FOR /F

Loop command: against the results of another command.

```
Syntax
      FOR /F ["options"] %%parameter IN ('command_to_process') DO command

Key
   options:
      delims=xxx   The delimiter character(s)
                   (default = a space)
      skip=n       A number of lines to skip at the beginning.
                   (default = 0)

      eol=;        Character at the start of each line to indicate a comment
                   The default is a semicolon ;

      tokens=n     The numbered items to  read from each line
                   (default = 1)

      usebackq     Use the alternate quoting style:
                   - Use double quotes for long file names in "filenameset".
                   - Use single quotes for 'Text string to process'
                   - Use back quotes for `command_to_process`

  command_to_process : The output of the 'command_to_process' is
                       passed into the FOR parameter.

  command     :  The command to carry out, including any
                 command-line parameters.

  %%parameter :  A replaceable parameter:
                 in a batch file use %%G (on the command line %G)
```

FOR /F processing of a command consists of reading the output from the command one line at a time and then breaking the line up into individual items of data or 'tokens'. The DO command is then executed with the parameter(s) set to the token(s) found.

The FOR command is the answer to innumerable questions where you want to take the output of some command, store it in a variable (%%G) then do something with the result.

For example the PING command returns serveral lines including one like:

```
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

To select that one line of output, you can search for the text "loss" (which is always present), then use the Tokens parameter to select the number of lost packets, here this is `0` but it will vary each time you run the command.

```
set _ping_cmd=ping -n 5 127.0.0.1

FOR /f "tokens=4 delims=(=" %%G IN ('%_ping_cmd% ^|find "loss"') DO echo Result is [%%G]
```

The tricky part is always splitting up the line of interest into the right tokens, in this case I'm splitting on the characters '=' and '('
these two characters split the line into 5 chunks of text and we pull out the fourth one with "tokens=4"

By default, /F breaks up the command output at each blank space, and any blank lines are skipped.
You can override this default parsing behavior by specifying the "options" parameter. The options must be contained within "quotes"

usebackq

This option is useful when dealing with a command that already contains one or more straight quotes.
The backquote character ` is just below the ESC key on most keyboards. See the FOR /F page for other effects of usebackq. Usebackq can be abbreviated to `useback` (undocumented.)

Tokens

`tokens=2,4,6` will cause the second, fourth and sixth items on each line to be processed

`tokens=2-6` will cause the second, third, fourth, fifth and sixth items on each line to be processed

`tokens=*` will cause all items on each line to be processed

`tokens=3*` will cause the 3rd and all subsequent items on each line to be processed

Each token specified will cause a corresponding parameter letter to be allocated.

If the last character in the tokens= string is an asterisk, then additional parameters are allocated for all the remaining text on the line.

The letters used for tokens are case sensitive, so you can specify up to 26 tokens (a-z) or 26 tokens (A-Z)

It is actually possible to assign up to 61 tokens by using the ASCII codes from this range: `? @ 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 [ \ ] ^ _ ` 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 {`
Note the caret character ^ must be escaped with a second caret: `^^`

FOR tokens variables (or parameter names) are global, so in complex scripts which call one FOR statement from within another FOR statement you can refer to both sets of parameters.

Delims

More than one delimiter may be specified so a string like '`abcd+efg+hijk;lmno;pqr`' can be broken up using "`delims=;+`".

You can use any character as a delimiter, but they are case sensitive.
If you don't specify delims it will default to "`delims=<tab><space>`"

n.b some text editors will enter the TAB character as a series of spaces, specifying more than one delimiter has been known to cause problems with some data sets.

eol

The default end-of-line character is a semicolon ';' when the FOR command reads a text file (or even a character string), any line that STARTS with the eol character will be ignored. In other words it is treated as a comment.
Use `eol=X` to change the eol character to X.

Often you will want to turn this feature off so that every line of your data file is processed, in theory "eol=" should turn this feature off, but in practice this fails to work correctly - it will set eol to whatever the next character is, often the quote or space character. One workaround is to set eol to some unusual character that you don't expect to ever encounter in the data file e.g. "`eol=€`" or "`eol=¬`". Another method is to escape every delimiter `For /f tokens^=*^ delims^=^ eol^= %%a in (file.txt) do...` (see forum for a discussion of this)

Examples:

To ECHO from the command line, the name of every environment variable.

```
FOR /F "delims==" %G IN ('SET') DO @Echo %G
```

The same command with usebackq:

```
FOR /F "usebackq delims==" %G IN (`SET`) DO @Echo %G
```

To put the Windows Version into an environment variable

```
@echo off
::parse the VER command
FOR /F "tokens=4*" %%G IN ('ver') DO SET _version=%%G
:: show the result
echo %_version%
```

List all the text files in a folder

```
FOR /F "tokens=*" %%G IN ('dir /b C:\docs\*.txt') DO echo %%G

FOR /F "tokens=*" %%G IN ('dir/b ^"c:\program files\*.txt^"') DO echo %%G
```

In the example above the long filename has to be surrounded in "quotes"
these quotes have to be escaped using ^
The "tokens=*" has been added to match all parts of any long filenames returned by the DIR command.

Although the above is a trivial example, being able to set %%G equal to each long filename in turn could allow much more complex processing to be done.

More examples can be found on the Syntax / Batch Files pages and the other FOR pages below.

FOR is an internal command.

"History never repeats itself, Mankind always does" - Voltaire

Related:

FOR - Summary of FOR Loop commands
FOR - Loop through a set of files in one folder
FOR /R - Loop through files (recurse subfolders)
FOR /D - Loop through several folders
FOR /L - Loop through a range of numbers
FOR /F - Loop through items in a text file
SETLOCAL - Control the visibility of variables inside a FOR loop
FORFILES - Batch process multiple files
GOTO - Direct a batch program to jump to a labelled line
IF - Conditionally perform a command
Powershell: ForEach-Object - Loop for each object in the pipeline
Equivalent bash command (Linux): for - Expand *words*, and execute *commands*