process
- `usebackq` Use another quoting style:

  Use double quotes for long file names in "files"

  Use single quotes for 'textStrings'

  Use back quotes for `command`

Examples

Looping through each line in a files set

The following will echo each line in the file `C:\scripts\testFile.txt`. Blank lines will not be processed.

```
for /F "tokens=*" %%A in (C:\scripts\testFile.txt) do (
  echo %%A
  rem do other stuff here
  )
```

More advanced example shows, how derived in FOR loop from a restricted files set data can be used to redirect batch execution, while saving the searched content to a file:

```
@echo off
setlocal enabledelayedexpansion

for /f %%i in ('dir "%temp%\test*.log" /o:-d /t:w /b') do (
    set "last=%temp%\%%i"
    type !last! | find /n /i "Completed" >nul 2>&1 >> %temp%\Completed.log ^
     && (echo Found in log %%i & goto :end) || (echo Not found in log %%i & set "result=1"))

:: add user tasks code here
if defined result echo Performing user tasks...

:end
echo All tasks completed
exit /b
```

Note, how long command strings are split into several code lines, and command groups are separated by parentheses.

Recursively Visit Directories in a Directory Tree

`for /r` command can be used to recursively visit all the directories in a directory tree and perform a command.

```
@echo off
rem start at the top of the tree to visit and loop though each directory
for /r %%a in (.) do (
  rem enter the directory
  pushd %%a
  echo In directory:
  cd
  rem leave the directory
  popd
  )
```

Notes:

- for /r - Loop through files (Recurse subfolders).
- pushd - Change the current directory/folder and store the previous folder/path for use by the POPD command.
- popd - Change directory back to the path/folder most recently stored by the PUSHD command.

Renaming all files in the current directory

The following uses a variable with a `for` loop to rename a group of files.

```
SetLocal EnableDelayedExpansion

for %%j in (*.*) do (
  set filename=%%~nj
  set filename=!filename:old=new!
  set filename=Prefix !filename!
  set filename=!filename! Suffix
  ren "%%j" "!filename!%%~xj"
```

```
)
```

By defining the variable name `%%j` and associating it with all current files `(*.*)`, we can use the variable in a `for` loop to represent each file in the current directory.

Every iteration (or pass) through the loop thereby processes a different file from the defined group (which might equally have been any group, e.g. `*.jpg` or `*.txt`).

In the first example, we substitute text: the text string "old" is replaced by the text string "new" (if, but only if, the text "old" is present in the file's name).

In the second example, we add text: the text "Prefix " is added to the start of the file name. This is not a substitution. This change will be applied to all files in the group.

In the third example, again we add text: the text " Suffix" is added to the end of the file name. Again, this is not a substitution. This change will be applied to all files in the group.

The final line actually handles the renaming.

Iteration

```
for /L %%A in (1,2,40) do echo %%A
```

This line will iterate from 1 to 39, increasing by 2 each time.

The first parameter, `1`, is the starting number.

The second parameter, `2`, is the increment.

The third parameter, `40`, is the maximum.

Read For Loops in Batch Files online: https://riptutorial.com/batch-file/topic/3695/for-loops-in-batch-files

Chapter 22: Functions

Remarks

You can add starting variables to the function by adding `<parameter>` to it's label. These starting variables can be accessed with `%n` where n is the starting variable's number (`%1` for the first, `%2` for the second. This `%n` method works for %1 - %9. For parameter 10 - 255, you will need to use the Shift command).
For example:

```
:function <var1> <var2>
```

Once you use `call :function param1 param2`, `param1` can be accessed with `%1`, and `param2` with `%2`.
Note: the `<parameter>` isn't strictly necessary, but it helps with readability.

A neat trick that is useful when many variable are flying about is to use `setlocal` and `endlocal` in tandem with `%n`. `setlocal` and `endlocal` essentially make the function it's own separate instance of the command prompt, variables set in it only stick around while it's in the frame.

If you are using `setlocal` and `endlocal`, and you are returning global values use this.

```
endlocal & set var=variable
```

This sets the global value `var` to `variable`. You can chain these together for multiple variables.

```
endlocal & set var=variable & set var2=variable number 2
```

This sets the global variable `var` to `variable` and the global value `var2` to `variable number 2`.
Since code in code blocks are also performed simultaneously, you can do this as well.

```
if "%var%"=="" (
    endlocal
    set %~2=10
)
```

But, you **cannot** do this.

```
if "%var%"=="" (
    set %~2=10
    endlocal
)
```

Examples

Simple Function

```
call :FunctionX
rem More code...

:FunctionX
rem Some code here.
goto :eof
```

This is a very simple function. Functions are in-program commands that do multiple commands at a time. Functions are made by creating a label and putting code in it, and once it is done, you add a `goto :eof` or `exit /b <ErrorlevelYou'dLike>` which returns to where it was invoked. Functions are invoked with `call :functionname adparams`.

Function With Parameters

```
call :tohex 14 result
rem More code...

:tohex <innum> <outvar>
set dec=%1
set outvar=%~2
rem %n and %~n are functionally identical, but %~n is slightly safer.
goto :eof
```

This takes the additional parameters from the `call` as if the function was a separate Batch file. Note: the `<parameter>` isn't necessary, but it helps with readability.

Function Utilizing setlocal and endlocal

```
set var1=123456789
set var2=abcdef
call :specialvars
echo %var1%, %var2%
rem More code...

:specialvars
setlocal
set var1=987654321
set var2=fedcba
endlocal
goto :eof
```

When inside the section `setlocal` , `endlocal` section, variables are seperate from the caller's variables, hence why %var1% and %var2% weren't changed.

Combining them all

```
set importantvar=importantstuff
call :stuff 123 var1
rem More code...

:stuff <arg1> <arg2>
setlocal
set importantvar=%~1
```

```
echo Writing some stuff into %~2!
endlocal
set %~2=some stuff
setlocal
set importantvar=junk
endlocal
goto :eof
```

This utilizes the basic function, `setlocal` and `endlocal` and arguments to create an odd little function.

Anonymous functions in batch files

Anonymous functions technique uses the fact that `CALL` command uses internally `GOTO` when subroutine is called and abusing help message printing with variable double expansion:

```
@echo off
setlocal
set "anonymous=/?"

call :%%anonymous%% a b c 3>&1 >nul

if "%0" == ":%anonymous%" (
  echo(
  echo Anonymous call:
  echo %%1=%1 %%2=%2 %%3=%3
  exit /b 0
)>&3
```

You can call an anonymous function only if it is defined after the `CALL` (or after finishing brackets context if the `CALL` is executed within brackets). It cannot be called from an outside script ,but is a slower than normal function call.

Calling functions from another batch file

Lets have the following file called **library.cmd** :

```
@echo off

echo -/-/- Batch Functions Library -/-/-

:function1
    echo argument1 - %1
    goto :eof
```

To execute only the **:function1** without the code of the rest of the file you should put a label **:function1** in the caller bat and use it like this:

```
@echo off

call :function1 ###
exit /b %errorlevel%
```

```
:function1
    library.bat %*
```

the output will be (the code outside the function in `library.cmd` is not executed):

    argument1 - ###

For more info check this.

Read Functions online: https://riptutorial.com/batch-file/topic/7646/functions

Chapter 23: If statements

Syntax

- if [/i] StringToCompare1 == StringToCompare2 (commandA) else (commandB)
- if errorlevel 1 (commandA) else (commandB)
- if %errorlevel% == 1 (commandA) else (commandB)
- if exist Filename (commandA) else (commandB)
- if defined VariableName (commandA) else (commandB)

Remarks

There are a few syntax to choose from in an `if` statement. We will use `if string1==string2` as an example.

1-Line Syntaxes

- `if string1==string2 commandA`
- `if string1==string2 (commandA)`

- `if string1==string2 (commandA) else (commandB)`
- `if string1==string2 (commandA) else commandB`

- `if string1==string2 (commandA)else (commandB)`
- `if string1==string2 (commandA)else commandB`

Multiline Syntaxes

```
if string1==string2 (
    commandA
)
```

Or

```
if string1==string2 (
    commandA
) else (
    commandB
)
```

There are still some extra syntaxes available.

Examples

Comparing numbers with IF statement

```
SET TEST=0

IF %TEST% == 0 (
    echo TEST FAILED
) ELSE IF %TEST% == 1 (
    echo TEST PASSED
) ELSE (
    echo TEST INVALID
)
```

Comparing strings

```
IF "%~1" == "-help" (
    ECHO "Hello"
)
```

where `%1` refers to the first command line argument and `~` removes any quotes that were included when the script was called.

Comparing Errorlevel

```
If Errorlevel 1 (
    Echo Errorlevel is 1 or higher

    REM The phrase "1 or higher" is used because If Errorlevel 1 statement means:
    REM                                          If %Errorlevel% GEQ 1
    REM                                    Not If %Errorlevel% EQU 1
)
```

or

```
If "%Errorlevel%"=="1" (
    Echo Errorlevel is 1
)
```

The script above would check the variable Errorlevel(built-in). The `not` operator can be used.

```
Set "Test=%Errorlevel%"

If "%Test%" == "1" (
    Echo Errorlevel is 1
)
```

This one also works.

Please note that some commands **do not affect the errorlevel**:

- Break
- Echo
- Endlocal
- For
- If
- Pause
- Rem
- Rd / Rmdir
- Set
- Title

The following commands **set but not clear errorlevel**:

- Cls
- Goto
- Keys
- Popd
- Shift

The following commands **set exit codes but not the errorlevel**:

- Rd / Rmdir

The following commands **set errorlevel but not the exit codes**:

- Md / Mkdir

Check if file exists

```
If exist "C:\Foo\Bar.baz" (
    Echo File exist
)
```

This checks if the file C:\Foo\Bar.baz's existence. If this exist, it echos File exist The `Not` operator can also be added.

If variable exists / set

```
If Defined Foo (
    Echo Foo is defined
)
```

This would check if a variable is defined or not. Again, the `Not` operator can be used.

Read If statements online: https://riptutorial.com/batch-file/topic/5475/if-statements

Chapter 24: Input and output redirection

Syntax

- [command] [[> | >> | < | 2> | 2>>] file]
- [[> | >> | < | 2> | 2>>] file] [command]

Parameters

Parameter	Details
command	Any valid command.
>	Write STDOUT to file.
>>	Append STDOUT to file.
<	Read file to STDIN.
2>	Write STDERR to file.
2>>	Append STDERR to file.
file	The path to a file.

Remarks

- You can add as many different redirections as you want, so long as the redirection symbol and file remain together and in the correct order.

Examples

An Example...

```
@echo off
setlocal
set /p "_myvar=what is your name?"
echo HELLO!>file.txt
echo %_myvar%!>>file.txt
echo done!
pause
type file.txt
endlocal
exit
```

Now file.txt looks like:

```
HELLO!
John Smith!
```

(assuming you typed `John Smith` as your name.)

Now your batch file's console looks like:

```
what is your name?John Smith
done!
Press any key to continue...
HELLO!
John Smith!
```

(and it should exit so quickly that you may not be able to see anything after the prompt `Press any key to coninue...`)

Redirect special character with delayed expansion enabled

This example echoes the special character `!` into a file. This would only work when DelayedExpansion is disabled. When delayed expansion in enabled, you will need to use three carets and an exclamation mark like this:

```
@echo off
setlocal enabledelayedexpansion


echo ^^^!>file
echo ^>>>file

goto :eof

    ^> is the text
    >> is the redirect operator

pause
endlocal
exit /b
```

This code will echo the following text into the file

```
!
>
```

as

```
^^^ escapes the ! and echos it into the file
^> escapes the > and echos it into the file
```

Write to a file

```
@echo off
```

```
cls
echo Please input the file path, surrounded by "double quotation marks" if necessary.
REM If you don't want to redirect, escape the > by preceding it with ^
set /p filepath=^>

echo Writing a random number
echo %RANDOM% > %filepath%
echo Reading the random number
type %filepath%

REM Successive file writes will overwrite the previous file contents
echo Writing the current directory tree:
> %filepath% tree /A
echo Reading the file
type %filepath%

REM nul is a special file. It is always empty, no matter what you write to it.
echo Writing to nul
type %windir%\win.ini > nul
echo Reading from nul
type nul

echo Writing nul's contents to the file
type nul > %filepath%
echo Reading the file
type %filepath%
```

Read Input and output redirection online: https://riptutorial.com/batch-file/topic/7502/input-and-output-redirection

Chapter 25: Random In Batch Files

Examples

Random Numbers

Using the dynamic variable `%Random%`, we can get a random integer from 0 to 32767. For example:

```
echo %random%
```

This obviously, returns an integer from 0 to 32767. But sometimes we want it to be in a specific range, say from 1 to 100.

Generating Random Numbers Within Specific Range

The basic method to do so is listed below.

```
set /a result=(%RANDOM%*max/32768)+min
```

where `max` is the top number that can be generated, and `min` is the smallest number that can be generated. Note that you will not get any decimal numbers because `set /a` rounds down automatically. To generate a decimal random number, try this:

```
set /a whole=(%RANDOM%*max/32768)+min
set /a decimal=(%RANDOM%*max/32768)+min
echo %whole%.%decimal%
```

Generating Random Numbers larger than 32767

If you try

```
set /a whole=(%RANDOM%*65536/32768)+1
```

you will most likely get random numbers that are odd.

To generate numbers larger than 32767, here is a better method.

```
set /a result=%random:~-1%%random:~-1%%random:~-1%%random:~-1%%random:~-1%%random:~-1%
```

The previous code extracts the 1 character from each `%random%`. But this is done on purpose.

Since the `random` number could be one digit number, extracting the last 2 digit won't work. That's why we extract only the last character. In this case, we have 6 `%random:~-1%`, generating the

maximum of `999999`, and the minimum at `000000`, you may need to adjust this to suit your needs.

Pseudorandom

`cmd.exe` generate the seed based on the time the `cmd` section started, so if you start mutliple section at the nearly same time, the result may not be 'random' enough.

Random Alphabets

Unfortunately, batch does not have a built-in method to generate alphabets, but using `%random%` and `for` loop, we can 'generate' alphabets.

This is a simple idea of how this works.

```
set /a result=(%random%*26/32768)+1
for /f "tokens=%result%" %%I in ("A B C D E F G H I J K L M N O P Q R S T U V W X Y Z") do (
    echo %%I
)
```

- The first `set /a` statement generate a random number `N` between 1 to 26
- The `for /f` statement picks the `N`th item from a list of A to Z.
    - Return the result

One can put a total `31` items in 1 `for` loop, and practically unlimited items using [this method].(Batch - for loop parameter order

Pseudorandom And Uniform Random In Batch

Pseudorandom Distribution

Accorinding to this Stack Overflow answer, user CherryDT pointed out this code:

```
set /a num=%random% %% 100
```

does not give a uniform distribution.

The internal dynamic variable `%random%` **does** gives a **uniform distribution**, but the above code will not be a uniformed random. This code generates a random number between 0 ~ 99, but the result will not be uniform. 0 ~ 67 will occur more than 68 ~ 99 since `32767 MOD 100` = `67`.

To generate a uniform distributed random using the above code, then `100` must be changed. Here is a method to get a number that creates a uniform distribution.

```
32767 mod (32767 / n)
```

where `n` is an integer, between 0 ~ 32767, the result may be decimal and may not work in batch.

Uniform Distribution

```
set /a result=(%RANDOM%*100/32768)+1
```

This method will generate a uniform distribution. It avoids using `%`, which is more like "remainder" then "modulus" in a batch script. Without using `%`, the result will be uniform.

Alternatively, here is an inefficient, but uniform method.

```
set /a test=%random%

if %test% geq [yourMinNumber] (
    if %test% leq [yourMaxNumber] (

        rem do something with your random number that is in the range.

   )
)
```

Change `[yourMinNumber]` and `[yourMaxNumber]` accordingly to your own values.

Read Random In Batch Files online: https://riptutorial.com/batch-file/topic/10841/random-in-batch-files

Chapter 26: Search strings in batch

Examples

Basic strings search

FIND command can scan large files line-by-line to find a certain string. It doesn't support wildcards in the search string.

```
find /i "Completed" "%userprofile%\Downloads\*.log" >> %targetdir%\tested.log

TYPE scan2.txt | FIND "Failed" /c && echo Scan failed || echo Scan Succeeded
```

FINDSTR command is more feature reach, and supports Regular Expressions (REGEX) search with wildcards in the search string.

```
FINDSTR /L /C:"Completed" Results.txt

echo %%G | findstr /r /b /c:"[ ]*staff.*" >nul && echo Found!
```

See FIND and FINDSTR help sources for more information.

Using search results

The following script shows more advanced *split file* technique, where FOR function loops through a list of files in a directory, and each file content is piped to FINDSTR that looks for a string containing substring `var` preceded by undefined number of spaces and superseded by any extra text. Once found, the searched file is replaced with a new one that contains only the text portion above the search string.

```
@echo off
setlocal enabledelayedexpansion
pushd "%temp%\Test"
for %%G in ("*.txt") do (set "break="
    (for /f "tokens=*" %%H in (%%~G) do (
        if not defined break (
            echo %%H | findstr /r /b /c:"[ ]*var.*" >nul && set break=TRUE || echo %%H )
    )) >> %%~nG_mod.txt
    del %%~G & ren %%~nG_mod.txt %%G )
popd
exit /b
```

Note, how setting `break=TRUE` allows to exit FOR loop from the file searched, once the 1st occurrence of the search string is found.

Read Search strings in batch online: https://riptutorial.com/batch-file/topic/5476/search-strings-in-batch

Chapter 27: Using Goto

Introduction

Goto Is simple. By using simple goto statements, you can move anywhere you want to in your code. It can be also used to make functions (Showed in how to make functions).

Syntax

- goto :Label
- goto Label
- goto :EOF

Parameters

Parameter	Details
`:Label`	Any label that is valid (defined by `:<LabelName>`)
`:EOF`	A pre-defined label that exits the current script of function(same as `exit /b`)

Remarks

So in other words, if the number the player inserted is 1, it'll go back to the :Name part of the code.

so if the input is equal to 1, go back to the line with :Name

Make Sure if you use this, the word begins with the Colen (:).

Examples

Example Programs

For Example:

```
echo Hello!
pause >nul
:Name
echo What Is Your Name
set /p Input=Name:
echo so %Input% Is Your Name, right?
echo Rename?
echo 1 For Yes
echo 2 For No
set /p Input=Rename:
```

```
if %Input%=1 goto Name
```

Another Example:

```
@echo off
echo 1 or 2?
set /p input=Choice:
if %input%=1 goto Skip
echo You Chose 1
pause >nul
echo So time for stuff
pause >nul
echo Random Stuf
pause >nul
:Skip
echo So that's it.
pause >nul
```

Goto with variable

`Goto` accepts the use of variable value to act as the label to goto.

Example:

```
@echo off

echo a = 1
echo b = 2

set /p "foo=Enter option:"
goto %foo%
```

However, you should check the input so it will not go to somewhere that does not exist. Going to an undefined label will terminate your batch script instantly.

Read Using Goto online: https://riptutorial.com/batch-file/topic/9164/using-goto

Chapter 28: Variables in Batch Files

Examples

Declaration

To create a simple variable and assign it to a value or string use the `SET` command:

```
SET var=10
```

Here, the code declares a new variable `var` with a value of `10`. By default all variables are stored internally as strings; this means that the value `10` is no different to `foo1234` or `Hello, World!`

Notes about quotation marks

Quotation marks used will be included in the variable's value:

```
SET var="new value"             <-- %var% == '"new value"'
```

Spaces in variables

Batch language considers spaces to be acceptable parts of variable names. For instance, `set var = 10` will result in a variable called `var ` that contains the value `10` (note the extra space to the right of var and the left of the 10).

Using quotation marks to eliminate spaces

In order to prevent spaces, use quotation marks around the entire assignment; the variable name and value. This also prevents accidental trailing spaces at the end of the line (the ␣ character denotes a space):

```
SET␣var=my␣new␣value␣        <-- '%var%' == 'my new value '
SET␣"var=my␣new␣value"␣       <-- '%var%' == 'my new value'
```

Also, use quotation marks when joining multiple statements with `&` or `|` - alternatively, put the symbol directly after the end of the variable's value:

```
SET var=val & goto :next        <-- '%var%' == 'val '
SET "var=val" & goto :next      <-- '%var%' == 'val'
SET var=val& goto :next         <-- '%var%' == 'val'
```

Usage

```
echo %var%
```

This code will echo the value of `var`

If `setLocal EnableDelayedExpansion` is used, the following will echo the value of `var` (the standard expression %var% will not work in that context).

```
echo !var!
```

In batch files, variables can be used in any context, including as parts of commands or parts of other variables. You may not call a variable prior to defining it.

Using variables as commands:

```
set var=echo
%var% This will be echoed
```

Using variables in other variables:

```
set var=part1
set %var%part2=Hello
echo %part1part2%
```

Variable Substitution

Unlike other programming languages, in a batch file a variable is substituted by its actual value **before** the batch script is run. In other words, the substitution is made when the script is *read* into memory by the command processor, not when the script is later *run*.

This enables the use of variables as commands within the script, and as part of other variable names in the script, etc. The "script" in this context being a line - or block - of code, surrounded by round brackets: `()`.

But this behaviour does mean that you cannot change a variable's value inside a block!

```
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    ECHO %VAR%
    SET VAR=Goodbye
)
```

will print

```
Hello
Hello
```

since (as you see, when watching the script run in the command window) it is evaluated to:

```
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo Hello
    SET VAR=Goodbye
)
```

In the above example, the `ECHO` command is evaluated as `Hello` when the script is read into memory, so the script will echo `Hello` forever, however many passes are made through the script.

The way to achieve the more "traditional" variable behaviour (of the variable being expanded whilst the script is running) is to enable "delayed expansion". This involves adding that command into the script prior to the loop instruction (usually a FOR loop, in a batch script), and using an exclamation mark (!) instead of a percent sign (%) in the variable's name:

```
setlocal enabledelayedexpansion
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo !VAR!
    SET VAR=Goodbye
)
endlocal
```

will print

```
Hello
Goodbye
```

The syntax `%%a in (1,1,2)` causes the loop to run 2 times: on the first occasion, the variable bears its initial value of 'Hello', but on the second pass through the loop - having executed the second SET instruction as the last action on the 1st pass - this has changed to the revised value 'Goodbye'.

Advanced variable substitution

Now, an advanced technique. Using the `CALL` command allows the batch command processor to expand a variable located on the same line of the script. This can deliver multilevel expansion, by repeated CALL and modifier use.

This is useful in, for example, a FOR loop. As in the following example, where we have a numbered list of variables:

```
"c:\MyFiles\test1.txt" "c:\MyFiles\test2.txt" "c:\MyFiles\test3.txt"
```

We can achieve this using the following FOR loop:

```
setlocal enabledelayedexpansion
for %%x in (%*) do (
    set /a "i+=1"
    call set path!i!=%%~!i!
    call echo %%path!i!%%
)
endlocal
```

Output:

```
c:\MyFiles\test1.txt
c:\MyFiles\test2.txt
c:\MyFiles\test3.txt
```

Note that the variable `!i!` is first expanded to its initial value, 1, then the resulting variable, %1, is expanded to its actual value of `c:\MyFiles\test1.txt`. This is *double expansion* of the variable `i`. On the next line, `i` is again double expanded, by use of the `CALL ECHO` command together with the `%%` variable prefix, then printed to the screen (i.e. displayed on screen).

On each successive pass through the loop, the initial number is increased by 1 (due to the code `i+=1`). Thus it increases to `2` on the 2nd pass through the loop, and to `3` on the 3rd pass. Thus the string echoed to the screen alters with each pass.

Declare multiple variables

When multiple variables are defined at the beginning of the batch, a short definition form may be used by employing a replacement string.

```
@echo off
set "vars=_A=good,_B=,_E=bad,_F=,_G=ugly,_C=,_H=,_I=,_J=,_K=,_L=,_D=6
set "%vars:,=" & set "%"

for /f %%l in ('set _') do echo %%l
exit /b

_A=good
_D=6
_E=bad
_G=ugly
```

Note in the above example, variables are natively alphabetically sorted, when printed to screen.

Using a Variable as an Array

It is possible to create a set of variables that can act similar to an array (although they are not an actual array object) by using spaces in the `SET` statement:

```
@echo off
SET var=A "foo bar" 123
for %%a in (%var%) do (
    echo %%a
)
echo Get the variable directly: %var%
```

Result:

```
A
"foo bar"
123
Get the variable directly: A "foo bar" 123
```

It is also possible to declare your variable using indexes so you may retrieve specific information. This will create multiple variables, with the illusion of an array:

```
@echo off
setlocal enabledelayedexpansion
SET var[0]=A
SET var[1]=foo bar
SET var[2]=123
for %%a in (0,1,2) do (
    echo !var[%%a]!
)
echo Get one of the variables directly: %var[1]%
```

Result:

```
A
foo bar
123
Get one of the variables directly: foo bar
```

Note that in the example above, you cannot reference `var` without stating what the desired index is, because `var` does not exist in its own. This example also uses `setlocal enabledelayedexpansion` in conjunction with the exclamation points at `!var[%%a]!`. You can view more information about this in the Variable Substitution Scope Documentation.

Operations on Variables

```
set var=10
set /a var=%var%+10
echo %var%
```

The final value of `var` is 20.

The second line is not working within a command block used for example on an **IF** condition or on a **FOR** loop as delayed expansion would be needed instead of standard environment variable expansion.

Here is another, better way working also in a command block:

```
set var=10
set /A var+=10
echo %var%
```

The command prompt environment supports with signed 32-bit integer values:

- addition `+` and `+=`
- subtraction `-` and `-=`
- multiplication `*` and `*=`
- division `/` and `/=`
- modulus division `%` and `%=`
- bitwise AND `&`

- bitwise OR `|`
- bitwise NOT `~`
- bitwise XOR `^`
- bitwise left shift `<<`
- bitwise right shift `>>`
- logical NOT `!`
- unary minus `-`
- grouping with `(` and `)`

The Windows command interpreter does not support 64-bit integer values or floating point values in arithmetic expressions.

Note: The operator `%` must be written in a batch file as `%%` to be interpreted as operator.

In a command prompt window executing the command line `set /A Value=8 % 3` assigns the value `2` to environment variable `Value` and additionally outputs `2`.

In a batch file must be written `set /A Value=8 %% 3` to assign the value `2` to environment variable `Value` and nothing is output respectively written to handle **STDOUT** (standard output). A line `set /A Value=8 % 3` in a batch file would result in error message *Missing operator* on execution of the batch file.

The environment requires the switch `/A` for arithmetic operations only, not for ordinary string variables.

Every string in the arithmetic expression after `set /A` being whether a number nor an operator is automatically interpreted as name of an environment variable.

For that reason referencing the value of a variable with `%variable%` or with `!variable!` is not necessary when the variable name consists only of word characters (0-9A-Za-z_) with first character not being a digit which is especially helpful within a command block starting with `(` and ending with a matching `)`.

Numbers are converted from string to integer with C/C++ function strtol with `base` being zero which means automatic base determination which can easily result in unexpected results.

Example:

```
set Divided=11
set Divisor=3

set /A Quotient=Divided / Divisor
set /A Remainder=Divided %% Divisor

echo %Divided% / %Divisor% = %Quotient%
echo %Divided% %% %Divisor% = %Remainder%

set HexValue1=0x14
set HexValue2=0x0A
set /A Result=(HexValue1 + HexValue2) * -3

echo (%HexValue1% + %HexValue2%) * -3 = (20 + 10) * -3 = %Result%
```

```
set /A Result%%=7
echo -90 %%= 7 = %Result%

set OctalValue=020
set DecimalValue=12
set /A Result=OctalValue - DecimalValue

echo %OctalValue% - %DecimalValue% = 16 - 12 = %Result%
```

The output of this example is:

```
11 / 3 = 3
11 % 3 = 2
(0x14 + 0x0A) * -3 = (20 + 10) * -3 = -90
-90 %= 7 = -6
020 - 12 = 16 - 12 = 4
```

Variables not defined on evaluation of the arithmetic expression are substituted with value 0.

Setting variables from an input

Using the `/p` switch with the `SET` command you can define variables from an Input.

This input can be a user Input (keyboard) :

```
echo Enter your name :
set /p name=
echo Your name is %name%
```

Which can be simplified like this :

```
set /p name=Enter your name :
echo Your name is %name%
```

Or you can get the input from a file :

```
set /p name=< file.txt
```

in this case you'll get the value of the first line from `file.txt`

Getting the value of various line in a file :

```
(
   set /p line1=
   set /p line2=
   set /p line3=

) < file.txt
```

Read Variables in Batch Files online: https://riptutorial.com/batch-file/topic/3528/variables-in-batch-files

Credits

S. No	Chapters	Contributors
1	Getting started with batch-file	BoeNoe, Clijsters, Community, DavidPostill, Derpcode, geisterfurz007, Jeffrey Lin, loadingnow, Mee, rudicangiotti, sambul35, Scott Beeson, Sourav Ghosh, stark, SteveFest, ths
2	Add delay to Batch file	SteveFest, Tearzz, wizzwizz4
3	Batch and JSCript hybrids	npocmaka, SteveFest
4	Batch and VBS hybrids	npocmaka, SteveFest
5	Batch file command line arguments	DavidPostill, LotPings, npocmaka, sambul35
6	Batch file macros	SteveFest
7	Batch files and Powershell hybrids	npocmaka, SteveFest
8	Best Practices	SteveFest
9	Bugs in cmd.exe processor	SteveFest, X. Liu
10	Bypass arithmetic limitations in batch files	npocmaka
11	Changing Directories and Listing their Contents	Aravind .KEN, SomethingDark, SteveFest, wizzwizz4
12	Comments in Batch Files	0x90h, BoeNoe, DavidPostill, Gábor Bakos, JosefZ, LotPings, SachaDee, SomethingDark, Sourav Ghosh, Stephan, SteveFest , wasatchwizard
13	Creating Files using Batch	Aravind .KEN, David Starkey, fruitSalad266, J03L, LeoDog896, loadingnow, Tearzz
14	Deprecated batch commands and their	npocmaka, SteveFest

	replacements	
15	Differences between Batch (Windows) and Terminal (Linux)	SomethingDark, SteveFest
16	Directory Stack	DavidPostill, wizzwizz4
17	Echo	DavidPostill, fruitSalad266, Keith Hall, npocmaka, SomethingDark, Tearzz, wizzwizz4
18	Elevated Privileges in Batch Files	DavidPostill, elzooilogico, sambul35
19	Escaping special characters	npocmaka, SteveFest
20	File Handling in batch files	BoeNoe, LeoDog896, SteveFest
21	For Loops in Batch Files	David Starkey, DavidPostill, Ed999, Mee, sambul35, SomethingDark, SteveFest
22	Functions	ender_scythe, npocmaka, SteveFest
23	If statements	npocmaka, Ozair Kafray, SomethingDark, SteveFest
24	Input and output redirection	cascading-style, SomethingDark, SteveFest, wizzwizz4
25	Random In Batch Files	SteveFest
26	Search strings in batch	sambul35
27	Using Goto	LeoDog896, SteveFest
28	Variables in Batch Files	DavidPostill, Ed999, Jay, Jeffrey Lin, Mee, Mofi, SachaDee, sambul35, SomethingDark, SteveFest, Tearzz, ths, Toomaja, wasatchwizard, wizzwizz4