Command line parameters

Batch files can only handle parameters %0 to %9

`%0` is the program name *as it was called,*
`%1` is the first command line parameter,
`%2` is the second command line parameter,
and so on till `%9`.

OK, tell me something new.

Since `%0` is the program name *as it was called,* in DOS `%0` will be empty for AUTOEXEC.BAT if started at boot time.
This means that, in AUTOEXEC.BAT, you can check if it is being started at boot time or from the command line, for example to prevent loading TSR's twice.

SHIFT

The batch file's limitation to handle parameters up to `%9` only can be overcome by using SHIFT.
Let us assume your batchfile is called with the command line parameters `A B C D E F G H I J K`.
Now `%1` equals A, `%2` equals B, etcetera, until `%9`, which equals I. However, `%10` does *not* equal J but A0; `%10` is interpreted as `%1`, immediately followed by a 0.
Does that mean the rest of the parameters is lost? Of course not. After your batch file handled its first parameter(s) it could `SHIFT` them (just insert a line with only the command `SHIFT`), resulting in `%1` getting the value B, `%2` getting the value C, etcetera, till `%9`, which now gets the value J.
Continue this process until at least `%9` is empty.
Use a loop to handle any number of command line parameters:

```
@ECHO OFF
:Loop
IF "%1"=="" GOTO Continue
   •
   • Here your batch file handles %1
   •
SHIFT
GOTO Loop
:Continue
```

Note: `IF "%1"==""` will cause problems if `%1` is enclosed in quotes itself.
In that case, use `IF [%1]==[]` or, in NT 4 (SP6) and later only, `IF "%~1"==""` instead.

In Windows NT 4, 2000 and XP you can `SHIFT` the command line parameters starting from the *n*th positions using `SHIFT`'s `/n` switch, where `n` can be any (integer) number between 0 and 8: `SHIFT /4` will leave `%0` through `%3` untouched, and shift `%5` to `%4`, `%6` to `%5`, etcetera.
To use this feature, Command Extensions should be enabled.

An easy work-around in NT 4 and later is:

```
FOR %%A IN (%*) DO (
       •
       • Now your batch file handles %%A instead of %1
       •
)
```

No need to use SHIFT anymore.

Delimiters

Some characters in the command line are ignored ***by batch files,*** depending on the DOS version, wether they are "escaped" or not, and often depending on their location in the command line:

- commas (",") are replaced by spaces, unless they are part of a string in doublequotes
- semicolons (";") are replaced by spaces, unless they are part of a string in doublequotes
- "=" characters are sometimes replaced by spaces, not if they are part of a string in doublequotes
- the first forward slash ("/") is replaced by a space *only if it immediately follows the command, without a leading space*
- multiple spaces are replaced by a single space, unless they are part of a string in doublequotes
- tabs are replaced by a single space
- leading spaces before the first command line argument are ignored

I know of several occasions where these seemingly useless "features" proved very handy.
Keep in mind, though, that these "features" may vary with the operating systems used.

More on command line parsing can be found on the PATH and FOR (especially FOR's interactive examples) pages.

More options in Windows NT 4/2000/XP

Windows NT 4 introduced a set of new features for command line parameters:

`%CmdCmdLine%`	will return the entire command line *as passed to* *`CMD.EXE`*
`%*`	will return the remainder of the command line starting at the first command line argument (in Windows NT 4, `%*` also includes all leading spaces)
`%~d`*n*	will return the drive letter of %*n* (*n* can range from 0 to 9) if %*n* is a valid path or file name (no UNC)
`%~p`*n*	will return the directory of %*n* if %*n* is a valid path or file name (no UNC)
`%~n`*n*	will return the file name only of %*n* if %*n* is a valid file name
`%~x`*n*	will return the file extension only of %*n* if %*n* is a valid file name
`%~f`*n*	will return the fully qualified path of %*n* if %*n* is a valid file name or directory

Note: `%CmdCmdLine%` and `%*` will leave all delimiters intact, except, in Windows 2000 and later, leading spaces before the first argument

Windows 2000 and XP add even more options.
More information can be found at the page explaining NT's CALL command.

To remove the leading space of %* included by NT 4 use the following commands:

```
SET commandline=%*
IF NOT CMDEXTVERSION 2 SET commandline=%commandline:~1%
```

Validate command line arguments using GOTO

A tip by Oliver Schneider:

```
@ECHO OFF
GOTO:%~1 2>NUL
ECHO Invalid argument: %1
ECHO.
ECHO Usage:  %~n0  number
ECHO.
ECHO Where:  number may be 1, 2 or 3 only
GOTO:EOF
:1
:2
:3
REM Code to do something with the validated argument starts here
   .
   .
REM End of batch file
```

For a limited number of allowed arguments, this is a time saving technique.

Do note, however, that labels are case sensitive, so you may not want to use this technique for "string type" arguments.

This technique is best used when each value for %1 has its own batch code to process it:

```
@ECHO OFF
GOTO:%~1 2>NUL
ECHO Invalid argument: %1
ECHO.
ECHO Usage:  %~n0  number
ECHO.
ECHO Where:  number may be 1, 2 or 3 only
GOTO:EOF

:1
REM Preprocess value 1
  •
  •
GOTO Common

:2
REM Preprocess value 2
  •
  •
GOTO Common

:3
REM Preprocess value 3
  •
  •

:Common
REM Common processing of preprocessed values
  •
  •
REM End of batch file
```