Pandoc’s Markdown

Modified excerpt from the November 22, 2019 version of the Pandoc Manual by John MacFarlane.

Paragraphs

A paragraph is one or more lines of text followed by one or more blank lines. 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.

Extension: `escaped_line_breaks`

A backslash followed by a newline is also a hard line break. Note: in multiline and grid table cells, this is the only way to create a hard line break, since trailing spaces in the cells are ignored.

Headings

There are two kinds of headings: Setext and ATX.

Setext-style headings

A setext-style heading is a line of text “underlined” with a row of `=` signs (for a level-one heading) or `-` signs (for a level-two heading):

```
A level-one heading
===================

A level-two heading
-------------------
```

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

ATX-style headings

An ATX-style heading 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 heading level:

```
## A level-two heading

### A level-three heading ###
```

As with setext-style headings, the heading text can contain formatting:

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

Extension: `blank_before_header`

Standard Markdown syntax does not require a blank line before a heading. 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.
```

Extension: `space_in_atx_header`

Many Markdown implementations do not require a space between the opening #s of an ATX heading and the heading text, so that `#5 bolt` and `#hashtag` count as headings. With this extension, pandoc does require the space.

Heading identifiers

See also the `auto_identifiers` extension.

Extension: `header_attributes`

Headings can be assigned attributes using this syntax at the end of the line containing the heading text:

```
{#identifier .class .class key=value key=value}
```

Thus, for example, the following headings will all be assigned the identifier `foo`:

```
# My heading {#foo}

## My heading ##    {#foo}
```

```
My other heading   {#foo}
---------------
```

(This syntax is compatible with PHP Markdown Extra.)

Note that although this syntax allows assignment of classes and key/value attributes, writers generally don't use all of this information. Identifiers, classes, and key/value attributes are used in HTML and HTML-based formats such as EPUB and slidy. Identifiers are used for labels and link anchors in the LaTeX, ConTeXt, Textile, Jira markup, and AsciiDoc writers.

Headings with the class `unnumbered` will not be numbered, even if `--number-sections` is specified. A single hyphen (`-`) in an attribute context is equivalent to `.unnumbered`, and preferable in non-English documents. So,

```
# My heading {-}
```

is just the same as

```
# My heading {.unnumbered}
```

If the `unlisted` class is present in addition to `unnumbered`, the heading will not be included in a table of contents. (Currently this feature is only implemented for certain formats: those based on LaTeX and HTML, PowerPoint, and RTF.)

Extension: `implicit_header_references`

Pandoc behaves as if reference links have been defined for each heading. So, to link to a heading

```
# Heading identifiers in HTML
```

you can simply write

```
[Heading identifiers in HTML]
```

or

```
[Heading identifiers in HTML][]
```

or

```
[the section on heading identifiers][heading identifiers in
HTML]
```

instead of giving the identifier explicitly:

```
[Heading identifiers in HTML](#heading-identifiers-in-html)
```

If there are multiple headings with identical text, the corresponding reference will link to the first one only, and you will need to use explicit links to link to the others, as described above.

Like regular reference links, these references are case-insensitive.

Explicit link reference definitions always take priority over implicit heading references. So, in the following example, the link will point to `bar`, not to `#foo`:

```
# Foo

[foo]: bar

See [foo]
```

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 headings), with each line preceded by a > character and an optional 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.
```

If the > character is followed by an optional space, that space will be considered part of the block quote marker and not part of the indentation of the contents. Thus, to put an indented code block in a block quote, you need five spaces after the >:

```
>     code
```

Extension: `blank_before_blockquote`

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 the `markdown_strict` format 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.

Fenced code blocks

Extension: `fenced_code_blocks`

In addition to standard indented code blocks, pandoc supports *fenced* 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 these lines is treated as code. No indentation is necessary:

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

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

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

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

Extension: `backtick_code_blocks`

Same as `fenced_code_blocks`, but uses backticks (`) instead of tildes (~).

Extension: `fenced_code_attributes`

Optionally, you may attach attributes to fenced or backtick code block using this syntax:

```
~~~~ {#mycode .haskell .numberLines startFrom="100"}
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
               qsort (filter (>= x) xs)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Here `mycode` is an identifier, `haskell` and `numberLines` are classes, and `startFrom` is an attribute with value `100`. Some output formats can use this information to do syntax highlighting. Currently, the only output formats that uses this information are HTML, LaTeX, Docx, Ms, and PowerPoint. If highlighting is supported for your output format and language, then the code block above will appear highlighted, with numbered lines. (To see which languages are supported, type `pandoc --list-highlight-languages`.) Otherwise, the code block above will appear as follows:

```
<pre id="mycode" class="haskell numberLines" startFrom="100">
  <code>
  ...
  </code>
</pre>
```

The `numberLines` (or `number-lines`) class will cause the lines of the code block to be numbered, starting with `1` or the value of the `startFrom` attribute. The `lineAnchors` (or `line-anchors`) class will cause the lines to be clickable anchors in HTML output.

A shortcut form can also be used for specifying the language of the code block:

~~~
```haskell
qsort [] = []
```
~~~

This is equivalent to:

~~~
``` {.haskell}
qsort [] = []
```
~~~

If the `fenced_code_attributes` extension is disabled, but input contains class attribute(s) for the code block, the first class attribute will be printed after the opening fence as a bare word.

To prevent all highlighting, use the `--no-highlight` flag. To set the highlighting style, use `--highlight-style`. For more information on highlighting, see Syntax highlighting.

Line blocks

Extension: `line_blocks`

A line block is a sequence of lines beginning with a vertical bar (`|`) followed by a space. The division into lines will be preserved in the output, as will any leading spaces; otherwise, the lines will be formatted as Markdown. This is useful for verse and addresses:

```
| The limerick packs laughs anatomical
| In space that is quite economical.
|    But the good ones I've seen
|    So seldom are clean
| And the clean ones so seldom are comical

| 200 Main St.
| Berkeley, CA 94718
```

The lines can be hard-wrapped if needed, but the continuation line must begin with a space.

```
| The Right Honorable Most Venerable and Righteous Samuel L.
  Constable, Jr.
| 200 Main St.
| Berkeley, CA 94718
```

This syntax is borrowed from [reStructuredText].

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.
```

Block content in list items

A list item may contain multiple paragraphs and other block-level content. However, subsequent paragraphs must be preceded by a blank line and indented to line up with the first non-space content after the list marker.

```
  * First paragraph.

    Continued.

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

Exception: if the list marker is followed by an indented code block, which must begin 5 spaces after the list marker, then subsequent paragraphs must begin two columns after the last character of the list marker:

```
*     code

  continuation paragraph
```

List items may include other lists. In this case the preceding blank line is optional. The nested list must be indented to line up with the first non-space character after the list marker of the containing list item.

```
* fruits
  + apples
    - macintosh
    - red delicious
  + pears
  + peaches
* vegetables
  + broccoli
  + 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.
```

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
```

Extension: `fancy_lists`

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.

The `fancy_lists` extension also allows '`#`' to be used as an ordered list marker in place of a numeral:

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

Extension: `startnum`

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
```

Pandoc will start a new list each time a different type of list marker is used. So, the following will create three lists:

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

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

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

Extension: `task_lists`

Pandoc supports task lists, using the syntax of GitHub-Flavored Markdown.

```
- [ ] an unchecked task list item
- [x] checked item
```

Definition lists

Extension: `definition_lists`

Pandoc supports definition lists, using the syntax of PHP Markdown Extra with some extensions.

```
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. The body of the definition (including the first line, aside from the colon or tilde) should be indented four spaces. However, as with other Markdown lists, you can "lazily" omit indentation except at the beginning of a paragraph or other block element:

```
Term 1

:   Definition
with lazy continuation.

    Second paragraph of the definition.
```

If you leave space before the definition (as in the example above), the text of the definition will be treated as a paragraph. In some output formats, this will mean greater spacing between term/definition pairs. For a more compact definition list, omit the space before the definition:

```
Term 1
  ~ Definition 1

Term 2
  ~ Definition 2a
  ~ Definition 2b
```

Note that space between items in a definition list is required. (A variant that loosens this requirement, but disallows "lazy" hard wrapping, can be activated with `compact_definition_lists`: see Non-pandoc extensions, below.)

Numbered example lists

Extension: `example_lists`

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.

Note: continuation paragraphs in example lists must always be indented four spaces, regardless of the length of the list marker. That is, example lists always behave as if the `four_space_rule` extension is set. This is because example labels tend to be long, and indenting content to the first non-space character after the label would be awkward.

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 `markdown_strict` format 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

<!-- -->

1.  uno
2.  dos
3.  tres
```

Horizontal rules

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

```
*  *  *  *

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

Tables

Four kinds of tables may be used. The first three kinds presuppose the use of a fixed-width font, such as Courier. The fourth kind can be used with proportionally spaced fonts, as it does not require lining up columns.

Extension: `table_captions`

A caption may optionally be provided with all 4 kinds of tables (as illustrated in the examples below). 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.

Extension: `simple_tables`

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 header 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:

- 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.

The column header row 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 the header row is 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.

Extension: `multiline_tables`

Multiline tables allow header 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 header row is 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.

The header 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 a header.
```

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.

Extension: `grid_tables`

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.). Cells that span multiple columns or rows are not supported. Grid tables can be created easily using Emacs' table-mode (`M-x table-insert`).

Alignments can be specified as with pipe tables, by putting colons at the boundaries of the separator line after the header:

```
+---------------+---------------+--------------------+
| Right         | Left          | Centered           |
+==============:+:==============+:==================:+
| Bananas       | $1.34         | built-in wrapper   |
+---------------+---------------+--------------------+
```

For headerless tables, the colons go on the top line instead:

```
+--------------:+:--------------+:------------------:+
| Right         | Left          | Centered           |
+---------------+---------------+--------------------+
```

Grid Table Limitations Pandoc does not support grid tables with row spans or column spans. This means that neither variable numbers of columns across rows nor variable numbers of rows across columns are supported by Pandoc. All grid tables must have the same number of columns in each row, and the same number of rows in each column. For example, the Docutils sample grid tables will not render as expected with Pandoc.

Extension: `pipe_tables`

Pipe tables look like this:

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

  : Demonstration of pipe table syntax.
```

The syntax is identical to PHP Markdown Extra tables. The beginning and ending pipe characters are optional, but pipes are required between all columns. The colons indicate column alignment as shown. The header cannot be omitted. To simulate a headerless table, include a header with blank cells.

Since the pipes indicate column boundaries, columns need not be vertically aligned, as they are in the above example. So, this is a perfectly legal (though ugly) pipe table:

```
fruit| price
-----|-----:
```

```
apple|2.05
pear|1.37
orange|3.09
```

The cells of pipe tables cannot contain block elements like paragraphs and lists, and cannot span multiple lines. If a pipe table contains a row whose printable content is wider than the column width (see `--columns`), then the table will take up the full text width and the cell contents will wrap, with the relative cell widths determined by the number of dashes in the line separating the table header from the table body. (For example `---|-` would make the first column 3/4 and the second column 1/4 of the full text width.) On the other hand, if no lines are wider than column width, then cell contents will not be wrapped, and the cells will be sized to their contents.

Note: pandoc also recognizes pipe tables of the following form, as can be produced by Emacs' orgtbl-mode:

```
| One | Two   |
|-----+-------|
| my  | table |
| is  | nice  |
```

The difference is that `+` is used instead of `|`. Other orgtbl features are not supported. In particular, to get non-default column alignment, you'll need to add colons as above.

Metadata blocks

Extension: `pandoc_title_block`

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.

Extension: `yaml_metadata_block`

A YAML metadata block is a valid YAML object, delimited by a line of three hyphens (`---`) at the top and a line of three hyphens (`---`) or three dots (`...`) at the bottom. A YAML metadata block may occur anywhere in the document, but if it is not at the beginning, it must be preceded by a blank line. (Note that, because of the way pandoc concatenates input files when several are provided, you may also keep the metadata in a separate YAML file and pass it to pandoc as an argument, along with your Markdown files:

```
pandoc chap1.md chap2.md chap3.md metadata.yaml -s -o book.html
```

Just be sure that the YAML file begins with `---` and ends with `---` or `...`.) Alternatively, you can use the `--metadata-file` option. Using that approach however, you cannot reference content (like footnotes) from the main markdown input document.

Metadata will be taken from the fields of the YAML object and added to any existing document metadata. Metadata can contain lists and objects (nested arbitrarily), but all string scalars will be interpreted as Markdown. Fields with names ending in an underscore will be ignored by pandoc. (They may be given a role by external processors.) Field names must not be interpretable as YAML numbers or boolean values (so, for example, `yes`, `True`, and `15` cannot be used as field names).

A document may contain multiple metadata blocks. If two metadata blocks attempt to set the same field, the value from the second block will be taken.

When pandoc is used with `-t markdown` to create a Markdown document, a YAML metadata block will be produced only if the `-s/--standalone` option is used. All of the metadata will appear in a single block at the beginning of the document.

Note that YAML escaping rules must be followed. Thus, for example, if a title contains a colon, it must be quoted. The pipe character (`|`) can be used to begin an indented block that will be interpreted literally, without need for escaping. This form is necessary when the field contains blank lines or block-level formatting:

```
---
title:  'This is the title: it contains a colon'
author:
- Author One
- Author Two
keywords: [nothing, nothingness]
abstract: |
  This is the abstract.

  It consists of two paragraphs.
...
```

Template variables will be set automatically from the metadata. Thus, for example, in writing HTML, the variable `abstract` will be set to the HTML equivalent of the Markdown in the `abstract` field:

```
<p>This is the abstract.</p>
<p>It consists of two paragraphs.</p>
```

Variables can contain arbitrary YAML structures, but the template must match this structure. The `author` variable in the default templates expects a simple list or string, but can be changed to support more complicated structures. The following combination, for example, would add an affiliation to the author if one is given:

```
---
title: The document title
author:
- name: Author One
  affiliation: University of Somewhere
- name: Author Two
  affiliation: University of Nowhere
...
```

To use the structured authors in the example above, you would need a custom template:

```
$for(author)$
$if(author.name)$
$author.name$$if(author.affiliation)$ ($author.affiliation$)$endif$
$else$
$author$
$endif$
$endfor$
```

Raw content to include in the document's header may be specified using `header-includes`; however, it is important to mark up this content as raw code for a particular output format, using the `raw_attribute` extension), or it will be interpreted as markdown. For example:

~~~
header-includes:
- |
  ```{=latex}
  \let\oldsection\section
  \renewcommand{\section}[1]{\clearpage\oldsection{#1}}
  ```
~~~

Backslash escapes

Extension: `all_symbols_escapable`

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 `markdown_strict` format is used, 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.

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\*.
```

Extension: `intraword_underscores`

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

Extension: `strikeout`

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

Extension: `superscript`, `subscript`

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.
```

The text between `^...^` or `~...~` may not contain spaces or newlines. 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 `^`, and also bad interactions with footnotes.) 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: `\*`.
```

Extension: `inline_code_attributes`

Attributes can be attached to verbatim text, just as with fenced code blocks:

```
`<$>`{.haskell}
```

Small caps

To write small caps, use the `smallcaps` class:

```
[Small caps]{.smallcaps}
```

Or, without the `bracketed_spans` extension:

```
<span class="smallcaps">Small caps</span>
```

For compatibility with other Markdown flavors, CSS is also supported:

```
<span style="font-variant:small-caps;">Small caps</span>
```

This will work in all output formats that support small caps.

Math

Extension: `tex_math_dollars`

Anything between two $ characters will be treated as TeX math. The opening $ must have a non-space character immediately to its right, while the closing $ must have a non-space character immediately to its left, and must not be followed immediately by a digit. 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:

LaTeX It will appear verbatim surrounded by `\(...\)` (for inline math) or `\[...\]` (for display math).

Markdown, Emacs Org mode, ConTeXt, ZimWiki It will appear verbatim surrounded by `$...$` (for inline math) or `$$...$$` (for display math).

XWiki It will appear verbatim surrounded by `{{formula}}..{{/formula}}`.

reStructuredText It will be rendered using an interpreted text role `:math:`.

AsciiDoc For AsciiDoc output format (`-t asciidoc`) it will appear verbatim surrounded by `latexmath:[$...$]` (for inline math) or `[latexmath]++++\[...\]+++` (for display math). For AsciiDoctor output format (`-t asciidoctor`) the LaTex delimiters (`$..$` and `\[..\]`) are omitted.

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

roff man, Jira markup It will be rendered verbatim without $'s.

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

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

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

ODT It will be rendered, if possible, using MathML.

DocBook If the `--mathml` flag is used, it will be rendered using MathML in an `inlineequation` or `informalequation` tag. Otherwise it will be rendered, if possible, using Unicode characters.

Docx It will be rendered using OMML math markup.

FictionBook2 If the `--webtex` option is used, formulas are rendered as images using CodeCogs or other compatible web service, downloaded and embedded in the e-book. Otherwise, they will appear verbatim.

HTML, Slidy, DZSlides, S5, EPUB The way math is rendered in HTML will depend on the command-line options selected. Therefore see Math rendering in HTML.

Raw HTML

Extension: `raw_html`

Markdown allows you to insert raw HTML (or DocBook) anywhere in a document (except verbatim contexts, where `<`, `>`, and `&` are interpreted literally). (Technically this is not an extension, since standard Markdown allows it, but it has been made an extension so that it can be disabled if desired.)

The raw HTML is passed through unchanged in HTML, S5, Slidy, Slideous, DZSlides, EPUB, Markdown, CommonMark, Emacs Org mode, and Textile output, and suppressed in other formats.

For a more explicit way of including raw HTML in a Markdown document, see the `raw_attribute` extension.

In the CommonMark format, if `raw_html` is enabled, superscripts, subscripts, strikeouts and small capitals will be represented as HTML. Otherwise, plain-text fallbacks will be used. Note that even if `raw_html` is disabled, tables will be rendered with HTML syntax if they cannot use pipe syntax.

Extension: `markdown_in_html_blocks`

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 the `markdown_strict` format is used; 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.

Extension: `native_divs`

Use native pandoc `Div` blocks for content inside `<div>` tags. For the most part this should give the same output as `markdown_in_html_blocks`, but it makes it easier to write pandoc filters to manipulate groups of blocks.

Extension: `native_spans`

Use native pandoc `Span` blocks for content inside `<span>` tags. For the most part this should give the same output as `raw_html`, but it makes it easier to write pandoc filters to manipulate groups of inlines.

Extension: `raw_tex`

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.

For a more explicit and flexible way of including raw TeX in a Markdown document, see the `raw_attribute` extension.

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

Generic raw attribute

Extension: `raw_attribute`

Inline spans and fenced code blocks with a special kind of attribute will be parsed as raw content with the designated format. For example, the following produces a raw roff `ms` block:

~~~
```{=ms}
.MYMACRO
blah blah
```
~~~

And the following produces a raw `html` inline element:

```
This is `<a>html</a>`{=html}
```

This can be useful to insert raw xml into `docx` documents, e.g. a pagebreak:

~~~
```{=openxml}
<w:p>
  <w:r>
    <w:br w:type="page"/>
  </w:r>
</w:p>
```
~~~

The format name should match the target format name (see `-t/--to`, above, for a list, or use `pandoc --list-output-formats`). Use `openxml` for `docx` output, `opendocument` for `odt` output, `html5` for `epub3` output, `html4` for `epub2` output, and `latex`, `beamer`, `ms`, or `html5` for `pdf` output (depending on what you use for `--pdf-engine`).

This extension presupposes that the relevant kind of inline code or fenced code block is enabled. Thus, for example, to use a raw attribute with a backtick code block, `backtick_code_blocks` must be enabled.

The raw attribute cannot be combined with regular attributes.

LaTeX macros

Extension: `latex_macros`

When this extension is enabled, pandoc will parse LaTeX macro definitions and apply the resulting macros to all LaTeX math and raw LaTeX. So, for example, the following will work in all output formats, not just LaTeX:

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

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

Note that LaTeX macros will not be applied if they occur inside a raw span or block marked with the `raw_attribute` extension.

When `latex_macros` is disabled, the raw LaTeX and math will not have macros applied. This is usually a better approach when you are targeting LaTeX or PDF.

Macro definitions in LaTeX will be passed through as raw LaTeX only if `latex_macros` is not enabled. Macro definitions in Markdown source (or other formats allowing `raw_tex`) will be passed through regardless of whether `latex_macros` is enabled.

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.

Email addresses in inline links are not autodetected, so they have to be prefixed with `mailto`:

```
[Write me!](mailto:sam@green.eggs.ham)
```

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 cannot be space between the two unless the `spaced_reference_links` extension is enabled.) The link definition 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. The label must not be parseable as a citation (assuming the `citations` extension is enabled): citations take precedence over link labels.

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:

```
See [my website][].

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

Note: In `Markdown.pl` and most other Markdown implementations, reference link definitions cannot occur in nested constructions such as list items or block quotes. Pandoc lifts this arbitrary seeming restriction. So the following is fine in pandoc, though not in most other implementations:

```
> My block [quote].
>
> [quote]: /foo
```

Extension: `shortcut_reference_links`

In a *shortcut* reference link, the second pair of brackets may be omitted entirely:

```
See [my website].

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

Internal links

To link to another section of the same document, use the automatically generated identifier (see Heading identifiers). For example:

```
See the [Introduction](#introduction).
```

or

```
See the [Introduction].

[Introduction]: #introduction
```

Internal links are currently supported for HTML formats (including HTML slide shows and EPUB), LaTeX, and ConTeXt.

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
```

Extension: `implicit_figures`

An image with nonempty alt text, occurring by itself in a paragraph, will be rendered as a figure with a caption. The image's alt text will be used as the caption.

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

How this is rendered depends on the output format. Some output formats (e.g. RTF) do not yet support figures. In those formats, you'll just get an image in a paragraph by itself, with no caption.

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)\
```

Note that in reveal.js slide shows, an image in a paragraph by itself that has the `stretch` class will fill the screen, and the caption and figure tags will be omitted.

Extension: `link_attributes`

Attributes can be set on links and images:

```
An inline ![image](foo.jpg){#id .class width=30 height=20px}
and a reference ![image][ref] with attributes.

[ref]: foo.jpg "optional title" {#id .class key=val key2="val 2"}
```

(This syntax is compatible with PHP Markdown Extra when only `#id` and `.class` are used.)

For HTML and EPUB, all known HTML5 attributes except `width` and `height` (but including `srcset` and `sizes`) are passed through as is. Unknown attributes are passed through as custom attributes, with `data-` prepended. The other writers ignore attributes that are not specifically supported by their output format.

The `width` and `height` attributes on images are treated specially. When used without a unit, the unit is assumed to be pixels. However, any of the following unit identifiers can be used: `px`, `cm`, `mm`, `in`, `inch` and `%`. There must not be any spaces between the number and the unit. For example:

```
![](file.jpg){ width=50% }
```

- Dimensions are converted to inches for output in page-based formats like LaTeX. Dimensions are converted to pixels for output in HTML-like formats. Use the `--dpi` option to specify the number of pixels per inch. The default is 96dpi.
- The `%` unit is generally relative to some available space. For example the above example will render to the following.
  - HTML: `<img href="file.jpg" style="width: 50%;" />`
  - LaTeX: `\includegraphics[width=0.5\textwidth,height=\textheight]{file`
    (If you're using a custom template, you need to configure `graphicx` as in the default template.)
  - ConTeXt: `\externalfigure[file.jpg][width=0.5\textwidth]`
- Some output formats have a notion of a class (ConTeXt) or a unique identifier (LaTeX `\caption`), or both (HTML).
- When no `width` or `height` attributes are specified, the fallback is to look at the image resolution and the dpi metadata embedded in the image file.

Divs and Spans

Using the `native_divs` and `native_spans` extensions (see above), HTML syntax can be used as part of markdown to create native `Div` and `Span` elements in the pandoc AST (as opposed to raw HTML). However, there is also nicer syntax available:

Extension: `fenced_divs`

Allow special fenced syntax for native `Div` blocks. A Div starts with a fence containing at least three consecutive colons plus some attributes. The attributes may optionally be followed by another string of consecutive colons. The attribute syntax is exactly as in fenced code blocks (see Extension: `fenced_code_attributes`). As with fenced code blocks, one can use either attributes in curly braces or a single unbraced word, which will be treated as a class name. The Div ends with another line containing a string of at least three consecutive colons. The fenced Div should be separated by blank lines from preceding and following blocks.

Example:

```
::::: {#special .sidebar}
```

```
Here is a paragraph.

And another.
:::::
```

Fenced divs can be nested. Opening fences are distinguished because they *must* have attributes:

```
::: Warning ::::::
This is a warning.

::: Danger
This is a warning within a warning.
:::
::::::::::::::::::
```

Fences without attributes are always closing fences. Unlike with fenced code blocks, the number of colons in the closing fence need not match the number in the opening fence. However, it can be helpful for visual clarity to use fences of different lengths to distinguish nested divs from their parents.

Extension: `bracketed_spans`

A bracketed sequence of inlines, as one would use to begin a link, will be treated as a `Span` with attributes if it is followed immediately by attributes:

```
[This is *some text*]{.class key="val"}
```

Footnotes

Extension: `footnotes`

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.). Each footnote should be separated from surrounding content (including other footnotes) by blank lines.

Extension: `inline_notes`

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

Extension: `citations`

Using an external filter, `pandoc-citeproc`, pandoc can automatically generate citations and a bibliography in a number of styles. Basic usage is

```
pandoc --filter pandoc-citeproc myinput.txt
```

In order to use this feature, you will need to specify a bibliography file using the `bibliography` metadata field in a YAML metadata section, or `--bibliography` command line argument. You can supply multiple `--bibliography` arguments or set `bibliography` metadata field to YAML array, if you want to use multiple bibliography files. The bibliography may have any of these formats:

Format	File extension
BibLaTeX	.bib
BibTeX	.bibtex

Format	File extension
Copac	.copac
CSL JSON	.json
CSL YAML	.yaml
EndNote	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
MODS	.mods
RIS	.ris

Note that `.bib` can be used with both BibTeX and BibLaTeX files; use `.bibtex` to force BibTeX.

Note that `pandoc-citeproc --bib2json` and `pandoc-citeproc --bib2yaml` can produce `.json` and `.yaml` files from any of the supported formats.

In-field markup: In BibTeX and BibLaTeX databases, pandoc-citeproc parses a subset of LaTeX markup; in CSL YAML databases, pandoc Markdown; and in CSL JSON databases, an HTML-like markup:

`<i>...</i>` italics
`<b>...</b>` bold
`<span style="font-variant:small-caps;">...</span>` or `<sc>...</sc>` small capitals
`<sub>...</sub>` subscript
`<sup>...</sup>` superscript
`<span class="nocase">...</span>` prevent a phrase from being capitalized as title case

`pandoc-citeproc -j` and `-y` interconvert the CSL JSON and CSL YAML formats as far as possible.

As an alternative to specifying a bibliography file using `--bibliography` or the YAML metadata field `bibliography`, you can include the citation data directly in the `references` field of the document's YAML metadata. The field should contain an array of YAML-encoded references, for example:

```
---
references:
- type: article-journal
  id: WatsonCrick1953
  author:
  - family: Watson
    given: J. D.
  - family: Crick
    given: F. H. C.
  issued:
    date-parts:
    - - 1953
      - 4
      - 25
  title: 'Molecular structure of nucleic acids: a structure for deoxyribose
    nucleic acid'
  title-short: Molecular structure of nucleic acids
  container-title: Nature
  volume: 171
  issue: 4356
  page: 737-738
  DOI: 10.1038/171737a0
  URL: http://www.nature.com/nature/journal/v171/n4356/abs/171737a0.html
  language: en-GB
...
```

(`pandoc-citeproc --bib2yaml` can produce these from a bibliography file in one of the supported formats.)

Citations and references can be formatted using any style supported by the Citation Style Language, listed in the Zotero Style Repository. These files are specified using the `--csl` option or the `csl` metadata field. By default, `pandoc-citeproc` will use the Chicago Manual of Style author-date format. The CSL project provides further information on finding and editing styles.

To make your citations hyperlinks to the corresponding bibliography entries, add `link-citations: true` to your YAML metadata.

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. The citation key must begin with a letter, digit, or _, and may contain alphanumerics, _, and internal punctuation characters (`:.#$%&-+?<>~/`). Here are some examples:

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

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

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

`pandoc-citeproc` detects locator terms in the CSL locale files. Either abbreviated or unabbreviated forms are accepted. In the `en-US` locale, locator terms can be written in either singular or plural forms, as `book`, `bk.`/`bks.`; `chapter`, `chap.`/`chaps.`; `column`, col./cols.; figure, fig./figs.; folio, fol./fols.; number, no./nos.; line, l./ll.; note, n./nn.; opus, op./opp.; page, p./pp.; paragraph, para./paras.; part, pt./pts.; section, sec./secs.; sub verbo, s.v./s.vv.; verse, v./vv.; volume, vol./vols.; ¶/¶¶; §/§§. If no locator term is used, "page" is assumed.

pandoc-citeproc will use heuristics to distinguish the locator from the suffix. In complex cases, the locator can be enclosed in curly braces (using pandoc-citeproc 0.15 and higher only):

```
[@smith{ii, A, D-Z}, with a suffix]
[@smith, {pp. iv, vi-xi, (xv)-(xvii)} with suffix here]
```

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 in a div with id refs, if one exists:

```
::: {#refs}
:::
```

Otherwise, it will be placed at the end of the document. Generation of the bibliography can be suppressed by setting suppress-bibliography: true in the YAML metadata.

If you wish the bibliography to have a section heading, you can set reference-section-title in the metadata, or put the heading at the beginning of the div with id refs (if you are using it) or at the end of your document:

```
last paragraph...

# References
```

The bibliography will be inserted after this heading. Note that the unnumbered class will be added to this heading, so that the section will not be numbered.

If you want to include items in the bibliography without actually citing them in the body text, you can define a dummy nocite metadata field and put the citations there:

```
---
nocite: |
  @item1, @item2
...
```

```
@item3
```

In this example, the document will contain a citation for `item3` only, but the bibliography will contain entries for `item1`, `item2`, and `item3`.

It is possible to create a bibliography with all the citations, whether or not they appear in the document, by using a wildcard:

```
---
nocite: |
  @*
...
```

For LaTeX output, you can also use `natbib` or `biblatex` to render the bibliography. In order to do so, specify bibliography files as outlined above, and add `--natbib` or `--biblatex` argument to `pandoc` invocation. Bear in mind that bibliography files have to be in respective format (either BibTeX or BibLaTeX).

For more information, see the pandoc-citeproc man page.

Non-pandoc extensions

The following Markdown syntax extensions are not enabled by default in pandoc, but may be enabled by adding `+EXTENSION` to the format name, where `EXTENSION` is the name of the extension. Thus, for example, `markdown+hard_line_breaks` is Markdown with hard line breaks.

Extension: `old_dashes`

Selects the pandoc <= 1.8.2.1 behavior for parsing smart dashes: `-` before a numeral is an en-dash, and `--` is an em-dash. This option only has an effect if `smart` is enabled. It is selected automatically for `textile` input.

Extension: `angle_brackets_escapable`

Allow < and > to be backslash-escaped, as they can be in GitHub flavored Markdown but not original Markdown. This is implied by pandoc's default `all_symbols_escapable`.

Extension: `lists_without_preceding_blankline`

Allow a list to occur right after a paragraph, with no intervening blank space.

Extension: `four_space_rule`

Selects the pandoc <= 2.0 behavior for parsing lists, so that four spaces indent are needed for list item continuation paragraphs.

Extension: `spaced_reference_links`

Allow whitespace between the two components of a reference link, for example,

`[foo] [bar]`.

Extension: `hard_line_breaks`

Causes all newlines within a paragraph to be interpreted as hard line breaks instead of spaces.

Extension: `ignore_line_breaks`

Causes newlines within a paragraph to be ignored, rather than being treated as spaces or as hard line breaks. This option is intended for use with East Asian languages where spaces are not used between words, but text is divided into lines for readability.

Extension: `east_asian_line_breaks`

Causes newlines within a paragraph to be ignored, rather than being treated as spaces or as hard line breaks, when they occur between two East Asian wide characters. This is a better choice than `ignore_line_breaks` for texts that include a mix of East Asian wide characters and other characters.

Extension: `emoji`

Parses textual emojis like `:smile:` as Unicode emoticons.

Extension: `tex_math_single_backslash`

Causes anything between `\(` and `\)` to be interpreted as inline TeX math, and anything between `\[` and `\]` to be interpreted as display TeX math. Note: a drawback of this extension is that it precludes escaping `(` and `[`.

Extension: `tex_math_double_backslash`

Causes anything between `\\(` and `\\)` to be interpreted as inline TeX math, and anything between `\\[` and `\\]` to be interpreted as display TeX math.

Extension: `markdown_attribute`

By default, pandoc interprets material inside block-level tags as Markdown. This extension changes the behavior so that Markdown is only parsed inside block-level tags if the tags have the attribute `markdown=1`.

Extension: `mmd_title_block`

Enables a [MultiMarkdown] style title block at the top of the document, for example:

```
Title:   My title
Author:  John Doe
Date:    September 1, 2008
Comment: This is a sample mmd title block, with
         a field spanning multiple lines.
```

See the MultiMarkdown documentation for details. If `pandoc_title_block` or `yaml_metadata_block` is enabled, it will take precedence over `mmd_title_block`.

Extension: `abbreviations`

Parses PHP Markdown Extra abbreviation keys, like

```
*[HTML]: Hypertext Markup Language
```

Note that the pandoc document model does not support abbreviations, so if this extension is enabled, abbreviation keys are simply skipped (as opposed to being parsed as paragraphs).

Extension: `autolink_bare_uris`

Makes all absolute URIs into links, even when not surrounded by pointy braces `<...>`.

Extension: `mmd_link_attributes`

Parses multimarkdown style key-value attributes on link and image references. This extension should not be confused with the `link_attributes` extension.

```
This is a reference ![image][ref] with multimarkdown attributes.

[ref]: http://path.to/image "Image title" width=20px height=30px
       id=myId class="myClass1 myClass2"
```

Extension: `mmd_header_identifiers`

Parses multimarkdown style heading identifiers (in square brackets, after the heading but before any trailing #s in an ATX heading).

Extension: `compact_definition_lists`

Activates the definition list syntax of pandoc 1.12.x and earlier. This syntax differs from the one described above under Definition lists in several respects:

- No blank line is required between consecutive items of the definition list.
- To get a "tight" or "compact" list, omit space between consecutive items; the space between a term and its definition does not affect anything.
- Lazy wrapping of paragraphs is not allowed: the entire definition must be indented four spaces.

Extension: `gutenberg`

Use Project Gutenberg conventions for `plain` output: all-caps for strong emphasis, surround by underscores for regular emphasis, add extra blank space around headings.

Markdown variants

In addition to pandoc's extended Markdown, the following Markdown variants are supported:

`markdown_phpextra` (PHP Markdown Extra) `footnotes`, `pipe_tables`, `raw_html`, `markdown_attribute`, `fenced_code_blocks`, `definition_lists`, `intraword_underscores`, `header_attributes`, `link_attributes`, `abbreviations`, `shortcut_reference_links`, `spaced_reference_links`.

`markdown_github` (deprecated GitHub-Flavored Markdown) `pipe_tables`, `raw_html`, `fenced_code_blocks`, `auto_identifiers`, `gfm_auto_identifiers`, `backtick_code_blocks`, `autolink_bare_uris`, `space_in_atx_header`, `intraword_underscores`, `strikeout`, `task_lists`, `emoji`, `shortcut_reference_links`, `angle_brackets_escapable`, `lists_without_preceding_blankline`.

`markdown_mmd` (MultiMarkdown) `pipe_tables`, `raw_html`, `markdown_attribute`, `mmd_link_attributes`, `tex_math_double_backslash`, `intraword_underscores`, `mmd_title_block`, `footnotes`, `definition_lists`, `all_symbols_escapable`, `implicit_header_references`, `auto_identifiers`, `mmd_header_identifiers`, `shortcut_reference_links`, `implicit_figures`, `superscript`, `subscript`, `backtick_code_blocks`, `spaced_reference_links`, `raw_attribute`.

`markdown_strict` (Markdown.pl) `raw_html`, `shortcut_reference_links`, `spaced_refer`

We also support `commonmark` and `gfm` (GitHub-Flavored Markdown, which is implemented as a set of extensions on `commonmark`).

Note, however, that `commonmark` and `gfm` have limited support for extensions. Only those listed below (and `smart`, `raw_tex`, and `hard_line_breaks`) will work. The extensions can, however, all be individually disabled. Also, `raw_tex` only affects `gfm` output, not input.

`gfm` (**GitHub-Flavored Markdown**) `pipe_tables`, `raw_html`, `fenced_code_blocks`, `auto_identifiers`, `gfm_auto_identifiers`, `backtick_code_blocks`, `autolink_bare_uris`, `space_in_atx_header`, `intraword_underscores`, `strikeout`, `task_lists`, `emoji`, `shortcut_reference_links`, `angle_brackets_escapable`, `lists_without_preceding_blankline`.