Syntax : Escape Characters, Delimiters and Quotes

Using "Double Quotes"

If a single parameter contains spaces, you can still pass it as one item by surrounding in "quotes" - this works well for long filenames.

If a parameter is used to supply a filename like this:

```
MyBatch.cmd "C:\Program Files\My Data File.txt"
```

This parameters will be:

```
%0 =MyBatch
%1 ="C:\Program Files\My Data File.txt"
```

To launch a batch script with spaces in the Program Path requiring "quotes"

```
CMD /k ""c:\batch files\test.cmd" "Parameter 1 with space" "Parameter2 with space""
```

In the FIND comand, the " quote can be escaped by doubling it to ""

Removing Quotes

A common method of removing quotes from a variable is the following:

```
:: Remove quotes
SET _string=###%_string%###
SET _string=%_string:"###=%
SET _string=%_string:###"=%
SET _string=%_string:###=%
```

Unfortunately this will fail if the variable is NULL.
Here are some batch routines to remove quotes.

Working without Quotes

Without surrounding quotes, a long filename will be passed as %1 %2 %3...

```
MyBatch C:\Program Files\My Data File.txt
```

To refer to the pathname above use %* rather than %1 %2 %3 - the %* will cover all parameters - even if there are more than %9

You can apply Extended Filename syntax to %* with the following workaround:

```
@ECHO OFF
SET _params=%*

:: pass params to a subroutine
CALL :sub "%_params%"
GOTO :eof

:sub
:: Now display just the filename (not path)
ECHO %~n1
```

Delimiters

Delimiters separate one parameter from the next - they split the command line up into words.

Parameters are most often separated by spaces, but any of the following are also valid delimiters:

Comma (,)
Semicolon (;)
Equals (=)
Space ( )
Tab ( )

Notice that although / and - are commonly used to separate command options, they are absent from the list above. This is because batch file parameters are passed to CMD.exe which can accept it's own parameters (which are invoked using / and -)

One exception to this standard list of delimiters is the FOR command where the default is just [space] and [tab] and you can use the `delims=` option to specify something different.

When using the TAB character as a delimiter be aware that many text editors will insert a TAB as a series of SPACEs.

When you use %* to refer to all parameters, the value returned will include the delimiters, under NT 4.0 this will include the leading space, under Windows 2000 and above it won't.

Escape Character

```
^  Escape character.
```

Adding the escape character before a command symbol allows it to be treated as ordinary text.
When piping or redirecting any of these charcters you should prefix with the escape character: `\ & | > < ^`

```
e.g.  ^\  ^&  ^|  ^>  ^<  ^^
```

The escape character can be used to escape itself `^^` (meaning don't treat the first `^` as an escape character), so you are escaping the escape character:

```
The characters in bold get escaped:
    ^&  =>  &
  ^^^&  =>  ^&
^^^^^&  => ^^&
```

Escaping Percents

The `%` character has a special meaning for command line parameters and FOR parameters.
To treat a percent as a regular character, double it to `%%`

Many characters such as `\, =, (, )` do not need to be escaped when they are used within a `"quoted string"` typically these are charcters you might find in a filename/path. The percent character is one exception to this rule, even though under NTFS `%` is a valid filename character.

Escaping CR/LF line endings.

The `^` escape character can be used to make long commands more readable by splitting them into multiple lines and escaping the Carriage Return + Line Feed (CR/LF) at the end of a line:

```
ROBOCOPY \\FileServ1\e$\users ^
 \\FileServ2\e$\BackupUsers ^
 /COPYALL /B /SEC /MIR ^
 /R:0 /W:0 /LOG:MyLogfile.txt /NFL /NDL
```

One thing to be careful of with this technique is that a stray space at the end of a line (after the ^) will break the command, this can be hard to spot unless you have a text editor that displays spaces and tab characters.

Some commands (e.g. REG and FINDSTR) use the standard escape character of `\` (as used by C, Python, SQL, bash and many other languages.)
The `\` escape can cause problems with quoted directory paths that contain a trailing backslash because the closing quote " at the end of the line will be escaped `\"`.

To save a directory path with a trailing backslash (`\`) requires adding a second backslash to 'escape the escape'
so for example instead of `"C:\My Docs\"` use `"C:\My Docs\\"`

To be sure that a path includes a trailing backslash, you can test for it:

```
Set _prog=C:\Program Files\SS64 App
IF %_prog:~-1% NEQ \ (Set _prog=%_prog%\)
Echo "%_prog%"
```

"All the best stories in the world are but one story in reality - the story of escape. It is the only thing which interests us all and at all times, how to escape" - A. C. Benson

Related:

SETLOCAL EnableDelayedExpansion - More examples, particularly for HTML.
Long Filenames and NTFS - Valid characters in filenames
FINDSTR Escapes and Length limits
cmd Syntax
Powershell Escape Character
In bash use \ to escape a line ending.