TEXcount*
Perl script for counting words in LATEX documents

ABSTRACT

TEXcount is a Perl script for counting words in LATEX documents. It recognizes most of the common macros, and has rules for which parameters to count and not to count; the main text is counted separately from the words in headers and in captions of figures and tables. Finally, it produces a colour coded version of the parsed document, either as a text document or as HTML to be viewed in a browser, indicating which parts of the document have been included in the count.

Contents

1 What does TEXcount do?

TEXcount is a Perl script made for counting the words in a LATEX document. Since LATEX documents are formated using lots of macro instructions and often contain both mathematical formulae and floating tables and figures, this is no trivial task.

Simple solutions to counting the words consists of detexing the documents, which often merely consisty of ignoring the TEX and LATEX instructions. This is a bad solution since it will usually result in over-estimating the number of words as mathematical formulae, citations, labels and references are counted.

A perfect solution, if such exists, needs to take into account how LATEX interprets each macro instruction. The simplest approach to taking this into account consisty of making the count based on the typeset document, but this too tends to over-estimate the word count as mathematical formulae, table contents and even page numbers may get counted.

A simple but robust approach, which is the one I have taken with TEXcount, is to parse the LATEX document using simple rules for how to interpret the different TEX and LATEX instructions. Rules for most of the common macro instructions are included in TEXcount, and it is possible to specify new rules in the TEX document.

The primary focus of TEXcount is to:

- provide an accurate count of the number of words in LaTeXdocuments;
- exclude or count separately document elements which are not part of the main text such as figure captions;
- enable the user to, with relative ease, check how TeXcount has parsed the document and which elements have been counted and which have not.

The last point on this list is one of the most important. Having an accurate word count is of little value unless you know that it is accurate; conversly, trusting an inaccurate word count can be potentially harmful, e.g. if you are submitting a paper or a report which has a strict word limit.

TeXcount handles complete LaTeX documents, i.e. that start with `\documentclass` and has the text between `\begin{document}` and `\end{document}`, as well as partial documents made to be included in another LaTeX document. Automatic inclusion of included documents is possible, but is by default turned off.

Since TeXcount relies on a relatively simple rules for how to handle the different macros and only performs limited checks on the validity of the LaTeX document, it is your responsibility to make sure the document actually typesets in LaTeX before running it through TeXcount. Also, TeXcount relies on handling macros taking parameters enclosed with { and }, and on ignoring options enclosed by [and]: macros with significantly different syntax such as `\vskip` cannot be handled.

1.1 What TeXcount does not do

While an ideal solution should be able to expand the macro instructions, thus being able to handle new macros, that would at it's worst require reimplementing much of TeX, something that is clearly unrealistic. While some limited methods for handling new macros based on definitions in the file might be done, I have opted for a simpler solution: to define rules stating which paramters to count and which to ignore. Thus, TeXcount cannot handle macros that may take a variable number of parameters. Nor can it handle macros that takes parameters on forms other than `{parameter}`.

In general, while TeXcount does the parsing in some detail, it does not do it exacly as TeX does it. In some respects there may therefore be critical difference: e.g. while TeX reads one character at a time, TeXcount reads one word at a time, so while LaTeX would interpret `\cite me` as `\cite{m}e`, TeXcount would interpret it like `\cite{me}`.

Another issue is that, since TeXcount does not know how to expand macros, it cannot handle macros like `\maketitle`. Instead, it will count `\title{title text}` when it occurs.

For users of languages containing letters other than A to Z, there may be an additional challenge. The script relies on Perl to recognize words as sequence of letters, and must therefore know which characters are considered to be letters. The Perl locale may be changed (as of now this is hard-coded!) to accommodate this, but this does not work for special letters encoded using TeX or LaTeX macros or codes: e.g. `\aa` and `\"a` will not be understood to be letters in the present implementation, whereas `å` and `ä` will.

2 Syntax and options

2.1 Running TeXcount

The command to run TeXcount may vary slightly depending on the operating system and the local settings.

Under Linux/Unix, it should be sufficient to run `texcount.pl` provided it is in the PATH and has been made executable (`chmod u+x texcount.pl`). The first line of the file, the one starting with `#!`, tells Linux/Unix which command to use to run the script. This may either direct to the `perl` command giving the full path to it (use `which perl` if you need to find out where `perl` is located) or, as is done in more recent versions of TeXcount, contain the line `#!/usr/bin/env perl` which should find the correct location for `perl` (provided the program `/usr/bin/env` is available).

Under Windows, running `texcount` from the command line suffices if `texcount.pl` is in the path and pl-files are defined to run as Perl scripts.

Alternatively, if the above methods do not work, you may have to execute it more exclicitly under Perl using `perl texcount.pl`. You then need to have the `perl` executable file in the path or give the explicit path.

I will simply write `texcount` in this manual for the code to execute the script. Then, the syntax becomes

```
texcount [options] [files]
```

where the options may be amongst the following:

`-v` Verbose (same as -v3)

`-v0` No details (default)

`-v1` Prints counted text, marks formulae

`-v2` Also prints ignored text

`-v3` Also includes comments and options

`-v4` Same as `-v3 -showstate`

`-showstate` Show internal states (with verbose)

`-brief` Only prints a one line summary of the counts

`-nc, -nocol` No colours (colours require ANSI)

`-inc` Include tex files included in the document

`-noinc` Do not include included tex files (default)

`-html` Output in HTML format

`-htmlcore` Only HTML body contents

`-h, -?, -help, /?` Help

`-version` Print version number

`-license` License information

If more than one file is given, TEXcount will perform the count on each of them printing the results for each, then print the total sum at the end.

2.2 Details

By selecting one of the `-v` options, you can choose how much detail is printed. This is useful for checking what TEXcount counts.

The output is colour coded with counted text coloured blue, other colours for other contexts. The colour coding is made using ANSI colour codes. These should work when printed directly to Linux xterm window, but need not work if piped through `more` or `less`: with `less` you need to use the option `-r` for the colours to be shown correcly. Under Windows or other operating systems, the ANSI colour codes cannot be expected to work, in which case the option `-nocol` (`-nc`) may be used to turn off colour coding; instead I recommend using HTML output which can be viewed in a browser.

To print the details encoded as HTML code, use the option `-html`; alternatively, with `-htmlcore` only the HTML body is printed. On Windows, I suggest using the options `-html -v` to get full detail, save this to a HTML file, e.g. using

```
texcount -html -v texfile.tex > details.html
```

and then view the details.html file in a browser.

2.3 Parsing options

If the option `-inc` is used, TEXcount will automatically count documents that are included (using `\input` or `\include`. As when giving a list of files to count, it will print the sum per file and a total sum at the end.

The default is `-noinc` indicating that included documents are not counted.

3 What TEXcount counts

The primary role is to count the words. It is not entirely clear what should be considered words, so I have had to make some decisions. A sequence of letters is certainly a word. I also count acronyms like 'e.g.', dashed words like 'over-all', and 'it's' as one word. I have decided also to count numbers as words unless they are placed in a math environment.

Mathematical formulae are not counted as words. Instead, TEXcount counts the number of inline formulae (`$maths$`) and displayed formulae separately.

Text in headers (`\title`, `\section`, etc.) are counted separately: it counts the number of headers as well as the number of words in headers.

Floating environments (or potentially floating environments) such as tables and figures are not counted as text, even if the cells of a table may containt text. However, if they have captions, these will be counted separately much like headers were. By default, begin–end environments do not modify the parsing state: i.e. environments within the text are counted as text, etc. Rules for the most common environments, at least those that require non-default treatment, should be predefined, but you may have to add more rules if you use environments defined in packages or by yourself.

Some macros are words by themselves: e.g. `\LaTeX`. These are counted as words, but you cannot expect TEXcount to count something like `\LaTeX-word` or `\{\TeX\}count` as one word, although the above explanation inicates that it should: TEXcount will in both cases evaluate the macro and the following text separately and thus count them as separate entities.

4 Macro handling rules

A very few rules are hard-coded, e.g. that text between `\documentclass` and `\begin{document}` is the preamble of the LATEX document which should be default not be included in the count. However, most of the rules consists fall into a few general categories:

macro In its simplest form, this type of rule just tells how many parameters to ignore following the macro. More generally, one may specify the number of parameters a macro takes and how each of these should be handled. Options enclosed in `[]` before, between and after parameters are also ignored; this also applies to macros not specified here, so for a macro with no rule, immediately following `[]`-options will be ignored. (This type of rule was called an exclude rule in older versions of TEXcount, the reason being that the rule originally only gave the number of parameters to ignore following a given macro.)

header Macros that are specified to be counted as headers. In fact, this only indicates that the macro should cause the number of headers to be increased by one; a rule is added to the macro-rule to count the following parameter as header text.

group For groups enclosed by `\begin{`*name*`}` and `\end{`*name*`}`, there are rules specifying how the contents should be interpreted. A macro rule is added for `begin`*name* (without the backslash!) which is TEXcount's internal representation of `\begin{`*name*`}`.

macroword This type of rule indicates that the macro itself represents one or more words. Initially, `\LaTeX` and `\TeX` are defined with values 1 indicating that each represents one word.

preamble A few macros should be counted even if they are in the preamble. In particular, `\title{title text}` is counted as a header assuming it will later be used to produce a title.

float inclusion Within floats (begin-end groups defined with parsing status -1) there may be texts that should still be counted: in particular captions. These are specified with the float inclusion rule.

file include If `-inc` is specified, included files will also be parsed and the total presented at the end. Initially, `\input` and `\include` trigger file inclusion, but more macros may be added here.

A macro parameter is normally something on the form `{something}`; more generally it may be anything TEXcount parses as a single unit (or token), e.g. a macro, but since TEXcount parses word by word rather than character by character this may not always be correct if parameters are not `{}`-enclosed or macros.

4.1 Cautions!

Since the rules are of a relatively general nature, macros that have a great deal of flexibility are hard to deal with. In particular this applies to macros with a variable number of parameters or where the handling of the parameters are not constant.

Also, `[]`-options following macros and macro parameters are always ignored, and TEXcount gives no flexibility in over-ruling that. Since options are, by definition of the term, meant to be optional, extending TEXcount to handle them would require extensive reprogramming as well as require much more detailed macro definition rules than what is now possible.

More critically, since TEXcount does not really know which macros take options or not, just assumes that options should never be included, there is some risk of misinterpreting as an option something that is not: e.g. `\bf[text]`. This is not likely to be a frequent problem. However, if something like `\bf[a lot of text]` gets ignored because it is considered an option, it can influence the word count substantially. I have therefore been somewhat restrictive with what (and how much) can go into an option.

More advanced macros are not supported and can potentially confuse TEXcount. In partcular, if you define macros that contain unbalanced begin–end groups, this will cause problems as TEXcount needs to keep track of these to know where different groups start and end.

5 Adding your own rules

Adding your own macro handling rules is relatively simple. Although editing the script is possible, and not too difficult, this has the disadvantage that the modifications will be lost if updating to a new version of TEXcount. A better and more flexible solution is to include instructions to TEXcount in the LATEX documents, alternatively to make a definition file in which new macro handling rules are defined.

Comment lines on the form

```
%TC:instruction name parameters
```

encountered in the parsed document are used to add macro handling rules. The instruction states what kind of rule, the name specifies the macro or begin-end group, and parameters specify the rule. Be aware that these are not syntax checked and may produce either Perl errors or incorrect results if entered incorrectly.

Macro names should be entered with their full name starting with backslash. Internally, begin-end groups are represented using macro names `begin`*`name`* without backslash, but rules for begin-end groups are specified through a separate TC-instruction.

Note that macro handling rules are added successively throughout the session: i.e. if more files are parsed, handling rules from previously parsed files still apply. This has advantages as well as disadvantages. If you give a list of files with the rules specified in the first file, these rules will be applied to all the documents. However, if you use the `-inc` option, included files will be parsed only after TEXcount has finished parsing the file in which they are included, so any rules specified in these will not apply to the initial document.

The instructions may be one of the following:

`macro` Takes one parameter which is either an integer or a `[]`-enclosed array of integers (e.g. `[0,1,0]`). An integer value n indicates that the n first parameters to the macro should be ignored. An array

of length n indicates that the first n parameters should be handled, and the numbers of the array specifies the parsing status (see below) with which they should be parsed. Giving the number n as parameter is equivalent to giving an array of n zeroes (`[0,...,0]`) as zero is the parsing status for ignoring text.

`header` This is specified much as the macro rule. The added effect is that the header counter is increase by 1. Note, however, that you should specify a parameter array, otherwise none of the parameters will be parsed as header text. The parser status for header text is 2, so a standard header macro that uses the first parameter as header should be given the parameter `[2]`.

`group` This specifies a begin-end group with the given name (no backslash). It takes two further parameters. The first parameter speficies the macro rule following `\begin{`*`name`*`}`. The second parameter specifies the parser status with which the contents should be parsed: e.g. 1 for text (default rule), 0 to ignore, -1 to specify a float (table, group, etc.) for which text should not be counted but captions should, 6 and 7 for inline or displated math.

`floatinclude` This may be used to specify macros which should be counted when within float groups. The handling rules are spefified as for `macro`, but like with `header` an array parameter should be provided and parameters that should be counted as text in floats should be specified by parsing status 3. Thus, a macro that takes one parameter which should be counted as float/caption text should take the parameter `[3]`.

`preambleinclude` The preamble, i.e. text between `\documentclass` and `\begin{document}`, if the document contains one, should generally not be included in the word count. However, there may be definitions, e.g. `\title{title text}`, that should still be counted. In order to be able to include these special cases, there is a preambleinclude rule in which one may speficy handling rules for macros within the preamble. Again, the rule is speficied like the `macro` rules, but since the default is to ignore text the only relevant rules to be specified require an array.

`fileinclude` By default, TEXcount does not automatically add files included in the document using `\input` or `\include`, but inclusion may be turned on by using the option `-inc`. If other macros are used to include files, these may be specifed by adding fileinclude rules for these macros. The specification takes one parameter: 1 if filetype `.tex` should be added if the file is without a filetype, 0 if it should not.

The parser status is used to dictat how each parameter should be parsed. E.g. if a macro has its parameter set defined by `[1,0,1]`, it means the first and third parameters are counted as text words (parser status 1) whereas the second is ignored (parser status 0). Another case is `\renewcommand` which is defined as `[-3,-2]`: the first parameter is to be read without interpreting the contents (which is going to be a macro name whose macro handling rules should not be applied here), and the second parameter should be ignored without requiring that begin–end groups be balanced. The different parsing states are:

0: ignore text, i.e. do not count, but will still parse the code;

1: count as text (i.e. count words);

2: count as header text;

3: count as float/caption text;

-1: float, ignore text but look for `floatinclude` macros;

-2: stronger ignore which ignore begin–end groups, e.g. to use in macro definitions where begin–end groups need not be balanced;

-3: even stronger ignore, handles macros as isolated tokens without handling their parameters, to use with macro definitions like `\newcommand` and `\def`;

-9: preamble, ignore text but look for `preambleinclude` macros.

Here are some examples together with corresponding macro definitions:

```
%TC:macroword \TeXcount 1
\newcommand\TeXcount{{\TeX}count}

%TC:macro \NB 1
\newcommand\NB[1]{\marginpar{#1}}

%TC:header \newsection [2,0]
\newcommand\newsection[2]{\section{#1}\label{sec:#2}}

%TC:group theorem 0 1
\newtheorem{theorem}{Theorem}
```

The predefined rules can easily be read off the script file: they are hash maps defined at the beginning of the script with names `TeXmacro`, `TeXheader`, etc.

6 License

The TEXcount package—script and accompanying documents—is distributed under the LATEX Project Public License (LPPL)

```
http://www.latex-project.org/lppl.txt
```

which grants you, the user, the right to use, modify and distribute the script. However, if the script is modified, you must change its name or use other technical means to avoid confusion with the original script.