MD2ME
A manual

Revised by Stephanie Björk[1] on January 9, 2018

1. Introduction

MD2ME (Markdown-to-ME) is a file converter written entirely in `sed` that converts a document formatted in Markdown into a document that can be processed by TROFF with the help of the `-me` macros. In other words, it converts Markdown syntax to TROFF syntax whilst also offering other extra features that extends Markdown. It is free and open-source software (c.f. § License).

Its original purpose was to convert simple Markdown documents that consisted of a few sections of paragraphs of text into Troff documents which drives the printer to print the page appealingly. It is an attempt to combine the convenience of *writing* prose in Markdown and the ability to *print* beautiful documents in Troff. The process of writing and printing is now separated; documents can be written virtually distraction-free without any worries about the printing process. From the original purpose, the converter has grown to cover more areas, and it is now quite capable of understanding a substantial amount of Markdown syntax.

Because MD2ME was made in `sed` , it can only parse a proper subset of standard Markdown. To make up for this, MD2ME adds and extends features from standard Markdown wherever it can by taking advantage of TROFF's flexibility and programmability, and of some of its preprocessors like eqn and tbl. Most of these are *special requests* and additional HTML-style entity characters. Consequently, standard Markdown syntax is also a proper subset of MD2ME's Markdown, and the two syntaxes will never completely match any time soon.

Fortunately, MD2ME's special requests are actually HTML-style comments in standard Markdown. So, if those special requests cannot be understood by the typical Markdown parser, then they simply do not appear on the resulting page. This guarantees that the special requests will affect the page if and only if they can be understood by a proper parser, like MD2ME. Unfortunately enough, additional HTML-style entity characters will not appear so well on a web browser.

This makes MD2ME very compatible with Markdown. Almost any Markdown document should be parsable by MD2ME and also any MD2ME document should be parsable by any standard markdown parser. In essence, MD2ME is just a

[1] Author's current address: ⟨katt16777216@gmail.com⟩

markdown parser with a few extra things.

MD2ME is **not** a Markdown-to-man converter. It does convert Markdown syntax to troff- and consequently nroff- compatible documents, but the intention is not to convert to a format suitable for UNIX manual pages. For those purposes, see [Ronn](https://rtomayko.github.io/ronn/).

MD2ME was made with the UNIX philosophy in mind. It uses the very basic utilities found in UNIX to achieve great things: document conversion. Thus, almost anyone with a terminal can use it, as `sed` and `groff` are very likely already installed amongst other things!

For succinctness, henceforth read *Markdown* as the standard Markdown formatting syntax itself and *markdown* as any program that parses standard Markdown code unadorned by MD2ME. Notice the different letter casings. To clarify, MD2ME is not *markdown* , but it understands *Markdown* .

1.1. Short introduction to TROFF

TROFF is a document formatting program along with its own document formatting language. A TROFF document is simply a text file of largely plain-text with its formatting language interspersed. The TROFF program reads a TROFF document and parses its formatting language. The language gives the program instructions on how text should look like on paper: the typeface, font style, font size, &c. The program then interprets the instructions and instructs a printer on how the printed page should look like.

The TROFF program itself knows very little about the layout of a page actually. That is, it is very easy to keep typing your document in the plain language of TROFF and have the text uncomfortably touch the margins. This is not seen as a disadvantage, but rather signifies TROFF's capability of flexibility: some day in the future, you might want to accurately recreate a 3-year-old's scribble on a page in TROFF, which obviously requires a lot of flexibility.

Because of the flexibility and programmability offered by TROFF, it allows for the development of other pre-programmed set of instructions written in the formatting language. The instructions give structure to the flexibility by defining commonly used constructs in a typical prose (e.g. paragraphs, lists, and footnotes) while also defining standard layouts (e.g. margins and automatic pagination). These aforementioned pre-programmed set of instructions are often called *macros* . Together, they make up a *macro package* . They are typically stored in a file called at **TMAC** file, dubbed from *Troff MACros* .

When writing a quintessential essay that is compliant with most style guides, it is appropriate to use TROFF **with** a macro package. Two of the good macro packages for miscellaneous writing are called `-me` and `-ms` which all result in approximately the same page layouts and prose style. However, if you do want utmost control and flexibility and probably not actually typing paragraphs, plain TROFF is the way to go.

Writing a document in TROFF with the help of a macro package and using TROFF to compile and print the document is quite like using a WYSIWYG word processor like *Microsoft Word* or *LibreOffice Writer* for that process. However, there are a few significant differences of which to be mindful:

- When used with a good macro package (like `-me`), the layout of the prose is automatic and consistent throughout the entire document. The writer is not responsible for the style of the prose. They are only responsible for the actual written content itself and properly laying out a few instructions to denote the semantic significance of parts of prose. All the styling details are abstracted to the macro package.
- In word processors like *Word* , the writer is most often responsible for keeping the styles consistent by manually adjusting spacing, sizes, and minor aspects here and there. This is okay for smaller documents, but a real nuisances for larger ones like books spanning several volumes.
- TROFF documents are simply plain text. This means that an intermediate user of TROFF can look at the TROFF source file with almost any text editor, including `ed` , read from it on paper even without access to a computer or a compiler, and even make a few adjustments. The last bit is important when you need to collaborate with other users. Since implementations like GROFF are free and open-source, it means that anyone can collaborate on a document project, regardless of how much money they have. The TROFF syntax has also been fairly consistent since 1973, so there should not be any unpleasant surprises like older formats being incompatible with newer versions of the software and vice versa.
- In word processors like *Word* , the document is likely in a proprietary format that cannot normally be read or made sense out of without a proper software on a proper computer. The only way to read it is to use Word or a Word-compatible software. Word itself is not free software, so technically, not everyone has access to it. A particular version of the document format can also be made de-facto obselete at anytime, often causing unpleasant surprises.
- There is a catch! TROFF does seem to be better than word processors, but the price to pay for all the convenience and comfort is that you have to remember necessary commands and practise them. This is one of the reasons that make document formatters like TROFF not very appealing to the general public. However, users who have gotten used to them do find word processors very inefficient, slow, and unwieldy.
- In word processors, commands hardly need to be learnt. Just type as you go! Formatting options are accessible by the click of a mouse, or a convenient shortcut combination.
- In TROFF, it is not possible to view documents live as you type character-by-character, unless you tell the computer to use TROFF to compile your documents every ½ second, which usually is not very efficient.
- In word processors, this ability to view documents live as you go is generally the main feature that is most likely taken for granted by many.

This makes TROFF in tandem with a macro package very similar to LaTeX than to any particular WYSIWYG software. Plain TROFF without a macro package is thus very similar to plain TeX.

TROFF is a document formatting system that first saw the light of day in around 1973 at Bell Labs. Back then, it was originally written by the late Joseph F. Ossanna. A few years later, it was rewritten in C and improved upon with the help of Brian W. Kernighan. Development continued slowly albeit steadily until Joseph F. Ossanna's death as a consequence of heart failure in 1979.

Thankfully, TROFF's development did not stop there for too long. In the consequent years, TROFF had undergone many significant improvements so as to be typesetter-independent[2] amongst other things, largely led by Brian W. Kernighan who has been using TROFF until today for many of his books.

Unfortunately, TROFF was proprietary and under AT&T's hands until its release not so long ago. Due to this, James Clark underwent an attempt to write a fork of TROFF in C++ in before 1990. Since 1999, Werner Lemberg and Ted Harding took over the maintainance of TROFF as one of GNU's projects, which is completely free and open-source. The fork is called GROFF.[3]

The importance of GROFF is that it has made significant improvements and added more features to classical AT&T TROFF and made TROFF widely accessible to the public domain. Along also came other free and open-source implementations, most notably Heirloom TROFF.

TROFF was derived from NROFF, which was derived from the original ROFF (run-off), which was pronounced like *rough* . Therefore, NROFF is pronounced like *en-rough* , TROFF is pronounced like *tee-rough* , and GROFF is pronounced like *gee-rough* . It is not necessary to write TROFF, NROFF, or GROFF in all capitals. Few publications do this, and would prefer *troff* , *nroff* , and *groff* instead.

It is **not** necessary to have familiarity with TROFF to use MD2ME, but it can definitely help. For more information about TROFF, see the Wikipedia page ⟨https://en.wikipedia.org/wiki/Troff⟩, [a TROFF resource site](http://troff.org/).

1.2. Short introduction to Markdown

In a sense, the idea for Markdown is quite similar to that of Troff. The writer writes her prose in a normal manner and intersperses special instructions to denote special things she needs in her document; like, * (asterisk) for italics, ** (double asterisks) for bold, # (hash tag) for sections, and ` (grave accent) for

---

[2] In the late 1900s, a typesetter was kind of analogous to a printer today. However, programs like TROFF in their early days were very much dependent on and relied upon the ability of a specific typesetter (i.e. printer) to drive the printhead print what must be printed onto paper.

[3] The dramatic suspense was not intended.

code. Markdown syntax is quite easy to type and quite obvious as to what they mean: for example, modes of emphases in Markdown look very natural for emphasis, and sections look just like sections.

Like TROFF, a Markdown document source file with interspersed instructions resides as a plaintext file. So, anyone can understand, read, and edit the document without any special or fancy software that cost large sums of money. Even without any computers, the source file can be printed as-is on paper and is still very much understandable.

Markdown is designed as a writing tool for the web. If you want to write an article, a draft, or take notes in a class, Markdown is likely for you. Unlike TROFF, the markup language for Markdown is very far from as complicated or as detailed. Markdown just needs to know what different parts of the document mean (italics, bolds, lists, section headers) and let the web browser handle the displaying of the content, which should be consistent throughout.

This means that it should be very easy to write naturally in Markdown as would anyone who can write with pen on paper.

Because MD2ME is a Markdown-to-Troff converter, it expects a certain amount of Markdown literacy from its users. Trust me on this: Markdown syntax is a million times less hellish than TROFF's and everything will most likely just click. For more information about Markdown, see Markdown Basics ⟨https://daringfireball.net/projects/markdown/basics⟩ and move on to Markdown Syntax ⟨https://daringfireball.net/projects/markdown/syntax⟩ if you have time.

2. Using this manual

This manual aims to completely and precisely describe all the Markdown syntax that MD2ME understands, their idiosyncrasies, and how they differ from standard Markdown syntax. Where necessary, it also documents any special requests and extended features provided by MD2ME that do not exist in standard Markdown, and features in standard Markdown that do not exist in MD2ME.

The `sed` script itself is already very well-commented throughout. So, taking a look at the source file may as well be a good idea if you want to understand how the parsing works.

Material in this manual assumes that you are already familiar with Markdown and TROFF. Very little to no introduction will be given for any technical points that may arise. Therefore, the manual does **not** serve as an introduction or reference manual for the standard Markdown syntax or TROFF syntax. If you need some familiarity or a reference manual for those two, see the introductory subsections above this section.

For a table of contents which enumerates all sections in this document, refer to the last page.

This manual was written by Stephanie Björk on January 9, 2018. It is licensed under Creative Commons BY-SA 4.0. Make sure you understand your rights and

limitations as stated by [the license](https://creativecommons.org/licenses/by-sa/4.0/).

The entirety of this document was written in Markdown with some special MD2ME syntax interspersed here and there. If the document you are reading has been rendered using TROFF or a compatible compiler like Groff or Heirloom Troff through MD2ME, it stands as an example of MD2ME's conversion capabilities and also as a showcase for the special requests it provides.

For the original Markdown document, see ⟨https://google.com/ncr⟩

3. Usage

MD2ME is a sed script that adheres to the UNIX philosophy. All input and output data are in plaintext format and relies on the concept of the standard input and standard output, programs can be glued together to form metaphorical pipelines with valves and mid-stream processors, and all data that must be prolonged are stored as plaintext in files on disk.

Assuming MD2ME is in one of the paths in `$PATH` and its basename has not been changed, executing MD2ME can be as simple as typing `md2me` on the shell and pressing ENTER. Without a list of files provided as arguments to be processed, md2me resorts to processing the standard input. Options to md2me are the same options to `sed`. Like `sed`, md2me operates in a line-oriented manner, which actually poses a few limitations as we shall see further.

Therefore, the synopsis for MD2ME goes something like this:

```
md2me [options]... [file]...
```

where `[file]...` is a list of one or more files separated by spaces.

3.1. Examples

3.1.1. Generating prints

MD2ME is very likely used for generating PDF or Postscript files with structured layout ready for priting out of Markdown documents. This can be done through a TROFF compiler called GROFF and specifying the output device (format) to PDF or PS. The following is an example that will generate PDF output to `article.pdf` from a Markdown file `article.md`, which may contain UTF-8 characters like *åäö*, equations, and tables within the prose.

```
md2me article.md | groff -Tpdf -Kutf8 -e -t > article.pdf
```

First, md2me converts Markdown syntax into TROFF syntax. The TROFF syntax is then passed over to groff which converts the human-readable TROFF instructions into PDF instructions which describe how text should end up on a page. Finally, the PDF output is redirected out of standard output into a file called `article.pdf`, which should be a valid PDF file if nothing goes wrong somewhere in the pipe line.

To groff, the option `-t` tells groff to preprocess the document and look for tables set in tbl, the option `-e` tells groff to preprocess the document and look for equations set in eqn, the `-Kutf8` tells groff to convert UTF-8 byte sequences into equivalent character entities that groff can understand, and `-Tpdf` tells groff to postprocess its machine-friendly typesetter-independent instructions using a PDF maker of some sorts called `gropdf`. Thus, the intellectual may use the following command, which achieves the same thing as the one before.

```
md2me article.md | groff -Z -Kutf8 -e -t | gropdf > article.pdf
```

Likewise, to have the output be Postscript, the following commands do the same thing as the examples above. The only difference is that the output is now in Postscript.

```
$ md2me article.md | groff -Tps -Kutf8 -e -t > article.ps
$ md2me article.md | groff -Z -Kutf8 -e -t | grops > article.ps
```

The latter variant is reserved for the very smartest people. By default, groff does the postprocessing from a typesetter-independent format into Postscript. So, there is no need to specify the option `-Tps`, as the output will be in PS anyway. Thus, the following command will do the same thing.

```
md2me article.md | groff -Kutf8 -e -t > article.ps
```

It is possible to omit the `-e` option if your document does not have equations (very likely), the `-t` option if your document does not have tables (likely), and the `-Kutf8` option if you don't write anything beyond ASCII (unlikely). Expect groff to run faster by a few milliseconds as the following command:

```
md2me article.md | groff -Kutf8 -e -t > article.ps
```

is internally run like this:

```
md2me article.md | tbl | eqn | preconv -eutf8 | troff | grops > article.ps
```

3.1.2. Getting a print preview

Another practical example is using zathura to read the Postscript output from groff directly from the Standard Output. This is a good way to preview your document before printing. The following example illustrates the command to be used. This assumes that Zathura is installed with a Postscript interpreter like `gs`.

```
md2me article.md | groff -Kutf8 | zathura -
```

For more information on options and arguments to md2me, groff, and zathura, see their respective pages on the section 1 of the manual pages.

3.1.3. Inspecting the TROFF source

The output from md2me is just TROFF, but there is no guarantee that it is very appealing to read. Nonetheless, it can be useful for those who want to inspect the TROFF source code to check for quality or to satisfy their curiosity. Though, it

is more often the case that the output TROFF source file may occasionally need to be edited to cater for something that MD2ME cannot do. The following command pipes the Troff output to a file called `article.tr` .

```
md2me article.md > article.tr
```

The author does not mind close examination of the generated TROFF output. Just do it for your own sake.

4. Initialization

4.1. TROFF document header

At the beginning of any input file read, whether it is a real file or the standard input, MD2ME will insert approximately 80 lines of plain GROFF instructions and comments. These comments give general information about what piece of software generated the TROFF file; the comments are obvious and also documents almost every line of instruction quite well, so read them if you want to. The instructions are a little bit more complex and all play a part in making the MD2ME possible.

`sed` cannot do everything, so some logic must be outsourced to TROFF. Nonetheless, the logic, whether it be processed by `sed` or TROFF, they still pertain to typesetting and getting text laid out on a page. The following list vaguely details what the instructions do listed in the order that they appear on the output file and are thus executed by TROFF.

(1) Is the compiler Heirloom TROFF or GNU TROFF? If it is, proceed. If not, fail with a message telling the user that their compiler is not compatible. The instruction lines following this and the typographic features required by MD2ME are simply not catered for by the classical AT&T TROFF.

(2) Are the `-me` macros already loaded, perhaps as a command-line option to groff earlier? If they have not been loaded, then automatically load it now. It's best practice not to load the macros first anyway for convenience and flexibility during the header of the document. This basically checks for the existence of the `.lp` macro which gives left-justified paragraphs.

(3) The format for the 24-hour clock is set with appropriate 0-paddings.

(4) Some strings are defined for [ ] (a pair of square brackets), ... (horizontal ellipses), links ⟨#⟩, the TEX logo, and the LATEX logo. They shall be used throughout the document wheresoever necessary.

(5) Some macro definitions are defined that toggle *italics* , **bold** , and ***bold italics*** fonts. A macro is defined that toggles the `monospace` typeface, and one is defined that collects all information about section headers, their numbers, and the number of the page in which they occur. Two blank macros, `.XS` and `.XE` , are also defined as placeholders for

verbatim blocks of TROFF code. They will get interspersed within the output wheresoever necessary. A macro also gets extra instructions appended to it so as to allow blockquotes to be nested.

(6) The typeface, font numbers, and standard point sizes are defined. The typeface Helvetica is defined to be the active and default typeface. Its fonts, roman, *italics* , **bold** , and ***bold italics*** , are mounted on slots, 1, *2* , **3** , and ***4*** respectively. The `monospaced` font is mounted on slot `6` for convenience. Paragraphs shall be set in 12 points, titles (headers and footers) in 12 points, block quotations in 10 points, section headings in 14 points, and footnotes in 10 points. Any non-paragraph text other than those shall be set in 12 points.

(7) Colours are enabled. Some colours are defined: black (bk), blue (bl). All text shall be set in black. Links are set in blue.

4.2. URLs for reference-style links

URLs for reference-style links like `[link][ref]` are collected when they are defined into a file called `reflinks.md2me` . This means that the `sed` script will need write access to the current working directory. If there are reference-style links in the document, they will be collected into that file. If such is the case, after the `sed` script exits, you must rename the file to `reflinks.md2me.tr` on the same directory, then rerun the sed script again on the same input file. After the initialization sequence detailed above, the contents of `reflinks.md2me.tr` will be placed there. Only then will URLs for reference style links show up properly.

The file `reflinks.md2me.tr` simply contains string definitions for every link whose URL has thus far been collected. Note that the file will be placed **after** the first line. Therefore, the first input line may **not** contain reference-style links but the 2nd and subsequent lines can, as the string definitions do not get loaded before the first line is processed, but right after it.

Two passes are necessary as `sed` traverses through the lines of the file in a single-pass mode. It cannot go in reverse, only forwards is allowed. Since the actual URLs for reference-style links *may* very well be defined after its associated reference-style links occur, this becomes quite a problem. This is analogous to why LaTeX ToCs need to be generated by running the LaTeX compiler twice.

For grace and simplicity, avoid reference-style links if at all possible.

For more information about using reference-style links, see the relevant section hereafter.

5. Token precedence

Not all lines, or tokens are created or interpreted equally. Some of them are processed with higher precedence than the others, though the level of precedences are always consistent throughout every input line read by MD2ME. The following

list enumerates each token in the order that they are processed by the `sed` script, starting from higher precedences to lower ones. Operands to *and* and *or* conjunctions used in the list are written to reflect on the precedence at which they are processed.

(1) HTML block elements
(2) HTML span elements
(3) HTML comments that are *not* special requests to MD2ME
(4) HTML doctypes
(5) MD2ME requests that stay on *one* line
(6) MD2ME footnote block entries
(7) MD2ME TROFF preprocessor blocks and TROFF-copy blocks
(8) MD2ME requests that do not exist
(9) Section headers
(10) Paragraphs
(11) Backslash escapes
(12) Implicit URLs and emails to be made as links
(13) HTML tags that cannot be parsed
(14) Horizontal rules
(15) Unordered and ordered lists
(16) Hard line-breaks
(17) Inline code and modes of emphases
(18) Capitalized names and acronyms whose point size need to be reduced
(19) English ordinal numbers with correct grammar
(20) Footnote numbers which need to be interpolated
(21) Markdown-style images
(22) Reference-style and inline-style links
(23) Sentences separated by a full stop, an exclamation mark, or a question mark; and followed by 3 spaces. Extra trailing spaces at the start or end of line that need to be removed.
(24) Proper names and acronyms which require special typographic features
(25) HTML and MD2ME-additional character entities

Assuming two tokens x and y in any string, if x has a greater precedence (value) than y, then y has a greater anti-precedence (value) than x. In this case, within any given string, x is processed before y is processed. A list of precendence is the inverse over the set of natural numbers of the anti-precendence list; likewise, a list of anti-precedence is the inverse over the set of natural numbers of the precedence list. In other words, the list above is an anti-precedence list, as

increasing values denote decreasing precedence. If the numberings in list above is in reverse order, then the resultant list is a precedence list, as decreasing values denote increasing precedence.

From the list above, a few of many conclusions can be derived:

- It is possible to have links (anti-precedence 22) within section headers (anti-precedence 9). The links will be set in the same font and size as regular section header text. This is because 22 > 9, and following from the axiom in the previous paragraph, the token with lower anti-precedence shall be processed first, then the token with the higher anti-precedence is processed later. Thus, the section header environment is already prepared *before* list items are.
- It is possible to have capitalized names and acronyms whose sizes shall be momentarily reduced (a.p. 18) within section headers (a.p. 9) and paragraphs (a.p. 10), entailing the same axiom.
- It is possible to have inline code and emphases (a.p. 17) within section headers (a.p. 9), entailing the same axiom.

Quod erat demonstrandum. However, the last conclusion is *not* possible in practice due to TROFF's limitations.

There is no need to memorise or practise these precedences. They are put in such a way that it should be natural for a Markdown writer.

6. Syntax

In the subsections to follow, Markdown syntax implemented by MD2ME and MD2ME-specific syntax will be described. The subsections are ordered by the anti-precedence values from the list in the previous section.

6.1. HTML elements

Only a very small proper subset of standard HTML elements are understood by MD2ME. These include some block-level elements and span-level elements.

6.1.1. Block-level elements

Markdown-style block quotations and block codes are **not** supported. The following HTML alternatives shall be used instead. The reason for this is because of how complicated it is to parse those constructs.

6.1.1.1. Block quotations

`<blockquote>` elements can be used to enclose a quotation. They can be nested. MD2ME does **not** support the email-style blockquotes that markdown supports comfortably. So, use this HTML element for any blockquotes and any other markdown parser will happily parse it too!

The `<blockquote>` tags themselves can be indented from the leftmost column by spaces.

Blockquotes in MD2ME differ from standard Markdown in that all syntax, including italics and bolds, still get processed and rendered within a blockquote.

Here is an example of a blockquote:

> Selection sort iterates through an array of size *n* once. Each iteration introduces another iteration over the subarray whose size is *n - k - 1* for any given *k* th iteration. In other words, each main iteration comes with a smaller iteration that shrinks as a function of the number of main iterations. Therefore, the total number of subiterations is…

It was typed like this:

```
<blockquote>
Selection sort iterates through an array of size *n* once.  Each iteration
introduces another iteration over the subarray whose size is *n - k - 1* for any
given *k*th iteration.  In other words, each main iteration comes with a smaller
iteration that shrinks as a function of the number of main iterations.
Therefore, the total number of subiterations is&hellip;
</blockquote>
```

6.1.1.2. Code blocks

`<code>` elements can be used to enclose a block of code. There really is no need to nest `<code>` elements, and doing so will only cause confusion to MD2ME: **never** nest them. MD2ME does not support Markdown's 4-space-/tab-indented code blocks. So, use this HTML element instead, and any markdown parser will happily parse it too!

`<code>` tags themselves cannot be indented. Nonetheless, they should not be indented even if they could be, as it causes a lot of confusion in prose.

Within code blocks, no tokens or lines are processed. The only processing procedures done are:

- Backslashes are automatically escaped wherever they occur.
- Dots (.) and apostrophes (') on the first column of a line get escaped to avoid misinterpretation by TROFF.
- A line with just `<code>` starts another code block. To get a literal `<code>` , type `&Mnn;<code>` .
- A line with just `</code>` will end the entire code block. To get a literal `</code>` , type `&Mnn;</code>` .
- The following character entities will be replaced with their characters:

  `&Mnn;` with nothing, `<` with <, and `>` with >.

It is best to use `<` and `>` for angle brackets when talking about HTML code. This is not necessary as HTML tags are normally not understood by TROFF anyway. It is quite a hassle, but it is just to be compatible with other markdown processors.

To be able to fit 80-column lines of code onto an A4 page, even if it hits the margin, code blocks are set 1 point size smaller than normal text and inline code spans. They are also set in no-fill mode, i.e. all newlines are displayed as newlines.

Here is an example of some code:

```
.\" Prints integers from 1 to 10, inclusive.
.nr i 0 1
.while \n+i<=10 \{ \
.       nop \ni
.\}
```

It was typed like this:

```
<pre>
<code>
.\" Prints integers from 1 to 10, inclusive.
.nr i 0 1
.while \n+i<=10 \{ \
.       nop \ni
.\}
</code>
</pre>
```

The inclusion of the `<pre>` tags is not necessary. They are simply there to be compatible with other markdowns so that web browsers know to set code in no-fill mode. In MD2ME, `<pre>` tags are deleted.

6.1.1.3. Comments

Single-line comments reside on a line of their own. They start the line with `<!--` and end it with `-->` . These comments will not appear on the final output, but will be copied into TROFF, again as comments.

For example, a single-line comment may look like this:

```
<!-- Sir, yes, sir! -->
```

Large block comments spanning many lines start with `<!--` on a single line of its own and end with `-->` on a single line of its own. Again the contents of the comments will not appear on the final output, but will be copied into TROFF, again as comments.

For example, as block comment may look like this:

```
<!--
It’s a piece of cake to bake a pretty cake.
If they way is hazy,
you gotta do the cooking by the book.
You know you can’t be lazy!
```

-->

Naturally, contents within the comments are not processed by MD2ME, unless they are a closing --> . If a line begins with a series of 2 or more dots, then that line is escaped, so that it does not accidentally close the comment.

Be mindful of one very important detail. Auto-increment number registers will be affected within a block comment. If you do not know what this means, you can ignore this paragraph.

DOCTYPES, or lines that start with <!DOCTYPE are treated somewhat like comments regardless of what line it lies in the file. Unlike comments, they are not copied into the TROFF output, but rather deleted instead.

Be careful with comments that start with an ! (exclamation mark), i.e. <!-- ! . These are **not** comments. They are not processed as comments, but rather as MD2ME's special requests. If you want a literal asterisk at the start of a comment, escape them using a backslash (\e) or &Mnn; , whichever you are comfortable with.

6.1.2. Span-level elements

Only a few span-level elements are processed. If the span-level element cannot be processed, it is left as-is and passed to the final output.

6.1.2.1. Superscripts and subscripts

Superscripts and subscripts are done using <sup> and <sub> elements anywhere within a line.

Thus, on the same line of input, I can say ax^2 and x_i.

This is what was typed:

```
Thus, on the same line of input, I can say ax<sup>2</sup> and x<sub>i</sub>.
```

6.1.2.2. Short quotes

Short “quotes” anywhere on a line can be set using " elements.

The paragraph above was typed like this:

```
Short <q>quotes</q> on a line can be set using
`<q>` elements.
```

It is generally recommended that you use the entities “ and ” for double quotes, and ‘ and ’ for single quotes instead.

6.1.2.3. Underlines

To get an underlined piece of text, use the <u> element.

So, I can say Hello there!

Note that an underlined block of text may not span to more than one line, and there should only be one underlined block of text for each line. Otherwise, prepare for a mess.

6.2. MD2ME special requests

Special requests by MD2ME are additional features only available on MD2ME and are not implemented by Markdown. A Markdown document with these special requests should still be compatible with markdown parser out there. Therefore, if the markdown parser cannot parse the requests, they simply will not appear on a web browser when the document is parsed using markdown. If the markdown parser like MD2ME can, then there shall be some special meaning associated to it.

For example, the following equation was set using eqn — an equation setter for TROFF. The instructions to eqn are sent directly to TROFF through the Markdown document, by using MD2ME's special requests. If you cannot see the equation, your markdown parser simply is not capable of processing the copy request; no errors or warnings are generated either.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

This means that documents in MD2ME are fully compatible with regular markdown parsers. The heart of this trick lies in the fact that the equation above was typed like this:

```
<!-- !eq
x = {-b +- sqrt {b sup 2 - 4ac}} over 2a
!eq -->
```

Yes! Special requests for MD2ME are simply comments in standard Markdown. If the parser sees it as a comment, it simply gets ignored and removed. If the parser sees it as an equation, it gets set on the final output.

This subsection lists all special requests by MD2ME along with their description. Please note that the requests' names are quite terse. If you are afraid you cannot remember them, create an account (if you haven't already) on Memrise ⟨https://memrise.com⟩, create your own course, and add into the course requests and their definitions that you struggle to remember.

6.2.1. Single-line requests

These requests only take up one line.

6.2.1.1. Page headers and footers

Within a page's margins, there can be header and footer texts. These are defined by the following requests. The headers are defined as 3-part titles. So, in a header string like `'foo'bar'lol'` , the left part of the margin takes takes

'foo', the centre takes 'bar', and the right takes 'lol'. The titles are printed without the quotes.

- <!-- !he 'foo'bar'lol' --> defines the page header of **all** pages to be 'foo' on the left, 'bar' in the centre, and 'lol' on the right. All requests for page titles take the same type of 3-part titles; the 3-part titles will henceforth be omitted.
- <!-- !fo --> defines the page footer of **all** pages.
- <!-- !oh --> defines the page header of only **odd** pages, i.e. pages whose page number is *not* divisible by 2.
- <!-- !of --> defines the page footer of only **odd** pages, i.e. pages whose page number is *not* divisble by 2.
- <!-- !eh --> defines the page header of only **even** pages, i.e. pages whose page number is divisible by 2.
- <!-- !ef --> defines the page footer of only **even** pages, i.e. pages whose page number is divisble by 2.

6.2.1.2. Page margins

The margins of a page can be set using the following commands where necessary. Values given here are **not** defaults. They are simply examples, and they should be replaced by whatever your use case demands.

- <!-- !po 1i --> defines the page offset, or the amount of spacing from the left most of the paper to the first character of a left-justified paragraph, to 1 inch.
- <!-- !pl 11i --> defines the page length, or the amount of vertical space that text can span before a pagination (page break) is forced upon it, to 11 inches. Try not to use this one in particular; it's a little bit complicated.
- <!-- !ll 8i --> defines the line length, or the amount of horizontal space that text can span before the line is broken, to 8 inches.
- <!-- !lt 8i --> defines the length of title (headers and footers) to 8 inches. If this value is smaller than the amount occupied by the titles, the titles may overlap.
- <!-- !m1 0.5i --> defines the vertical spacing between the top of the paper to the top of the page's header line, to ½ inch.
- <!-- !m2 0.5i --> defines the vertical spacing between the bottom of the page's header line to the first line of actual content, to ½ inch.
- <!-- !m3 0.5i --> defines the vertical spacing between the bottom of the last line of actual content to the top of the page's footer line, to ½ inch.
- <!-- !m4 0.5i --> defines the vertical spacing between the bottom of the page's footer line to the bottom of the paper, to ½ inch.
- <!-- !mm 0.5i 0.25i 0.25i 0.5i --> (see next paragraph).

There is also a construct like <!-- !mm 0.5i 0.25i 0.25i 0.5i --> , which is equivalent to:

```
<!-- !m1 0.5i -->
```

```
<!-- !m2 0.25i -->
<!-- !m3 0.25i -->
<!-- !m4 0.5i -->
```

i (inches) is just one of the many scaling factors provided. These are the same scaling factors as the ones provided by TROFF. There are also c for centimetres, v for one line-space, m for ems, n for ens, and many more. For more information, see TROFF's user manual.

6.2.1.3. Chapters and Parts

Although Markdown is designed with writing for articles for the web in mind, MD2ME provides the facility to subdivide a Markdown document into several chapters and parts (Abstract, Preliminaries, Main Content, Bibliography, Appendices). This means that a Markdown document can easily be a small book, with the help of MD2ME.

The following requests pertain to this:

- <!-- !ch Introductory Mathematics --> creates a chapter called "Introductory Mathematics." There is also an additional amount of vertical space from the top of the page. This amount of space is the same amount used by PhD theses at UC Berkelly.
- <!-- !ct x --> defines the current portion of the book to x . x can be replaced with: AB for the Abstract, P for the preliminaries, C for the main content, B for the bibliography, and A for appendicies. This information is used to give the right page numbering and chapter style. For example, page numbers for preliminaries will be in roman numerals, and chapter titles for appendices will be titled like **APPENDIX A** .

6.2.1.4. MD2ME-style sections

Instead of using Markdown-style sections with one or more hashtags to indicate levels, MD2ME-specific sections can be used, but are *not* compatible with standard Markdown.

The following requests pertain to this:

- <!-- !sh n Hello World --> creates a section titled Hello World on level n . The levels can be from 1 to 6, inclusive.
- <!-- !uh Final thoughts --> creates an unnumbered section on the first depth called Final thoughts .

6.2.1.5. Section indentation as a function of section depth

To adjust the amount of indentation per section depth, the <!-- si --> request can be used.

For example, <!-- si 2n --> sets section indents to 2 ens per section depth. So, on the first section depth, the section's text will be indented by 2 ens from the left margin; on the second section depth, the subsection's text will be indented by

4 ens from the margin, and henceforth.

All scaling factors provided by TROFF apply.

6.2.1.6. Changing point sizes

The point size can be adjusted momentarily using the request `<!-- sz -->` .

For example, `<!-- sz 16 -->` makes any text **right after** it set in 16 points. The point size is then reset to the defaults upon the next paragraph or block-quote.

The default sizes are defined in the first few lines at the start of the TROFF output file. To change them, either change the output file or use a TROFF-copy block. The reason for not giving much control on point sizes is because there are very many constructs whose point sizes can be customized, and it is generally better to use just “plain” TROFF for it.

6.2.1.7. Delimiters for inline EQN equations

EQN allows for equations or mathematical expressions to appear within a line of text. This paragraph, for example, contains an inline equation: $\lim_{x \to 0} \frac{1}{x} = \infty$.

Unlike block-level equations seen in the introductory part of the section on special requests, inline equations will appear as cacophonous jargon as paragraph text if the Markdown document is parsed by a typical markdown processor. Therefore, inline equations are **not** compatible with Markdown. So, it is recommended only to use them if your intention is only to use MD2ME.

Most writers who use TROFF use the $ (dollar sign) to delimit inline equations, so eqn can differentiate between text and mathematics. Another popular alternative delimiter is the @ (commercial at). By default, the delimiter for EQN is **not** set, thus in-line equations are not initially possible. To set it, `<!-- !ed -->` request can be used.

To use dollar signs ($) as eqn delimiters, do this:

```
<!-- !ed $$ -->
```

A few points are to note. Firstly, it is not possible to use HTML or MD2ME character entities to represent delimiter characters, for those will not be processed. Secondly, the delimiter character has to be typed twice, so it is `<!-- !ed $$ -->` , not `<!-- !ed $ -->` . Thirdly, if a character is used as a delimiter, that same character cannot be used as anything else: not even within code blocks; otherwise, the delimiter character may get interpreted wrongly as an equation.

To turn off inline equations and restore the delimiter for normal usage, the following request will do:

```
<!-- !ed off -->
```

Quick tip: If you use the dollar sign as a delimiter for EQN and you want to use the dollar sign to represent currency or something else again, you must use MD2ME's dollar sign character entity: `&Mdo;` . These entities will be able to escape the grasps of EQN and avoid misinterpretation.

Therefore, it is possible to say: Michael and Jack both share \$ $\left(\frac{\sqrt{2}x}{\log 5} + \frac{5y}{6} \right)$, with the delimiters on as dollar signs.

6.2.1.8. MD2ME-style images

Images can be included into the document using a TROFF macro called `PSPIC` . To avoid having to use plain TROFF just to call PSPIC, the `<!-- !ps -->` request exists. All arguments to the request are directly passed to PISPIC without any further processing done to the line.

The only image type supported is `.eps` or *encapsulated postscript* . Therefore, not only is the request not interpreted by Markdown, the image type is not generally supported by most web browsers either. For this reason, you should only go through the hassle of converting your JPG or PNG files to EPS if you are certain your document is only going to be used with MD2ME. Tools to convert include ImageMagick.

Thus, to include a picture from `selfie.eps` , you can type:

```
<!-- !ps selfie.eps -->
```

Since arguments to the request are the same as arguments to the `.PSPIC` macro, the line above is equivalent to:

```
<!-- !tr .PSPIC selfie.eps -->
```

The size, alignment, and indentation of images can optionally be specified as options to `.PSPIC` . For more information, consult `groff_tmac` on section 7 of the manual pages.

6.2.1.9. Keeps

For important or long parts of prose or figures, it may be necessary to make sure they are able to utilise the most amount of paper as possible with little to no interruptions in flow. This is where the concept of *keeps* come in.

For any text within a *keep* , if the text reaches the end of the page (too long to fit current page), then that text gets moved over in one whole lot to the next page. However, if the text does not reach the end of the page (short enough to be in one page), then it is left untouched.

There are two types of keeps and they all try to achieve the same purpose, but with different behaviour.

- A **block keep** is a simple but selfish beast. If the text it is guarding is too long, it simply moves itself and its precious text to the next page and leave a

potentially large blank space bereft of content behind its wake. Otherwise, it does nothing and stays where it was put.
- A **floating keep** is a more complicated but thoughtful soul. If the text it is guarding is too long, it moves itself and its precious text to the next page, but also pulls in the other content that came after it. Otherwise, it moves itself and its text and resides on the bottom of the page.

In other words, with a floating keep, any holes are filled and text can flow without interruptions.

In MD2ME, keeps should **never** be nested.

6.2.1.9.1. Block keeps

Block keeps are started and ended using two requests respectively:

```
<!-- !bs -->
<!-- !be -->
```

The requests take no arguments. Any normal text in between those two lines are protected in the keep, obviously. `<!-- !be -->` must come before EOF.

6.2.1.9.2. Floating keeps

Floating keeps are started and ended using two requests respectively:

```
<!-- !zs -->
<!-- !ze -->
```

The requests take no arguments. Any normal text in between those two lines are protected in the keep, obviously. `<!-- !ze -->` must come before EOF.

6.2.1.10. Table of Contents

Each occurrence of a section, whether in MD2ME or Markdown style, will have their numbers, names, and the page numbers in which they occur automatically recorded as entries in an accumulative buffer. Two buffers are used:

- `sh` for numbered sections of depth 1–6, inclusive;
- `uh` for unnumbered sections.

To print out the buffer(s) of your choice, use the request `<!-- xp -->` which takes one argument: the buffer to dump.

For example: to dump the `sh` buffer and list out all the numbered sections and the pages in which they occur, type:

```
<!-- !xp sh -->
```

For an example of this, see the end of this manual.

6.2.1.11. n- column processing

Text can be set in 1-, 2-, or more columns. That is, text on a page can be subdivided into many columns on the same page. This can help save space for certain types of prose, or it could just be a style choice. This is an example of 2-column processing.

MD2ME offers facilities to display text in multiple columns using the following requests:

- <!-- !2c --> . Unadorned, this request switches to 2-column processing. Arguments to the request specify the amount of spacing between columns (TROFF units apply) and how many columns are to be set. Two is cramped enough, actually. So, <!-- !2c 5n 3 --> sets subdivides a page into 3 columns separated by 5 ens of horizontal space.
- <!-- !1c --> . Takes no arguments. Simply turns off multi column processing.

6.2.1.12. Pagination

A new page or column break can be called for with the following requests respectively:

- <!-- !bp --> . Unadorned, this request simply starts a new page. All the footers are printed, nonetheless. Optionally, an argument may specify the number of the next page: you might be on page 21 (20) now and you want the next page to be page 64; so, you type <!-- !bp 64 --> .
- <!-- bc --> begins a new column in multi-column mode. If not in multi-column mode or there are no columns to traverse on the same page, a new page is started.

6.2.1.13. Mystery

<!-- FR --> . Can you guess what it does? ;)

6.2.1.14. TROFF-copy line

To copy one line directly into TROFF without any processing. Simply put the TROFF code with the <!-- !tr --> request in juxtaposition, like this:

```
<!-- !tr .sp 0.5i -->
```

That line directly instructs TROFF to yield ½ inch of space.

No further processing is done within the line, so asterisk and backslashes can be used freely without any misinterpretations. It also means that HTML and MD2ME entities are not supported. Thus, the following line is possible.

```
<!-- !tr .ds s "\*(wk \*(td -->
```

6.2.1.15. Miscellaneous

The following requests are truly miscellaneous and may or may not be removed some point in the future.

- `<!-- !mx -->` causes MD2ME to abort all text processing and quit with an exit status of 0. Anything beyond the request is not processed or output at all.
- `<!-- !ln -->` causes MD2ME to dump the current number of input lines thus far processed. The number also takes into account the current line with that request.
- `<!-- !ex -->` causes the TROFF compiler to abort right at that point. More like a trap to annoy people who want to compile your documents.

6.2.2. Block-level requests

These are requests that can span many, many lines long. There are not many and are all comments in standard Markdown.

6.2.2.1. Footnote entries

For any given footnote, an entry that corresponds to that footnote can be added. The entry will be put at the end of the page on which the footnote entry is entred. A footnote entry starts with `<!-- !fn` on one line and ends with `!fn -->` on another line of its own. Thus, to get footnotes like this[4], type this:

```
<!-- !fn
This is a fun footnote entry!
!fn -->
```

The superscripted number for footnotes is made by doing surrounding any cardinal number or one hashtag with two enclosing square brackets, like `[123]` or `[#]` . These will get replaced as the current footnote number superscripted. For them to be set in superscript, these footnotes need to be on a line that does not have any inline code blocks. The numbers within the brackets do not actually matter. Footnote numberings are kept by the `-me` macros and cannot be changed.

A footnote entry is not interpreted as Markdown. Rather, it is interpreted as plain TROFF. Therefore, using an HTML entity or Markdown-style emphases will not work. Use TROFF's requests instead.

6.2.2.2. Preprocessor support

There exist a handful requests that properly intersperse and copy instructions to a TROFF preprocessor. Like footnote entries, the start and end must be on a line of their own, and the content is encompassed therewithin. The requests are as

[4] This is a fun footnote entry!

follows.

- <!-- !eq , !eq --> for equations to eqn.
- <!-- !tb , !tb --> for tables to tbl.
- <!-- !pc , !pc --> for diagrams to pic.
- <!-- !xx , !xx --> for literal troff code to be copied directly to troff.

There is no such preprocessor as xx , actually. This is simply a delimiter to mark the boundary of Markdown and TROFF.

Unlike footnote entries, the start and end requests can take arguments. The arguments get parsed directly to the preprocessor's start/end requests. Therefore, it is possible to get a centred, continued, and number equation like this:

$$\begin{aligned} \int_{0}^{\pi} \sin(x)\, dx &= \cos(0) - \cos(\pi) \\ &= 2 \end{aligned} \tag{1.1}$$

by typing this:

```
<!-- !eq C (1.1)
int from 0 to pi sin (x) ~ dx mark = cos (0) - cos ( pi )
!eq C -->
<!-- !eq
lineup = 2
!eq -->
```

A preprocessor entry is not interpreted as Markdown. It simply gets passed TROFF as if it were a copy block. Therefore, it is not possible to use HTML- or MD2ME-style character entities or Markdown-style annotations, since this is when you enter the world of plain TROFF.

For more information about each preprocessor, see their respective manuals. It may also be required to add extra options to groff during compile time so as to make sure the lines meant for preprocessors get preprocessed.

6.2.3. Catch all

If a command to a preprocessor does not exist, it is still not treated as a comment, but rather removed.

So, typing this obviously invalid request:

```
<!-- !qwerty -->
```

will result in that line being removed.

6.3. General syntax

This section details syntax mostly from Markdown, but also a few other points that make MD2ME different will also be detailed. There are also a few syntax that are specific to MD2ME; those will be described as well.

6.3.1. Section headers

MD2ME supports Markdown's atx-style headers of levels from 1 through 6 inclusively, but **not** settext-style headers. The reason for this is because settext-style headers are rather difficult to parse. It also supports unnumbered sections as well.

Note that numbered sections are intrinsic to standard Markdown and will get parsed properly, but unnumbered sections are a special feature provided by MD2ME.

A paragraph after a section should not be separated by a single back line, because upon spawning a new section, a paragraph mark is already created. Thus, it is better to type this:

```
# First section
First paragraph
```

than this:

```
# First section

First paragraph
```

Numbered sections

The number of hashtags for atx-style headers has a positive linear correlation to the depth of the section headers in the TROFF output. Section headers from 1 through to 6 inclusively are supported. So, to get a section header on the 6th depth, type:

```
###### Challenger deep
```

The section numbers are kept and incremented automatically in lexicographical order by TROFF.

This manual is an example that extensively uses numbered sections.

Unnumbered sections

For unnumbered section headings that do not interfere with section numbers, use 7 hash tags. So, the following will result in an unnumbered section:

```
####### Earth's core
```

This section is an example that contains two unnumbered sections.

6.3.2. Paragraphs

Paragraphs are denoted with one blank line before the paragraph text, just like in Markdown. Paragraphs are seldom confused for other typographical elements like lists and MD2ME's special requests that must start a new line.

Paragraphs are normally glued together, but in quite a weird way. Like Markdown, there could be 20 blank lines but there will just be one paragraph. Unlike Markdown, paragraphs are glued together correctly only 50% of the time.

This is the situation in MD2ME. Imagine two paragraphs. If the two paragraphs are separated by one blank line, they are considered two separate paragraphs. However, if the two paragraphs are separated by two blank lines, the two paragraphs are considered as one very long paragraph. If the two are separated by three blank lines, they are considered two separate paragraphs; if the two are separated with four, the two are considered as one very long paragraph. The behaviour goes on in this pattern.

If we let *n* be the number of blank lines between two paragraphs, then the following piecewise function determines if the two paragraphs are seen as paragraphs or not.

$$\textbf{areParagraphs(n)} = \begin{cases} \textbf{true} & \textbf{if n mod 2} = \textbf{1} \\ \textbf{false} & \textbf{if n mod 2} = \textbf{0} \end{cases}$$

What remains certain is that one will never get more than one paragraph mark for two separate paragraphs. This means that excessive paragraph marks, and therefore excessive spacings, can be mitigated.

6.3.3. Backslash escapes

Both MD2ME and Markdown provide backslash escapes for the following characters. To get them on your page, simply put a backslash (\) before them.

- [ ] (square brackets)
- \ (backslash)
- ` (backtick)
- * (asterisk)
- _ (underscore)
- { } (curly braces)
- ( ) (parentheses)
- # (hashtags)
- + (plus sign)
- - (hyphen)
- . (dot)
- ’ (apostrophe)
- ! (exclamation mark)

For example, to get a literal asterisk, you would type `&Mas;` .

Additionally, MD2ME also provides backslash escapes for the following characters.

- < > (angle brackets)
- & (ampersand)

All of these backslash escapes may not work correctly when used on the same line as inline code.

6.3.4. Horizontal rules

Horizontal lines within the document can be made by typing 3 or more asterisks or hyphens on one line, optionally separated by spaces, like `* * *` or `- - -`.

The line above is a horizontal rule.

6.3.5. Lists

MD2ME provides support for both unordered and ordered lists. The lists cannot be nested to higher depths or orders. Each list item cannot span paragraphs. Each list item may not consist of other block elements.

For the ability to typeset more complex lists with paragraphs, nested lists, custom lexicographical set, you may need to touch TROFF and check out [lists.tmac](https://github.com/katt64/lists.tmac).

6.3.5.1. Unordered lists

Unordered lists are lists whose items' designator do not follow a particular lexicographical order. Unordered lists begin on a "paragraph" of their own. Each item of the list juxtaposes the next and should not be separated by blank lines. Here is an example of an unordered list:

- Attend high school class early: class project starts at 7.30.
- Meet Malin after school.
- Do the chores and help parents move house.
- Feed the hamster.
- Do homework.

It was typed as such:

```
- Attend high school class early: class project starts at 7.30.
- Meet Malin after school.
- Do the chores and help parents move house.
- Feed the hamster.
- Do homework.
```

Each item of an unordered list may start a line with `-` , `*` or `+` interchangeably.

6.3.5.2. Ordered list

Ordered lists are lists whose items' designator follow a particular lexicographical order. In MD2ME, the total order is assumed over the set of natural numbers from 1 to some limit n . For a total order assumed over other sets (alphabetical, roman numerals, &c.) refer to *lists.tmac* above.

Each item of the list juxtaposes the next and should not be separated by blank lines. Here is an example of an ordered list:

(1) This is an odd number.

(3) This is an odd number.

(4) This is an even number.

(8) This is an even number.

(17) This is a prime number.

The list was typed by:

```
1. This is an odd number.
3. This is an odd number.
4. This is an even number.
8. This is an even number.
17. This is a prime number.
```

Unlike in Markdown, the numberings of an ordered list as you type it **matter** . You must make sure to type the numbers as you want it to display on screen, as these are not automatically kept by MD2ME or TROFF.

6.3.6. Hard line breaks

To force a newline to be broken within the same paragraph, end a line with two spaces and a newline character and the next line shall be broken. In other words, given two consituent parts within the same paragraph, if two space characters followed by one linebreak separates them, a line break is generated between the two consituents.

For example, this
is a paragraph whose constituents
have been broken with hard breaks.

Essentially, it is the same rule as in Markdown. It is very hard to demonstrate the example above in a code block due to invisible whitespaces at the end of the lines. Though, if the lines above were to be parsed as input to `sed 'l;d'` , you would get something like this:

```
For example, this  $
is a paragraph whose constituents  $
have been broken with hard breaks.$
```

6.3.7. Inline code

Inline code is any block of code that exists within a paragraph of normal text. Inline code is delimited by ` (backticks) as if they were quotes.

Therefore, this line contains an inline code: `.. int *addr_af = a[b][c] + '\n'` . That was invalid C code, but it is just an example.

Inline code may span many input lines long. However, only the lines that do contain the delimiting backticks will have backslashes, square brackets, underscores, asterisks, &c. escaped appropriately. This escaping affects the entire input line on which the backticks are. This provides convenience in that all the backslashes and stuff are escaped automatically, but also means that any emphases, links, or backslashes on the same line as a backtick but **outside** of the inline code will not render properly. Simply break to a new input line if you need to have other inline annotations appear on the same line as code.

To have inline code that consists of a backtick within itself, use two backticks to delimit the inline code. Therefore, ``.tl `left`centre`right` `` renders well.

Although both types of inline code will escape all characters that otherwise have special meanings properly, there are several differences between double-backtick and single-backtick inline code. Firstly and obviously, double-backtick inline code allows the use of single bacticks or two non-contiguous backticks within the inline code it encompasses. Secondly, however, double-backtick code may **only** span one line in put, no more than that.

Within double-backtick inline codes, it is necessary to leave a space between a literal backtick you want to appear on your document and the two backticks that delimit the inline code. Just like in Markdown.

Unlike block codes, inline code is set in fill-mode. Meaning, any newlines within single-backtick inline codes will not be interpreted literally. Nonetheless, the inline code is set in a monospaced typeface.

The two paragraphs containing code were typed like this:

```
Therefore, this line contains an inline code: `.. int *addr_af = a[b][c] + '\n'`.
That was invalid C code, but it is just an example.

To have inline code that consists of a backtick within itself, use two backticks
to delimit the inline code.  Therefore, ``.tl `left`centre`right` `` renders
well.
```

HTML enties and MD2ME special characters will still need to be escaped within inline code. Therefore, to get `&` , you need to type `&` .

6.3.8. Fonts

Special fonts for *italics* , **bold** , and ***bold italics*** are provided for use within a paragraph. To activate them anywhere within a paragraph use asterisk(s) or underscore(s) to delimit the part you want the font changed. The number of asterisks denote different emphasis levels and therefore call for different fonts.

- Single asterisks like `*OMG*` call for italics.
- Double asterisks like `**OMG**` call for bold.

- Triple asterisks like `***OMG***` call for bold italics (i.e. both bold and italics).

It is not normally possible to have underlined bold italics, except ***you can do it*** if you are willing to use plain TROFF and accept the fact that your document may not completely display on a standard markdown compiler.

There can be italics, bolds, and bold italics on the same line, and each annotation can span several input lines.

Requests for special fonts within a line do not work properly on the same line as backticks.

6.3.9. Capitalized names and acronyms

Names like FORTRAN and BASIC have their point size reduced by 1 point before being displayed on the document. More generally, names and acronyms that consist of 3 or more capitalized letters consecutively from one or more of the English, Swedish, and Norwegian alphabets are surrounded by TROFF instructions to lower the point size by 1.

Therefore, simply typing in all capitals without any adornments, ABCÅÄÖÆØÉ, will work.

This is an automatic feature that is unique in MD2ME. The reason for having this is to give visually-pleasing output for names that consist of many capital letters. If those names were typed in the normal point size, they would look rather big and shouty due to an optical illusion; reducing their point sizes by 1 helps with that tremendously.

Capitalized names separated by spaces or newlines before there are ≥ 3 consecutive characters do not work. Compare ABCD and AB CD.

To get ≥ 3 capitalized letters printed in the same point size is a little bit difficult. In such cases, this feature can be quite a nuisance. Nonetheless, this can be done by interspersing `&Mnn;` after every 2nd character, like this: ABCD.

6.3.10. English ordinal numbers

In English typography and certain style guides, it is a convention to superscript the last two characters of the written ordinal numbers in shortened form. *first* , *second* , *third* , and *fourth* end with *st* , *nd* , *rd* , and *th* respectively. Therefore, they can either be written in full or as “shortened” ordinal numbers like: 1st, 2nd, 3rd, and 4th.

Unlike in Markdown, MD2ME treats these shortened ordinal numbers specially by superscripting the last two characters after the digits **if and only if** it is grammatically correct to do so. This superscripting feature is automatic; simply typing an ordinal quantity will spring this feature.

Thus, typing `1st 2nd 3rd 4th` , will get you: 1st 2nd 3rd 4th superscripted properly; but typing `1nd 2st 3th 4rd` will get you: 1nd 2st 3th 4rd without any superscripting.

MD2ME is actually pretty smart. 11th 12th 13th and 14th will have their last two characters superscripted propery as will 21st 22nd 23rd and 24th, but since 11st 12nd and 13rd is grammatically incorrect, no superscripting is done.

This superscripting does not work **at all** on the same line as backticks. This is so as to avoid any code list `print("3rd")` from being potentially misread as something else.

On a line of plain-text, if you would like to avoid superscripting, this automatic feature can prove to be quite nuisance. Nonetheless, to do this, put an `&Mnn;` between the number and the two-letter suffix, like this: `3&Mnn;rd` .

6.3.11. Footnote numbers

To get superscripted footnote numbers anywhere within a paragraph, simply surround any positive integer or one hashtag within encompassing square brackets. These would work: `[1]` and `[#]` .

The integer within the superscripted footnotes are simply for aesthetics if the document should later be compiled with a Markdown compiler, which do not have the capabilities for text footnotes. As a matter of fact, in MD2ME, the integers do not matter; they get auto-incremented by TROFF for each footnote entry called. Thus, the integers within brackets are ultimately ignored.

The advantage of using an integer over a hashtag is that it will still look good if the document should be rendered by a web browser later on. It will continue to look good until the reader cannot actually find a footnote entry associated to it. Thus, the writer may look like she has schizophrenia.

Footnote numbers are usually put **after** words, clauses, or sentences that need to be elaborated elsewhere without distracting the reader. In the case that they are put after clauses or sentences, the footnote numbers are usually put after any punctuation marks. This is just a style guide. Of course, in MD2ME you can put footnote numbers anywhere in a section header, paragraph, blockquote, except...

You cannot put footnote numbers on the same line as a backtick that has been used for inline code. The reason is the same: square brackets are escaped for code. If you want footnotes on the same logical line as code, put the square brackets on another physical input line that does not contain any quotes.

Of course, square brackets within code blocks and TROFF-copy lines/blocks are passed literally as square brackets.

This is a paragraph with a footnote number.[5] See the associated footnote entry for more details.

[5] Hello! See the section on footnote entries in MD2ME special requests!

Footnote numbers and footnote entries are MD2ME-specific features. Markdown does not support them.

6.3.12. Links

There are three types of links supported by both Markdown and MD2ME.

- Inline links with and without a title.
- Reference-style links with and without a title.
- Implicit links with no link text.

6.3.12.1. Inline links

To get an inline link without a title anywhere within a paragraph, intersperse something like this:

```
[my website](https://katt64.github.io)
```

This should get you: my website ⟨https://katt64.github.io⟩ . In TROFF, this is **not** a clickable link. Rather, it is set in blue without any underlines. The link's text is in regular font and the link's URL is surrounded by wide angled brackets.

To get an inline link with a title anywhere within a paragraph, intersperse something like this:

```
[my website](https://katt64.github.io "Personal Website")
```

This should get you: my website (Personal Website) ⟨https://katt64.github.io⟩ . As with titleless links, the link is set in blue. The link's text is set in regular font, the URL is surrounded by wide angled brackets, and the title is surrounded by parentheses after the link's text.

6.3.12.2. Reference-style links

Reference style links work the same as they would in Markdown. The following will be replaced with a link proper:

```
[my website][mywebs]
```

Between the link's text and reference ID, there can be one delimiting space, like this:

```
[my website] [mywebs]
```

The defintion for the link(s) whose reference ID is `mywebs` is defined on another line of its own:

```
[mywebs]: https://katt64.github.io (My personal website)
```

Within a link definition like that, the URL can optionally be surrounded by angled brackets. There can optionally be a link title after the URL, which can be surrounded by either a double quote ("), a single quote (') or a pair of matching parentheses ().

Reference IDs can be implicit. To clarify, the link's text can be a reference ID, but **iff** the link's text does not contain spaces. So, you can do something like this:

```
Please visit my [website][].

[website]: <https://katt64.github.io> "My personal website"
```

Most markdown parsers make it easy to use reference-style links, but in MD2ME, reference-style links are quite painful. When MD2ME finds a line that is a definition for a particular reference ID like the line above, it converts the line into TROFF instructions and stores it in another file called `reflinks.md2me` on the current working directory. This file contains all the reference IDs and their associated definitions.

`reflinks.md2me` is a file that collects all the link definitions thus far found. This is neccessary because `sed` runs through its input files in a one-pass fashion. Once it finds a link definition, it is not very feasible to go back to the link associated with that defintion and add more information to it. Therefore, if reference-style links are used in a document, the normal operation of md2me is a little bit different and you must follow these steps:

(1) Run md2me on your input file. You can pipe the output to `/dev/null` this time.

(2) Rename `reflinks.md2me` to `reflinks.md2me.tr` . Notice the additional extension: *tr* .

(3) Run md2me on the same input file again. The output from md2me on this run is the final output with all link definitions resolved.

On the 2nd run, md2me reads from the file called `reflinks.md2me.tr` and puts it **after** the 1st line of input from the file. This means that it is impossible to have reference-style links working properly if the link exists on a the first line of the file.

Unlike Markdown, IDs for reference-style links are case-sensitive and must **never** contain spaces.

There are no examples of reference-style links in this manual because of how complicated they are for MD2ME to parse.

6.3.12.3. Notes about links

There is a known bug in MD2ME. If a full stop or apostrophe is to follow a link, you must put a space between the full stop and the link, like this:

```
Visit [my website](https://katt64.github.io) .
```

No part of inline or reference style links may span more than one line. They must be kept on one line at all costs. Link definitions for reference-style links must also stay on one line no matter what.

6.3.12.4. Implicit links

Standard-compliant URLs and email addresses enclosed within angled brackets will be set in blue and the angled brackets will be replaced with more obtuse-angled brackets.

For example, interspersing this:

```
<https://katt64.github.io>
```

will get you something like this: ⟨https://katt64.github.io⟩.

Interspersing this:

```
<katt16777216@gmail.com>
```

will get you something like this: ⟨katt16777216@gmail.com⟩.

In MD2ME, email addresses are not obscured with random decimal/hexadecimal character entities. This is simply because there is no need to do it on if the document is to be printed on paper.

URLs must follow the following regular expression to be recognized as a URL to be transformed into a link. Ignore the space and backslash at the end of lines.

```
https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6} \
([-a-zA-Z0-9@:%_\+.~#?&\/\/=]*)
```

Likewise, email addresses must follow the following regular expression to be recognized as an email address to be transformed into a link.

```
[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}
```

To get a literal URL or email address within angle brackets, simply break the regular expressions by interspersing `&Mns;` or `&Mnn;` where necessary. So, interspersing this:

```
<http&Mns;s://katt64.github.io> <katt16777216@gmail.com&Mns;>
```

ill get you this: <https://katt64.github.io> <katt16777216@gmail.com>

Implicit links must remain on one line at all costs.

6.3.13. Images

Images can be included in two ways:

- Using an MD2ME's special request
- Using Markdown's syntax

The special request for MD2ME has already been detailed in a previous section. In this section, only the Markdown syntax will be mentioned.

Anywhere within a paragraph or on a line of its own, pictures can be included as such:

```
![alt text](images/first.eps "Title")
```

The title in double quotes is optional, and so is the alternate text. What is mandatory is the URL `images/first.eps` . Unlike standard Markdown, it is a lot more difficult to get images onto a TROFF document. Here are some criteria for the URL:

- The URL must point to a local resource. It can be an absolute path beginning at `/` or a relative path beginning in the current working directory.
- The local resource to which the URL points must exist, obviously.
- The local resource must be an image file that points to an encapsulated post-script (EPS) image.

The last criterion is what draws the line of what is possible in *both* Markdown and MD2ME. Most web browsers cannot open EPS files, but TROFF can only open EPS files for images. So, this is quite a major compatibility issue. In other words, you may choose either to work with web browsers or TROFF: your choice. :p

As mentioned long before, it is possible to convert images to EPS using free and open-source tools such as ImageMagick.

Images within the output document will not stay inline even if the instruction for an image was interspersed within a paragraph. They will ultimately start their own line and break the paragraph. Thus, it is best to use them on a line of their own.

The image will be centralized and will span the full width of the document. The alternate text and title will be put right under the image and set in italics.

It is not possible to have reference-style images, like this:

```
![alt text][ref]
```

6.3.14. Sentences

If you were born in the very late 20th century to present (1990-present), you are very likely taught to just leave one space after full stops when you write or type. It is good; just keep doing that for your grades, but...

The documents produced by MD2ME are designed to eventually be compiled by TROFF. The TROFF compiler then compiles the output document and generates it in print. This means that we must obey TROFF's line justification algorithm, which works a little bit differently from what you expect.

TROFF algorithm fills and justifies text on a line basis, not a paragraph basis. Thus, to make things easy for this algorithm, each sentence should start on a line of its own. In the modern world, this is kind of unacceptable.

A more acceptable solution to meet both ends is that, if possible, each sentence should be separated by two spaces. Your writing should then look something like the first line, not the second below. Notice the inter-sentence spacings.

```
Hello.  Anyone here?  OMG!  ARGH...
Hello. Anyone here? OMG! ARGH...
```

There should be two spaces after full stops, question marks, exclamation marks, or any sentence-boundary punctuation mark. The reason for this is that MD2ME will look for the specific permutation of a “punctuation mark followed by a space” and substitute that for a new line. Therefore, this automates the stipulation put out by TROFF’s algorithm that two sentences must be put on a line of their own.

There are implications if you let TROFF’s algorithm take its own course and you follow by its rules, sentences in the output document will be separated by two spaces. This is generally the desired and recommended behaviour. I like it. Do you like it? No? Oh, dear!

Two spaces are recommended for separating sentences, even in the Markdown file, because this distinguishes sentences that need to be double-space separated from acronyms like A. B. C., which have dots that do not act as sentence delimiters. The only way MD2ME knows whether the full stop finishes a sentence or just delimits letters in an acronym is if the full stop is followed by two spaces.

That is all you need to do to help MD2ME satisfy TROFF’s algorithm, really. The rest of this section details things MD2ME does automatically to save the day.

Additionally, to satisfy TROFF’s algorithm, MD2ME automatically strips beginning and trailing spaces from lines. This improves the algorithm’s performance greatly.

6.3.15. Proper names

MD2ME recognizes the following names as proper names or trademarks. The names will be replaced with the proper renditions of those names anywhere within a paragraph.

At present, the following names are known:

- MD2ME by `MD2ME`
- AT&T by `AT&T`
- C++ by `C++`
- LaTeX by `LaTeX`
- TeX by `TeX`

From this list, it can be seen clearly that to get the LaTeX mark, type `LaTeX` . The names are *case-sensitive* and will only render properly if the input name’s casing matches.

To override this behaviour. Either intersperse `&Mns;` or `&Mnn;` anywhere within the name on input. For example, to get a literal LaTeX, type `LaTe&Mnn;X`

6.3.16. Character entities

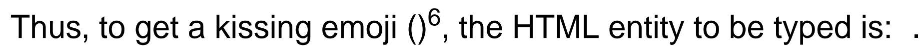
Character entities are useful when you need special characters you cannot type yourself or you need certain characters on the final output without interfering with other aspects of document rendition.

At present, the following HTML entities are supported:

- & by `&`
- © by `©`
- < by `<`
- > by `>`
- ≤ by `≤`
- ≥ by `≥`
-   by ` `
- — by `—`
- – by `–`
-   by ` `
- — by `—`
- – by `–`
- − by `−`
- § by `§`
- ∈ by `∈`
- ∼ by `∼`
- ¢ by `¢`
- £ by `£`
- « by `«t;`
- » by `»t;`
- ® by `®`
- ° by `°`
- ± by `±`
- ¶ by `¶`
- · by `·`
- ½ by `½`
- ‘ by `‘`
- ’ by `’`
- ‚ by `‚`
- “ by `“`
- ” by `”`
- „ by `„`
- ∀ by `∀`
- ∃ by `&exists;`
- † by `†`
- ‡ by `‡`
- • by `•`
- … by `…`
- ′ by `′`
- ″ by `″`
- € by `€`
- ™ by `™`

- ≈ by `≈`
- ≠ by `≠`
- ≤ by `≤`
- ≥ by `≥`
- É by `&Eaccute;`
- é by `&eaccute;`

Additionally, entering the hexadecimal unicode for any character is possible using the entity `&#x...;` where the dots are replaced with the hexadecimal unicode of the character, with 4-5 digits padded where necessary, and in all capitals. Thus, to get a kissing emoji ()[6], the HTML entity to be typed is: .



Note that ` ` does not give a *non-breaking* space. It is actually an unpaddable space, but the space can be broken break.

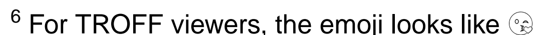
The entities above get replaced by TROFF's characters in the output.

Furthermore, MD2ME supports the following character entities. These entities start with a capital M after the ampersand and occupy only 1 – 2 characters. They are not compatible with standard Markdown.

- ½ by `&M12;`
- by `&Mds;` (a space the size of a digit)
- by `&Mhs;` (a space that is half of ` `)
- by `&Mks;` (a space that is a quarter of ` `)
- by `&Mns;` (a space of no width, removed by TROFF)
- ∅ by `&Mø;` (an empty set)
- ∅ by `&Mes;` (an empty set)
- * by `&Mas;` (asterisk)
- _ by `&Mus;` (underscore)
- $ by `&Mdo;` (a dollar sign that can escape EQN's delimiters)
- January 12, 2018 by `&Mtd;` (Today's date)
- Friday by `&Mdw;` (Today's day)
- 02:33:02 by `&Mtm;` (Current time in the 24-hour clock)
- [by `&Mos;` (special opening square bracket for dire times)
-] by `&Mcs;` (special closing square bracket for dire times)
- ` by `&Mga;` (literal grave accent)
- ´ by `&Maa;` (literal accute accent)
- by `&Mbs;` (Bell System's logo)
- “ by `&Mlq;` (left double quote)
- ” by `&Mrq;` (right double quote)
- ~ by `⤅` (approximately equal to)
- by `&Mhb;` (hyphenless breaking point)

---

[6] For TROFF viewers, the emoji looks like 😘



- ☜ b   y `&Mlh;` (hand pointing left)
- ☞ by `&Mrh;` (hand pointing right)
- @ by `&Mca;` (literal commercial at)
- (by `&Mlp;` (literal left parenthesis)
-) by `&Mrp;` (literal right parenthesis)
- ! by `&Mex;` (literal exclamation mark)
- - by `&Mhy;` (literal hyphen)
- by `&Mnn;` (a character that is removed by MD2ME)
- 38 by `&Mpn;` (the current page number)
- by `&Mee;` (An escape character to TROFF)

Additionally, for more knowledgable users of TROFF, entering a named-character in TROFF is possible with the entity `&Mx...;` which corresponds to `\[...]` in the output. For example, to get the Bell System's logo, type `&Mxbs;` which will get substituted for `\[bs]` .

These characters might be really confusing for those not familiar with typography and TROFF. These are characters that are mostly used internally by MD2ME as ancillary entities that will get replaced by the appropriate characters at a later time. Some of these entities are really, really, really unstable if you do not know how to use them properly.

Give your skills a try: when I had to type `\[bs]` and get it to display on this document, do you know why I had to type `&Mee;e[bs]` ?

7. Aftermath

In the future, I wish to implement a better Markdown-to-TROFF converter made in an actual programming language like Python or C.

Doing it in sed is absolutely crazy. I never thought that the parser would be coerced into expanding to cover this much of Markdown's syntax. All I initially wanted was a quick way to take notes and print them out as proof-readable essays.

There are too many idiosyncracies and rules that could be relaxed. There are also a few flaws here and there that I haven't got the energy or time to fix.

Consider this parser implementation **not** as a real Markdown-to-TROFF converter, although it does work to some extent. Consider this as an idea I had when I was 17 (I am 17 as I am writing this). Above all, consider this as a starting point and a draft for an idea that I think is worthwhile living for, that I think is worthwhile implementing into another fully-featured programming language when I have the wherewithal.

Nonetheless, you are welcome to make improvements or derivatives of this sed script.

8. Authors

Stephanie Björk (Katt) ⟨katt16777216@gmail.com⟩. The writing of the `sed` script began on January 3 2018, and reached stability on January 12, 2018.

9. Licenses

The script is licensed under the 3-clause BSD license.


```
BSD 3-Clause License

Copyright (c) 2018, Stephanie Björk
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name of the copyright holder nor the names of its
  contributors may be used to endorse or promote products derived from
  this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```


The license text above is not included as a comment within the `sed` script because it is rather quite long.

This manual was written in Markdown and took 3 days to write. It is licensed under CC BY-SA 4.0 ⟨https://creativecommons.org/licenses/by-sa/4.0/⟩ .

10. Special thanks

I would like to thank the following persons who have been with me throughout this endeavour. It was such a fun, great, and enlightening experience. These are the persons I could not have done anything without.

- **My mom** , a forever understanding and loving person.
- **Francesca** for the addition of ∀ and ∃ entities in MD2ME.
- **Many people at work** . They are the reason why I needed to create the sed script, which was quite an enlightening experience.
- **Hund** for his support on everything and for trying out the script in its early stages. He makes many interesting blog posts. Visit his blog on ⟨https://hund.github.io/⟩

I also have to thank my ex-partner for her patience and absolute radio silence over the emails. Her birthday was on the same day I made the `sed` script. I was going to write her a birthday card, but she will have to wait. Haha!