NAME

pandoc_markdown - markdown syntax for pandoc(1)

DESCRIPTION

Pandoc understands an extended and slightly revised version of John Gruber's markdown syntax. This document explains the syntax, noting differences from standard markdown. Except where noted, these differences can be suppressed by specifying the `--strict` command-line option.

PHILOSOPHY

Markdown is designed to be easy to write, and, even more importantly, easy to read:

> A Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions. -- John Gruber

This principle has guided pandoc's decisions in finding syntax for tables, footnotes, and other extensions.

There is, however, one respect in which pandoc's aims are different from the original aims of markdown. Whereas markdown was originally designed with HTML generation in mind, pandoc is designed for multiple output formats. Thus, while pandoc allows the embedding of raw HTML, it discourages it, and provides other, non-HTMLish ways of representing important document elements like definition lists, tables, mathematics, and footnotes.

PARAGRAPHS

A paragraph is one or more lines of text followed by one or more blank line. Newlines are treated as spaces, so you can reflow your paragraphs as you like. If you need a hard line break, put two or more spaces at the end of a line, or or type a backslash followed by a newline.

HEADERS

There are two kinds of headers, Setext and atx.

Setext-style headers

A setext-style header is a line of text "underlined" with a row of = signs (for a level one header) of - signs (for a level two header):

```
A level-one header
==================

A level-two header
------------------
```

The header text can contain inline formatting, such as emphasis (see Inline formatting, below).

Atx-style headers

An Atx-style header consists of one to six # signs and a line of text, optionally followed by any number of # signs. The number of # signs at the beginning of the line is the header level:

```
## A level-two header

### A level-three header ###
```

As with setext-style headers, the header text can contain formatting:

```
# A level-one header with a [link](/url) and *emphasis*
```

Standard markdown syntax does not require a blank line before a header. Pandoc does require this (except, of course, at the beginning of the document). The reason for the requirement is that it is all too easy for a # to end up at the beginning of a line by accident (perhaps through line wrapping). Consider, for example:

```
I like several of their flavors of ice cream:
#22, for example, and #5.
```

Header identifiers in HTML

Pandoc extension.

Each header element in pandoc's HTML output is given a unique identifier. This identifier is based on the text of the header. To derive the identifier from the header text,

- Remove all formatting, links, etc.
- Remove all punctuation, except underscores, hyphens, and periods.
- Replace all spaces and newlines with hyphens.
- Convert all alphabetic characters to lowercase.
- Remove everything up to the first letter (identifiers may not begin with a number or punctuation mark).
- If nothing is left after this, use the identifier `section`.

Thus, for example,

Header	Identifier
Header identifiers in HTML	`header-identifiers-in-html`
Dogs?--in *my* house?	`dogs--in-my-house`
HTML, S5, or RTF?	`html-s5-or-rtf`
3. Applications	`applications`
33	`section`

These rules should, in most cases, allow one to determine the identifier from the header text. The exception is when several headers have the same text; in this case, the first will get an identifier as described above; the second will get the same identifier with `-1` appended; the third with `-2`; and so on.

These identifiers are used to provide link targets in the table of contents generated by the `--toc`|`--table-of-contents` option. They also make it easy to provide links from one section of a document to another. A link to this section, for example, might look like this:

```
See the section on
[header identifiers](#header-identifiers-in-html).
```

Note, however, that this method of providing links to sections works only in HTML.

If the `--section-divs` option is specified, then each section will be wrapped in a `div` (or a `section`, if `--html5` was specified), and the identifier will be attached to the enclosing `<div>` (or `<section>`) tag rather than the header itself. This allows entire sections to be manipulated using javascript or treated differently in CSS.

BLOCK QUOTATIONS

Markdown uses email conventions for quoting blocks of text. A block quotation is one or more paragraphs or other block elements (such as lists or headers), with each line preceded by a > character and a space. (The > need not start at the left margin, but it should not be indented more than three spaces.)

```
> This is a block quote. This
> paragraph has two lines.
>
> 1. This is a list inside a block quote.
> 2. Second item.
```

A "lazy" form, which requires the > character only on the first line of each block, is also allowed:

```
> This is a block quote. This
paragraph has two lines.

> 1. This is a list inside a block quote.
2. Second item.
```

Among the block elements that can be contained in a block quote are other block quotes. That is, block quotes can be nested:

```
> This is a block quote.
>
> > A block quote within a block quote.
```

Standard markdown syntax does not require a blank line before a block quote. Pandoc does require this (except, of course, at the beginning of the document). The reason for the requirement is that it is all too easy for a > to end up at the beginning of a line by accident (perhaps through line wrapping). So, unless `--strict` is used, the following does not produce a nested block quote in pandoc:

```
> This is a block quote.
>> Nested.
```

VERBATIM (CODE) BLOCKS

Indented code blocks

A block of text indented four spaces (or one tab) is treated as verbatim text: that is, special characters do not trigger special formatting, and all spaces and line breaks are preserved. For example,

```
    if (a > 3) {
      moveShip(5 * gravity, DOWN);
    }
```

The initial (four space or one tab) indentation is not considered part of the verbatim text, and is removed in the output.

Note: blank lines in the verbatim text need not begin with four spaces.

Delimited code blocks

Pandoc extension.

In addition to standard indented code blocks, Pandoc supports *delimited* code blocks. These begin with a row of three or more tildes (˜) and end with a row of tildes that must be at least as long as the starting row. Everything between the tilde-lines is treated as code. No indentation is necessary:

```
~~~~~~~
if (a > 3) {
  moveShip(5 * gravity, DOWN);
}
~~~~~~~
```

Like regular code blocks, delimited code blocks must be separated from surrounding text by blank lines.

If the code itself contains a row of tildes, just use a longer row of tildes at the start and end:

```
~~~~~~~~~~~~~~~~
~~~~~~~~~~
code including tildes
~~~~~~~~~~
~~~~~~~~~~~~~~~~
```

Optionally, you may specify the language of the code block using this syntax:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ {.haskell .numberLines}
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
               qsort (filter (>= x) xs)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Some output formats can use this information to do syntax highlighting. Currently, the only output format that uses this information is HTML.

If pandoc has been compiled with syntax highlighting support, then the code block above will appear highlighted, with numbered lines. (To see which languages are supported, do `pandoc --version`.)

If pandoc has not been compiled with syntax highlighting support, the code block above will appear as follows:

```
<pre class="haskell">
  <code>
  ...
  </code>
</pre>
```

LISTS

Bullet lists

A bullet list is a list of bulleted list items. A bulleted list item begins with a bullet (`*`, `+`, or `-`). Here is a simple example:

```
* one
* two
* three
```

This will produce a "compact" list. If you want a "loose" list, in which each item is formatted as a paragraph, put spaces between the items:

```
* one

* two

* three
```

The bullets need not be flush with the left margin; they may be indented one, two, or three spaces. The bullet must be followed by whitespace.

List items look best if subsequent lines are flush with the first line (after the bullet):

```
* here is my first
  list item.
* and my second.
```

But markdown also allows a "lazy" format:

```
* here is my first
list item.
* and my second.
```

The four-space rule

A list item may contain multiple paragraphs and other block-level content. However, subsequent paragraphs must be preceded by a blank line and indented four spaces or a tab. The list will look better if the first paragraph is aligned with the rest:

```
  * First paragraph.

    Continued.

  * Second paragraph. With a code block, which must be indented
    eight spaces:

        { code }
```

List items may include other lists. In this case the preceding blank line is optional. The nested list must be indented four spaces or one tab:

```
* fruits
    + apples
        - macintosh
        - red delicious
    + pears
    + peaches
* vegetables
    + brocolli
    + chard
```

As noted above, markdown allows you to write list items "lazily," instead of indenting continuation lines. However, if there are multiple paragraphs or other blocks in a list item, the first line of each must be indented.

```
+ A lazy, lazy, list
item.
```

```
+ Another one; this looks
bad but is legal.

    Second paragraph of second
list item.
```

Note: Although the four-space rule for continuation paragraphs comes from the official markdown syntax guide, the reference implementation, `Markdown.pl`, does not follow it. So pandoc will give different results than `Markdown.pl` when authors have indented continuation paragraphs fewer than four spaces.

The markdown syntax guide is not explicit whether the four-space rule applies to *all* block-level content in a list item; it only mentions paragraphs and code blocks. But it implies that the rule applies to all block-level content (including nested lists), and pandoc interprets it that way.

Ordered lists

Ordered lists work just like bulleted lists, except that the items begin with enumerators rather than bullets.

In standard markdown, enumerators are decimal numbers followed by a period and a space. The numbers themselves are ignored, so there is no difference between this list:

```
1.  one
2.  two
3.  three
```

and this one:

```
5.  one
7.  two
1.  three
```

Pandoc extension.

Unlike standard markdown, Pandoc allows ordered list items to be marked with uppercase and lowercase letters and roman numerals, in addition to arabic numerals. List markers may be enclosed in parentheses or followed by a single right-parentheses or period. They must be separated from the text that follows by at least one space, and, if the list marker is a capital letter with a period, by at least two spaces.[1]

Pandoc also pays attention to the type of list marker used, and to the starting number, and both of these are preserved where possible in the output format. Thus, the following yields a list with numbers followed by a single parenthesis, starting with 9, and a sublist with lowercase roman numerals:

```
 9)  Ninth
10)  Tenth
11)  Eleventh
       i. subone
      ii. subtwo
     iii. subthree
```

Note that Pandoc pays attention only to the *starting* marker in a list. So, the following yields a list numbered sequentially starting from 2:

```
(2) Two
(5) Three
1.  Four
*   Five
```

If default list markers are desired, use `#.`:

```
#.  one
#.  two
#.  three
```

Definition lists

Pandoc extension.

Pandoc supports definition lists, using a syntax inspired by PHP Markdown Extra and reStructuredText:[2]

```
Term 1

:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

Each term must fit on one line, which may optionally be followed by a blank line, and must be followed by one or more definitions. A definition begins with a colon or tilde, which may be indented one or two spaces. A term may have multiple definitions, and each definition may consist of one or more block elements (paragraph, code block, list, etc.) , each indented four spaces or one tab stop.

If you leave space after the definition (as in the example above), the blocks of the definitions will be considered paragraphs. In some output formats, this will mean greater spacing between term/definition pairs. For a compact definition list, do not leave space between the definition and the next term:

```
Term 1
  ˜ Definition 1
Term 2
  ˜ Definition 2a
  ˜ Definition 2b
```

Numbered example lists

Pandoc extension.

The special list marker `@` can be used for sequentially numbered examples. The first list item with a `@` marker will be numbered '1', the next '2', and so on, throughout the document. The numbered examples need not occur in a single list; each new list using `@` will take up where the last stopped. So, for example:

```
(@)  My first example will be numbered (1).
(@)  My second example will be numbered (2).

Explanation of examples.

(@)  My third example will be numbered (3).
```

Numbered examples can be labeled and referred to elsewhere in the document:

```
(@good)  This is a good example.

As (@good) illustrates, ...
```

The label can be any string of alphanumeric characters, underscores, or hyphens.

Compact and loose lists

Pandoc behaves differently from `Markdown.pl` on some "edge cases" involving lists. Consider this source:

```
+   First
+   Second:
      -   Fee
      -   Fie
      -   Foe

+   Third
```

Pandoc transforms this into a "compact list" (with no <p> tags around "First", "Second", or "Third"), while markdown puts <p> tags around "Second" and "Third" (but not "First"), because of the blank space around "Third". Pandoc follows a simple rule: if the text is followed by a blank line, it is treated as a paragraph. Since "Second" is followed by a list, and not a blank line, it isn't treated as a paragraph. The fact that the list is followed by a blank line is irrelevant. (Note: Pandoc works this way even when the `--strict` option is specified. This behavior is consistent with the official markdown syntax description, even though it is different from that of `Markdown.pl`.)

Ending a list

What if you want to put an indented code block after a list?

```
-   item one
-   item two

    { my code block }
```

Trouble! Here pandoc (like other markdown implementations) will treat `{ my code block }` as the second paragraph of item two, and not as a code block.

To "cut off" the list after item two, you can insert some non-indented content, like an HTML comment, which won't produce visible output in any format:

```
-   item one
-   item two

<!-- end of list -->

    { my code block }
```

You can use the same trick if you want two consecutive lists instead of one big list:

```
1.  one
2.  two
3.  three

<!-- -->

a.  uno
b.  dos
c.  tres
```

HORIZONTAL RULES

A line containing a row of three or more `*`, `-`, or `_` characters (optionally separated by spaces) produces a horizontal rule:

```
*  *  *  *

---------------
```

TABLES

Pandoc extension.

Three kinds of tables may be used. All three kinds presuppose the use of a fixed-width font, such as Courier.

Simple tables look like this:

```
  Right     Left     Center     Default
-------     ------ ----------   -------
     12     12        12            12
    123     123       123          123
      1     1          1             1

Table:  Demonstration of simple table syntax.
```

The headers and table rows must each fit on one line. Column alignments are determined by the position of the header text relative to the dashed line below it:[3]

- If the dashed line is flush with the header text on the right side but extends beyond it on the left, the column is right-aligned.

- If the dashed line is flush with the header text on the left side but extends beyond it on the right, the column is left-aligned.
- If the dashed line extends beyond the header text on both sides, the column is centered.
- If the dashed line is flush with the header text on both sides, the default alignment is used (in most cases, this will be left).

The table must end with a blank line, or a line of dashes followed by a blank line. A caption may optionally be provided (as illustrated in the example above). A caption is a paragraph beginning with the string `Table:` (or just `:`), which will be stripped off. It may appear either before or after the table.

The column headers may be omitted, provided a dashed line is used to end the table. For example:

```
-------     ------ ----------   -------
     12     12        12             12
    123     123       123           123
      1     1          1              1
-------     ------ ----------   -------
```

When headers are omitted, column alignments are determined on the basis of the first line of the table body. So, in the tables above, the columns would be right, left, center, and right aligned, respectively.

Multiline tables allow headers and table rows to span multiple lines of text (but cells that span multiple columns or rows of the table are not supported). Here is an example:

```
-------------------------------------------------------------
 Centered   Default           Right Left
  Header    Aligned         Aligned Aligned
----------- ------- --------------- -------------------------
   First    row                12.0 Example of a row that
                                    spans multiple lines.

  Second    row                 5.0 Here's another one. Note
                                    the blank line between
                                    rows.
-------------------------------------------------------------

Table: Here's the caption. It, too, may span
multiple lines.
```

These work like simple tables, but with the following differences:

- They must begin with a row of dashes, before the header text (unless the headers are omitted).
- They must end with a row of dashes, then a blank line.
- The rows must be separated by blank lines.

In multiline tables, the table parser pays attention to the widths of the columns, and the writers try to reproduce these relative widths in the output. So, if you find that one of the columns is too narrow in the output, try widening it in the markdown source.

Headers may be omitted in multiline tables as well as simple tables:

```
----------- ------- --------------- -------------------------
   First    row                12.0 Example of a row that
                                    spans multiple lines.

  Second    row                 5.0 Here's another one. Note
                                    the blank line between
                                    rows.
-------------------------------------------------------------

: Here's a multiline table without headers.
```

It is possible for a multiline table to have just one row, but the row should be followed by a blank line (and then the row of dashes that ends the table), or the table may be interpreted as a simple table.

Grid tables look like this:

```
: Sample grid table.

+---------------+---------------+--------------------+
| Fruit         | Price         | Advantages         |
+===============+===============+====================+
| Bananas       | $1.34         | - built-in wrapper |
|               |               | - bright color     |
+---------------+---------------+--------------------+
| Oranges       | $2.10         | - cures scurvy     |
|               |               | - tasty            |
+---------------+---------------+--------------------+
```

The row of =s separates the header from the table body, and can be omitted for a headerless table. The cells of grid tables may contain arbitrary block elements (multiple paragraphs, code blocks, lists, etc.) . Alignments are not supported, nor are cells that span multiple columns or rows. Grid tables can be created easily using Emacs table mode.

TITLE BLOCK

Pandoc extension.

If the file begins with a title block

```
% title
% author(s) (separated by semicolons)
% date
```

it will be parsed as bibliographic information, not regular text. (It will be used, for example, in the title of standalone LaTeX or HTML output.)
The block may contain just a title, a title and an author, or all three elements. If you want to include an author but no title, or a title and a date but no author, you need a blank line:

```
%
% Author

% My title
%
% June 15, 2006
```

The title may occupy multiple lines, but continuation lines must begin with leading space, thus:

```
% My title
  on multiple lines
```

If a document has multiple authors, the authors may be put on separate lines with leading space, or separated by semicolons, or both. So, all of the following are equivalent:

```
% Author One
  Author Two

% Author One; Author Two

% Author One;
  Author Two
```

The date must fit on one line.

All three metadata fields may contain standard inline formatting (italics, links, footnotes, etc.) .

Title blocks will always be parsed, but they will affect the output only when the `--standalone` (`-s`) option is chosen. In HTML output, titles will appear twice: once in the document head -- this is the title that will appear at the top of the window in a browser -- and once at the beginning of the document body. The title in the document head can have an optional prefix attached (`--title-prefix` or `-T` option). The title in the body appears as an H1 element with class "title", so it can be suppressed or reformatted with CSS. If a title prefix is specified with `-T` and no title block appears in the document, the title prefix will be used by itself as the HTML title.

The man page writer extracts a title, man page section number, and other header and footer information from the title line. The title is assumed to be the first word on the title line, which may optionally end with a (single-digit) section number in parentheses. (There should be no space between the title and the parentheses.)
Anything after this is assumed to be additional footer and header text. A single pipe character (|) should be used to separate the footer text from the header text. Thus,

```
% PANDOC(1)
```

will yield a man page with the title PANDOC and section 1.

```
% PANDOC(1) Pandoc User Manuals
```

will also have "Pandoc User Manuals" in the footer.

```
% PANDOC(1) Pandoc User Manuals | Version 4.0
```

will also have "Version 4.0" in the header.

BACKSLASH ESCAPES

Except inside a code block or inline code, any punctuation or space character preceded by a backslash will be treated literally, even if it would normally indicate formatting. Thus, for example, if one writes

```
*\*hello\**
```

one will get

```
<em>*hello*</em>
```

instead of

```
<strong>hello</strong>
```

This rule is easier to remember than standard markdown's rule, which allows only the following characters to be backslash-escaped:

```
\'*_{}[]()>#+-.!
```

(However, if the `--strict` option is supplied, the standard markdown rule will be used.)

A backslash-escaped space is parsed as a nonbreaking space. It will appear in TeX output as ˜ and in HTML and XML as `\&#160;` or `\&nbsp;`.

A backslash-escaped newline (i.e. a backslash occurring at the end of a line) is parsed as a hard line break. It will appear in TeX output as `\\` and in HTML as `<br />`. This is a nice alternative to markdown's "invisible" way of indicating hard line breaks using two trailing spaces on a line.

Backslash escapes do not work in verbatim contexts.

SMART PUNCTUATION

If the `--smart` option is specified, pandoc will produce typographically correct output, converting straight quotes to curly quotes, `---` and `--` to Em-dashes, and `...` to ellipses. Nonbreaking spaces are inserted after certain abbreviations, such as "Mr."

INLINE FORMATTING

Emphasis

To *emphasize* some text, surround it with `*`s or `_`, like this:

```
This text is _emphasized with underscores_, and this
is *emphasized with asterisks*.
```

Double `*` or `_` produces **strong emphasis**:

```
This is **strong emphasis** and __with underscores__.
```

A `*` or `_` character surrounded by spaces, or backslash-escaped, will not trigger emphasis:

```
This is * not emphasized *, and \*neither is this\*.
```

Because `_` is sometimes used inside words and identifiers, pandoc does not interpret a `_` surrounded by alphanumeric characters as an emphasis marker. If you want to emphasize just part of a word, use `*`:

```
feas*ible*, not feas*able*.
```

Strikeout

Pandoc extension.

To strikeout a section of text with a horizontal line, begin and end it with `~~`. Thus, for example,

```
This ~~is deleted text.~~
```

Superscripts and subscripts

Pandoc extension.

Superscripts may be written by surrounding the superscripted text by `^` characters; subscripts may be written by surrounding the subscripted text by `~` characters. Thus, for example,

```
H~2~O is a liquid.  2^10^ is 1024.
```

If the superscripted or subscripted text contains spaces, these spaces must be escaped with backslashes. (This is to prevent accidental superscripting and subscripting through the ordinary use of `~` and `^`.) Thus, if you want the letter P with 'a cat' in subscripts, use `P~a\ cat~`, not `P~a cat~`.

Verbatim

To make a short span of text verbatim, put it inside backticks:

```
What is the difference between ‘>>=‘ and ‘>>‘?
```

If the verbatim text includes a backtick, use double backticks:

```
Here is a literal backtick ‘‘ ‘ ‘‘.
```

(The spaces after the opening backticks and before the closing backticks will be ignored.)

The general rule is that a verbatim span starts with a string of consecutive backticks (optionally followed by a space) and ends with a string of the same number of backticks (optionally preceded by a space).

Note that backslash-escapes (and other markdown constructs) do not work in verbatim contexts:

```
This is a backslash followed by an asterisk: ‘\*‘.
```

MATH

Pandoc extension.

Anything between two `$` characters will be treated as TeX math. The opening `$` must have a character immediately to its right, while the closing `$` must have a character immediately to its left. Thus, `$20,000 and $30,000` won't parse as math. If for some reason you need to enclose text in literal `$` characters, backslash-escape them and they won't be treated as math delimiters.

TeX math will be printed in all output formats. How it is rendered depends on the output format:

Markdown, reStructuredText, LaTeX, Org-Mode, ConTeXt
It will appear verbatim between `$` characters.

reStructuredText
It will be rendered using an interpreted text role `:math:`, as described here.

Texinfo
It will be rendered inside a `@math` command.

groff man
It will be rendered verbatim without `$`'s.

MediaWiki
It will be rendered inside `<math>` tags.

Textile It will be rendered inside `<span class="math">` tags.

RTF, Docbook, OpenDocument, ODT
It will be rendered, if possible, using unicode characters, and will otherwise appear verbatim.

HTML, Slidy, S5, EPUB
The way math is rendered in HTML will depend on the command-line options selected:

1. The default is to render TeX math as far as possible using unicode characters, as with RTF, Docbook, and OpenDocument output. Formulas are put inside a `span` with `class="math"`, so that they may be styled differently from the surrounding text if

needed.

2. If the `--latexmathml` option is used, TeX math will be displayed between $ or $$ characters and put in `<span>` tags with class `LaTeX`. The LaTeXMathML script will be used to render it as formulas. (This trick does not work in all browsers, but it works in Firefox. In browsers that do not support LaTeXMathML, TeX math will appear verbatim between $ characters.)
3. If the `--jsmath` option is used, TeX math will be put inside `<span>` tags (for inline math) or `<div>` tags (for display math) with class `math`. The jsMath script will be used to render it.
4. If the `--mimetex` option is used, the mimeTeX CGI script will be called to generate images for each TeX formula. This should work in all browsers. The `--mimetex` option takes an optional URL as argument. If no URL is specified, it will be assumed that the mimeTeX CGI script is at `/cgi-bin/mimetex.cgi`.
5. If the `--gladtex` option is used, TeX formulas will be enclosed in `<eq>` tags in the HTML output. The resulting `htex` file may then be processed by gladTeX, which will produce image files for each formula and an `html` file with links to these images. So, the procedure is:

```
pandoc -s --gladtex myfile.txt -o myfile.htex
gladtex -d myfile-images myfile.htex
# produces myfile.html and images in myfile-images
```

6. If the `--webtex` option is used, TeX formulas will be converted to `<img>` tags that link to an external script that converts formulas to images. The formula will be URL-encoded and concatenated with the URL provided. If no URL is specified, the Google Chart API will be used (`http://chart.apis.google.com/chart?cht=tx&chl=`).

RAW HTML

Markdown allows you to insert raw HTML anywhere in a document (except verbatim contexts, where `<`, `>`, and `&` are interpreted literally).

The raw HTML is passed through unchanged in HTML, S5, Slidy, EPUB, Markdown, and Textile output, and suppressed in other formats.

Pandoc extension.

Standard markdown allows you to include HTML "blocks": blocks of HTML between balanced tags that are separated from the surrounding text with blank lines, and start and end at the left margin. Within these blocks, everything is interpreted as HTML, not markdown; so (for example), `*` does not signify emphasis.

Pandoc behaves this way when `--strict` is specified; but by default, pandoc interprets material between HTML block tags as markdown. Thus, for example, Pandoc will turn

```
<table>
      <tr>
            <td>*one*</td>
            <td>[a link](http://google.com)</td>
      </tr>
</table>
```

into

```
<table>
      <tr>
            <td><em>one</em></td>
            <td><a href="http://google.com">a link</a></td>
      </tr>
</table>
```

whereas `Markdown.pl` will preserve it as is.

There is one exception to this rule: text between `<script>` and `<style>` tags is not interpreted as markdown.

This departure from standard markdown should make it easier to mix markdown with HTML block elements. For example, one can surround a block of markdown text with `<div>` tags without preventing it from being interpreted as markdown.

RAW TEX

Pandoc extension.

In addition to raw HTML, pandoc allows raw LaTeX, TeX, and ConTeXt to be included in a document. Inline TeX commands will be preserved and passed unchanged to the LaTeX and ConTeXt writers. Thus, for example, you can use LaTeX to include BibTeX citations:

```
This result was proved in \cite{jones.1967}.
```

Note that in LaTeX environments, like

```
\begin{tabular}{|l|l|}\hline
Age & Frequency \\ \hline
18--25  & 15 \\
26--35  & 33 \\
36--45  & 22 \\ \hline
\end{tabular}
```

the material between the begin and end tags will be interpreted as raw LaTeX, not as markdown.

Inline LaTeX is ignored in output formats other than Markdown, LaTeX, and ConTeXt.

Macros

For output formats other than LaTeX, pandoc will parse LaTeX `\newcommand` and `\renewcommand` definitions and apply the resulting macros to all LaTeX math. So, for example, the following will work in all output formats, not just LaTeX:

```
\newcommand{\tuple}[1]{\langle #1 \rangle}

$\tuple{a, b, c}$
```

In LaTeX output, the `\newcommand` definition will simply be passed unchanged to the output.

LINKS

Markdown allows links to be specified in several ways.

Automatic links

If you enclose a URL or email address in pointy brackets, it will become a link:

```
<http://google.com>
<sam@green.eggs.ham>
```

Inline links

An inline link consists of the link text in square brackets, followed by the URL in parentheses. (Optionally, the URL can be followed by a link title, in quotes.)

```
This is an [inline link](/url), and here's [one with
a title](http://fsf.org "click here for a good time!").
```

There can be no space between the bracketed part and the parenthesized part. The link text can contain formatting (such as emphasis), but the title cannot.

Reference links

An *explicit* reference link has two parts, the link itself and the link definition, which may occur elsewhere in the document (either before or after the link).

The link consists of link text in square brackets, followed by a label in square brackets. (There can be space between the two.)
The link definition must begin at the left margin or indented no more than three spaces. It consists of the bracketed label, followed by a colon and a space, followed by the URL, and optionally (after a space) a link title either in quotes or in parentheses.

Here are some examples:

```
[my label 1]: /foo/bar.html  "My title, optional"
[my label 2]: /foo
[my label 3]: http://fsf.org (The free software foundation)
```

```
[my label 4]: /bar#special  'A title in single quotes'
```

The URL may optionally be surrounded by angle brackets:

```
[my label 5]: <http://foo.bar.baz>
```

The title may go on the next line:

```
[my label 3]: http://fsf.org
  "The free software foundation"
```

Note that link labels are not case sensitive. So, this will work:

```
Here is [my link][FOO]

[Foo]: /bar/baz
```

In an *implicit* reference link, the second pair of brackets is empty, or omitted entirely:

```
See [my website][], or [my website].

[my website]: http://foo.bar.baz
```

IMAGES

A link immediately preceded by a `!` will be treated as an image. The link text will be used as the image's alt text:

```
![la lune](lalune.jpg "Voyage to the moon")

![movie reel]

[movie reel]: movie.gif
```

Pictures with captions

Pandoc extension.

An image occurring by itself in a paragraph will be rendered as a figure with a caption.[4] (In LaTeX, a figure environment will be used; in HTML, the image will be placed in a `div` with class `figure`, together with a caption in a `p` with class `caption`.)
The image's alt text will be used as the caption.

```
![This is the caption](/url/of/image.png)
```

If you just want a regular inline image, just make sure it is not the only thing in the paragraph. One way to do this is to insert a nonbreaking space after the image:

```
![This image won't be a figure](/url/of/image.png)\
```

FOOTNOTES

Pandoc extension.

Pandoc's markdown allows footnotes, using the following syntax:

```
Here is a footnote reference,[^1] and another.[^longnote]

[^1]: Here is the footnote.

[^longnote]: Here's one with multiple blocks.

    Subsequent paragraphs are indented to show that they
belong to the previous footnote.

        { some.code }

    The whole paragraph can be indented, or just the first
    line.  In this way, multi-paragraph footnotes work like
    multi-paragraph list items.

This paragraph won't be part of the note, because it
```

```
isn't indented.
```

The identifiers in footnote references may not contain spaces, tabs, or newlines. These identifiers are used only to correlate the footnote reference with the note itself; in the output, footnotes will be numbered sequentially.

The footnotes themselves need not be placed at the end of the document. They may appear anywhere except inside other block elements (lists, block quotes, tables, etc.) .

Inline footnotes are also allowed (though, unlike regular notes, they cannot contain multiple paragraphs). The syntax is as follows:

```
Here is an inline note.^[Inlines notes are easier to write, since
you don't have to pick an identifier and move down to type the
note.]
```

Inline and regular footnotes may be mixed freely.

CITATIONS

Pandoc extension.

Pandoc can automatically generate citations and a bibliography in a number of styles (using Andrea Rossato's `hs-citeproc`). In order to use this feature, you will need a bibliographic database in one of the following formats:

Format	File extension
MODS	.mods
BibTeX	.bib
BibLaTeX	.bbx
RIS	.ris
EndNote	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
Copac	.copac
JSON citeproc	.json

You will need to specify the bibliography file using the `--bibliography` command-line option (which may be repeated if you have several bibliographies).

By default, pandoc will use a Chicago author-date format for citations and references. To use another style, you will need to use the `--csl` option to specify a CSL 1.0 style file. A primer on creating and modifying CSL styles can be found at `http://citationstyles.org/downloads/primer.html`. A repository of CSL styles can be found at `https://github.com/citation-style-language/styles`.

Citations go inside square brackets and are separated by semicolons. Each citation must have a key, composed of '@' + the citation identifier from the database, and may optionally have a prefix, a locator, and a suffix. Here are some examples:

```
Blah blah [see @doe99, pp. 33-35; also @smith04, ch. 1].

Blah blah [@doe99, pp. 33-35, 38-39 and *passim*].

Blah blah [@smith04; @doe99].
```

A minus sign (`-`) before the `@` will suppress mention of the author in the citation. This can be useful when the author is already mentioned in the text:

```
Smith says blah [-@smith04].
```

You can also write an in-text citation, as follows:

```
@smith04 says blah.

@smith04 [p. 33] says blah.
```

If the style calls for a list of works cited, it will be placed at the end of the document. Normally, you will want to end your document with an appropriate header:

```
last paragraph...

# References
```

The bibliography will be inserted after this header.

NOTES

[1]

The point of this rule is to ensure that normal paragraphs starting with people's initials, like

```
B. Russell was an English philosopher.
```

do not get treated as list items.

This rule will not prevent

```
(C) 2007 Joe Smith
```

from being interpreted as a list item. In this case, a backslash escape can be used:

```
(C\) 2007 Joe Smith
```

[2]

I have also been influenced by the suggestions of David Wheeler.

[3]

This scheme is due to Michel Fortin, who proposed it on the Markdown discussion list.

[4]

This feature is not yet implemented for RTF, OpenDocument, or ODT. In those formats, you'll just get an image in a paragraph by itself, with no caption.

SEE ALSO

`pandoc` (1).