## Contents

### Synopsis

1

### Description

3

- Using pandoc .................................................. 3
- Specifying formats ......................................... 4
- Character encoding ........................................ 4
- Creating a PDF ................................................ 5
- Reading from the Web ...................................... 5

### Options

7

- General options ............................................. 7
- Reader options ............................................... 10
- General writer options ................................... 13
- Options affecting specific writers ...................... 15
- Citation rendering ......................................... 21
- Math rendering in HTML .................................. 22
- Options for wrapper scripts ............................ 23

### Templates

25

- Metadata variables ....................................... 25
- Language variables ....................................... 26
- Variables for HTML slides ................................ 27
- Variables for Beamer slides ............................. 27
- Variables for LaTeX ......................................... 28
  - Layout ....................................................... 28
  - Fonts ....................................................... 29
  - Links ....................................................... 30
  - Front matter ............................................. 30
  - BibLaTeX bibliographies ................................. 30
- Variables for ConTeXt .................................... 31
- Variables for wkhtmltopdf ................................. 32
- Variables for man pages ................................ 32
- Variables for ms ............................................ 32
- Structural variables ...................................... 32
- Using variables in templates ......................... 33

### Extensions

35

- Typography .................................................. 35
- Headings and sections .................................... 36
- Math Input ................................................... 37
- Raw HTML/TeX ................................................ 37
## Contents

- Literate Haskell support .................................................. 38
- Other extensions .............................................................. 39

### Pandoc’s Markdown 41
- Philosophy ................................................................. 41
- Paragraphs ................................................................. 41
- Headings ................................................................. 42
  - Setext-style headings ................................................ 42
  - ATX-style headings .................................................. 42
  - Heading identifiers .................................................. 43
- Block quotations ........................................................... 45
- Verbatim (code) blocks .................................................. 46
  - Indented code blocks ............................................... 46
  - Fenced code blocks .................................................. 47
- Line blocks ............................................................... 49
- Lists ........................................................................ 49
  - Bullet lists .......................................................... 49
  - Block content in list items ........................................ 50
  - Ordered lists ......................................................... 51
  - Definition lists ...................................................... 53
  - Numbered example lists ............................................ 54
  - Compact and loose lists ............................................ 55
  - Ending a list .......................................................... 56
- Horizontal rules .......................................................... 56
- Tables ................................................................. 57
- Metadata blocks ........................................................ 61
- Backslash escapes ....................................................... 65
- Inline formatting ........................................................ 66
  - Emphasis .............................................................. 66
  - Strikeout .............................................................. 66
  - Superscripts and subscripts ........................................ 67
  - Verbatim .............................................................. 67
  - Small caps ............................................................. 68
- Math ................................................................. 68
- Raw HTML ............................................................... 69
  - Generic raw attribute .............................................. 71
- LaTeX macros ............................................................ 72
- Links ................................................................. 73
  - Automatic links ...................................................... 73
  - Inline links .......................................................... 73
  - Reference links ....................................................... 73
  - Internal links ........................................................ 75
- Images ............................................................... 75
- Divs and Spans .......................................................... 77
- Footnotes ............................................................... 78
- Citations ............................................................... 79
- Non-pandoc extensions ................................................ 83
- Markdown variants ...................................................... 86
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Producing slide shows with pandoc</td>
<td>89</td>
</tr>
<tr>
<td>Structuring the slide show</td>
<td>90</td>
</tr>
<tr>
<td>Incremental lists</td>
<td>91</td>
</tr>
<tr>
<td>Inserting pauses</td>
<td>92</td>
</tr>
<tr>
<td>Styling the slides</td>
<td>92</td>
</tr>
<tr>
<td>Speaker notes</td>
<td>93</td>
</tr>
<tr>
<td>Columns</td>
<td>93</td>
</tr>
<tr>
<td>Frame attributes in beamer</td>
<td>94</td>
</tr>
<tr>
<td>Background in reveal.js and beamer</td>
<td>94</td>
</tr>
<tr>
<td>Creating EPUBs with pandoc</td>
<td>97</td>
</tr>
<tr>
<td>EPUB Metadata</td>
<td>97</td>
</tr>
<tr>
<td>The <code>epub:type</code> attribute</td>
<td>98</td>
</tr>
<tr>
<td>Linked media</td>
<td>99</td>
</tr>
<tr>
<td>Creating Jupyter notebooks with pandoc</td>
<td>101</td>
</tr>
<tr>
<td>Syntax highlighting</td>
<td>105</td>
</tr>
<tr>
<td>Custom Styles</td>
<td>107</td>
</tr>
<tr>
<td>Output</td>
<td>107</td>
</tr>
<tr>
<td>Input</td>
<td>108</td>
</tr>
<tr>
<td>Custom writers</td>
<td>111</td>
</tr>
<tr>
<td>A note on security</td>
<td>113</td>
</tr>
<tr>
<td>Authors</td>
<td>115</td>
</tr>
</tbody>
</table>
Synopsis

pandoc [options] [input-file]...
Description

Pandoc is a Haskell library for converting from one markup format to another, and a command-line tool that uses this library.

Pandoc can convert between numerous markup and word processing formats, including, but not limited to, various flavors of Markdown, HTML, LaTeX and Word docx. For the full lists of input and output formats, see the --from and --to options below. Pandoc can also produce PDF output: see creating a PDF, below.

Pandoc’s enhanced version of Markdown includes syntax for tables, definition lists, metadata blocks, footnotes, citations, math, and much more. See below under Pandoc’s Markdown.

Pandoc has a modular design: it consists of a set of readers, which parse text in a given format and produce a native representation of the document (an abstract syntax tree or AST), and a set of writers, which convert this native representation into a target format. Thus, adding an input or output format requires only adding a reader or writer. Users can also run custom pandoc filters to modify the intermediate AST.

Because pandoc’s intermediate representation of a document is less expressive than many of the formats it converts between, one should not expect perfect conversions between every format and every other. Pandoc attempts to preserve the structural elements of a document, but not formatting details such as margin size. And some document elements, such as complex tables, may not fit into pandoc’s simple document model. While conversions from pandoc’s Markdown to all formats aspire to be perfect, conversions from formats more expressive than pandoc’s Markdown can be expected to be lossy.

Using pandoc

If no input-files are specified, input is read from stdin. Output goes to stdout by default. For output to a file, use the -o option:

pandoc -o output.html input.txt

By default, pandoc produces a document fragment. To produce a standalone document (e.g. a valid HTML file including <head> and <body>), use the -s or --standalone flag:
Description

pandoc -s -o output.html input.txt

For more information on how standalone documents are produced, see Templates below.

If multiple input files are given, pandoc will concatenate them all (with blank lines between them) before parsing. (Use --file-scope to parse files individually.)

**Specifying formats**

The format of the input and output can be specified explicitly using command-line options. The input format can be specified using the -f/--from option, the output format using the -t/--to option. Thus, to convert hello.txt from Markdown to LaTeX, you could type:

pandoc -f markdown -t latex hello.txt

To convert hello.html from HTML to Markdown:

pandoc -f html -t markdown hello.html

Supported input and output formats are listed below under Options (see -f for input formats and -t for output formats). You can also use pandoc --list-input-formats and pandoc --list-output-formats to print lists of supported formats.

If the input or output format is not specified explicitly, pandoc will attempt to guess it from the extensions of the filenames. Thus, for example,

pandoc -o hello.tex hello.txt

will convert hello.txt from Markdown to LaTeX. If no output file is specified (so that output goes to stdout), or if the output file’s extension is unknown, the output format will default to HTML. If no input file is specified (so that input comes from stdin), or if the input files’ extensions are unknown, the input format will be assumed to be Markdown.

**Character encoding**

Pandoc uses the UTF-8 character encoding for both input and output. If your local character encoding is not UTF-8, you should pipe input and output through iconv:

iconv -t utf-8 input.txt | pandoc | iconv -f utf-8

Note that in some output formats (such as HTML, LaTeX, ConTeXt, RTF, OPML, DocBook, and Texinfo), information about the character encoding is included in the document header, which will only be included if you use the -s/--standalone option.
Creating a PDF

To produce a PDF, specify an output file with a .pdf extension:

```
pandoc test.txt -o test.pdf
```

By default, pandoc will use LaTeX to create the PDF, which requires that a LaTeX engine be installed (see --pdf-engine below).

Alternatively, pandoc can use ConTeXt, pdfroff, or any of the following HTML/CSS-to-PDF-engines, to create a PDF: wkhtmltopdf, weasyprint or prince. To do this, specify an output file with a .pdf extension, as before, but add the --pdf-engine option or -t context, -t html, or -t ms to the command line (-t html defaults to --pdf-engine=wkhtmltopdf).

PDF output uses variables for LaTeX (with a LaTeX engine); variables for ConTeXt (with ConTeXt); or variables for wkhtmltopdf (an HTML/CSS-to-PDF engine; --css also affects the output).

To debug the PDF creation, it can be useful to look at the intermediate representation: instead of -o test.pdf, use for example -s -o test.tex to output the generated LaTeX. You can then test it with pdflatex test.tex.

When using LaTeX, the following packages need to be available (they are included with all recent versions of TeX Live): amsfonts, amsmath, lm, unicode-math, ifxetex, ifluatex, listings (if the --listings option is used), fancyvrb, longtable, booktabs, graphicx and grffile (if the document contains images), hyperref, xcolor, ulem, geometry (with the geometry variable set), setspace (with linestretch), and babel (with lang). The use of xelatex or lualatex as the PDF engine requires fontspec. xelatex uses polyglossia (with lang), xecjk, and bidi (with the dir variable set). If the mathspec variable is set, xelatex will use mathspec instead of unicode-math. The upquote and microtype packages are used if available, and csquotes will be used for typography if \usepackage{csquotes} is present in the template or included via /H/--include-in-header. The natbib, biblatex, bibtex, and biber packages can optionally be used for citation rendering. The following packages will be used to improve output quality if present, but pandoc does not require them to be present: upquote (for straight quotes in verbatim environments), microtype (for better spacing adjustments), parskip (for better inter-paragraph spaces), xurl (for better line breaks in URLs), bookmark (for better PDF bookmarks), and footnotehyper or footnote (to allow footnotes in tables).

Reading from the Web

Instead of an input file, an absolute URI may be given. In this case pandoc will fetch the content using HTTP:
Description

pandoc -f html -t markdown http://www.fsf.org

It is possible to supply a custom User-Agent string or other header when requesting a document from a URL:

pandoc -f html -t markdown --request-header User-Agent:"Mozilla/5.0" \ http://www.fsf.org
Options

General options

\texttt{-f \textit{FORMAT}, -r \textit{FORMAT}, --from=\textit{FORMAT}, --read=\textit{FORMAT}} Specify input format. \textit{FORMAT} can be:

- commonmark (CommonMark Markdown)
- creole (Creole 1.0)
- docbook (DocBook)
- docx (Word docx)
- dokuwiki (DokuWiki markup)
- epub (EPUB)
- fb2 (FictionBook2 e-book)
- gfm (GitHub-Flavored Markdown), or the deprecated and less accurate markdown\_github; use markdown\_github only if you need extensions not supported in gfm.
- haddock (Haddock markup)
- html (HTML)
- ipynb (Jupyter notebook)
- jats (JATS XML)
- json (JSON version of native AST)
- latex (LaTeX)
- markdown (Pandoc’s Markdown)
- markdown\_mmd (MultiMarkdown)
- markdown\_phpextra (PHP Markdown Extra)
- markdown\_strict (original unextended Markdown)
- mediawiki (MediaWiki markup)
- man (roff man)
- muse (Muse)
- native (native Haskell)
- odt (ODT)
- opml (OPML)
- org (Emacs Org mode)
- rst (reStructuredText)
- t2t (txt2tags)
- textile (Textile)
Options

- tikiwiki (TikiWiki markup)
- twiki (TWiki markup)
- vimwiki (Vimwiki)

Extensions can be individually enabled or disabled by appending +EXTENSION or -EXTENSION to the format name. See Extensions below, for a list of extensions and their names. See --list-input-formats and --list-extensions, below.

-t FORMAT, -w FORMAT, --to=FORMAT, --write=FORMAT Specify output format. FORMAT can be:

- asciidoc (AsciiDoc) or asciidoctor (AsciiDoctor)
- beamer (LaTeX beamer slide show)
- commonmark (CommonMark Markdown)
- context (ConTeXt)
- docbook or docbook4 (DocBook 4)
- docbook5 (DocBook 5)
- dox (Word docx)
- dokuwiki (DokuWiki markup)
- epub or epub3 (EPUB v3 book)
- epub2 (EPUB v2)
- fb2 (FictionBook2 e-book)
- gfm (GitHub-Flavored Markdown), or the deprecated and less accurate markdown_github; use markdown_github only if you need extensions not supported in gfm.
- haddock (Haddock markup)
- html or html5 (HTML, i.e. HTML5/XHTML polyglot markup)
- html4 (XHTML 1.0 Transitional)
- icml (InDesign ICML)
- ipynb (Jupyter notebook)
- jats (JATS XML)
- jira (Jira wiki markup)
- json (JSON version of native AST)
- latex (LaTeX)
- man (roff man)
- markdown (Pandoc's Markdown)
- markdown_mmd (MultiMarkdown)
- markdown_phpextra (PHP Markdown Extra)
- markdown_strict (original unextended Markdown)
- mediawiki (MediaWiki markup)
- ms (roff ms)
- muse (Muse),
- native (native Haskell),
- odt (OpenOffice text document)
- opml (OPML)
• opendocument (OpenDocument)
• org (Emacs Org mode)
• plain (plain text),
• pptx (PowerPoint slide show)
• rst (reStructuredText)
• rtf (Rich Text Format)
• texinfo (GNU Texinfo)
• textile (Textile)
• slideous (Slideous HTML and JavaScript slide show)
• slidy (Slidy HTML and JavaScript slide show)
• dzslides (DZSlides HTML5 + JavaScript slide show),
• revealjs (reveal.js HTML5 + JavaScript slide show)
• s5 (S5 HTML and JavaScript slide show)
• tei (TEI Simple)
• xwiki (XWiki markup)
• zimwiki (ZimWiki markup)
• the path of a custom lua writer, see Custom writers below

Note that odt, docx, and epub output will not be directed to stdout unless forced with
-o -.

Extensions can be individually enabled or disabled by appending +EXTENSION or –EXTENSION to the format name. See Extensions below, for a list of extensions and their names. See --list-output-formats and --list-extensions, below.

-o FILE, --output=FILE Write output to FILE instead of stdout. If FILE is –, output will go
to stdout, even if a non-textual format (docx, odt, epub2, epub3) is specified.

--data-dir=DIRECTORY Specify the user data directory to search for pandoc data files.
If this option is not specified, the default user data directory will be used. On *nix
and macOS systems this will be the pandoc subdirectory of the XDG data direc-
tory (by default, $HOME/.local/share, overridable by setting the XDG_DATA_HOME
environment variable). If that directory does not exist, $HOME/.pandoc will be
used (for backwards compatibility). In Windows the default user data directory
is C:\Users\USERNAME\AppData\Roaming\pandoc. You can find the default user
data directory on your system by looking at the output of pandoc --version. A
reference.odt, reference.docx, epub.css, templates, slidy, slideous, or s5
directory placed in this directory will override pandoc's normal defaults.

--bash-completion Generate a bash completion script. To enable bash completion with
pandoc, add this to your .bashrc:

eval "$(pandoc --bash-completion)"

--verbose Give verbose debugging output. Currently this only has an effect with PDF output.
Options

--quiet Suppress warning messages.

--fail-if-warnings Exit with error status if there are any warnings.

--log=FILE Write log messages in machine-readable JSON format to FILE. All messages above DEBUG level will be written, regardless of verbosity settings (--verbose, --quiet).

--list-input-formats List supported input formats, one per line.

--list-output-formats List supported output formats, one per line.

--list-extensions[=FORMAT] List supported extensions, one per line, preceded by a + or - indicating whether it is enabled by default in FORMAT. If FORMAT is not specified, defaults for pandoc’s Markdown are given.

--list-highlight-languages List supported languages for syntax highlighting, one per line.

--list-highlight-styles List supported styles for syntax highlighting, one per line. See --highlight-style.

-v, --version Print version.

-h, --help Show usage message.

Reader options

--base-header-level=NUMBER Specify the base level for headings (defaults to 1).

--strip-empty-paragraphs Deprecated. Use the +empty_paragraphs extension instead. Ignore paragraphs with no content. This option is useful for converting word processing documents where users have used empty paragraphs to create inter-paragraph space.

--indented-code-classes=CLASSES Specify classes to use for indented code blocks—for example, perl, numberLines or haskell. Multiple classes may be separated by spaces or commas.

--default-image-extension=EXTENSION Specify a default extension to use when image paths/URLs have no extension. This allows you to use the same source for formats that require different kinds of images. Currently this option only affects the Markdown and LaTeX readers.

--file-scope Parse each file individually before combining for multifile documents. This will allow footnotes in different files with the same identifiers to work as expected. If this option is set, footnotes and links will not work across files. Reading binary files (docx, odt, epub) implies --file-scope.
--F PROGRAM, --filter=PROGRAM Specify an executable to be used as a filter transforming the pandoc AST after the input is parsed and before the output is written. The executable should read JSON from stdin and write JSON to stdout. The JSON must be formatted like pandoc’s own JSON input and output. The name of the output format will be passed to the filter as the first argument. Hence,

```
pandoc --filter ./caps.py -t latex
```

is equivalent to

```
pandoc -t json | ./caps.py latex | pandoc -f json -t latex
```

The latter form may be useful for debugging filters.

Filters may be written in any language. Text.Pandoc.JSON exports toJSONFilter to facilitate writing filters in Haskell. Those who would prefer to write filters in python can use the module pandocfilters, installable from PyPI. There are also pandoc filter libraries in PHP, perl, and JavaScript/node.js.

In order of preference, pandoc will look for filters in

1. a specified full or relative path (executable or non-executable)
2. $DATADIR/filters (executable or non-executable) where $DATADIR is the user data directory (see --data-dir, above).
3. $PATH (executable only)

Filters and lua-filters are applied in the order specified on the command line.

--lua-filter=SCRIPT Transform the document in a similar fashion as JSON filters (see --filter), but use pandoc’s build-in lua filtering system. The given lua script is expected to return a list of lua filters which will be applied in order. Each lua filter must contain element-transforming functions indexed by the name of the AST element on which the filter function should be applied.

The pandoc lua module provides helper functions for element creation. It is always loaded into the script’s lua environment.

The following is an example lua script for macro-expansion:

```
function expand_hello_world(inline)
    if inline.c == '{{helloworld}}' then
        return pandoc.Emph{ pandoc.Str "Hello, World" }
    else
        return inline
    end
end

return {{Str = expand_hello_world}}
```
Options

In order of preference, pandoc will look for lua filters in

1. a specified full or relative path (executable or non-executable)
2. $DATADIR/filters (executable or non-executable) where $DATADIR is the user data directory (see --data-dir, above).

--M KEY=[VAL], --metadata=KEY:[VAL] Set the metadata field KEY to the value VAL. A value specified on the command line overrides a value specified in the document using YAML metadata blocks. Values will be parsed as YAML boolean or string values. If no value is specified, the value will be treated as Boolean true. Like --variable, --metadata causes template variables to be set. But unlike --variable, --metadata affects the metadata of the underlying document (which is accessible from filters and may be printed in some output formats) and metadata values will be escaped when inserted into the template.

--metadata-file=FILE Read metadata from the supplied YAML (or JSON) file. This option can be used with every input format, but string scalars in the YAML file will always be parsed as Markdown. Generally, the input will be handled the same as in YAML metadata blocks. Metadata values specified inside the document, or by using --M, overwrite values specified with this option.

-p, --preserve-tabs Preserve tabs instead of converting them to spaces (the default). Note that this will only affect tabs in literal code spans and code blocks; tabs in regular text will be treated as spaces.

--tab-stop=NUMBER Specify the number of spaces per tab (default is 4).

--track-changes=accept|reject|all Specifies what to do with insertions, deletions, and comments produced by the MS Word “Track Changes” feature. accept (the default), inserts all insertions, and ignores all deletions. reject inserts all deletions and ignores insertions. Both accept and reject ignore comments. all puts in insertions, deletions, and comments, wrapped in spans with insertion, deletion, comment-start, and comment-end classes, respectively. The author and time of change is included. all is useful for scripting: only accepting changes from a certain reviewer, say, or before a certain date. If a paragraph is inserted or deleted, track-changes=all produces a span with the class paragraph-insertion/paragraph-deletion before the affected paragraph break. This option only affects the docx reader.

--extract-media=DIR Extract images and other media contained in or linked from the source document to the path DIR, creating it if necessary, and adjust the images references in the document so they point to the extracted files. If the source format is a binary container (docx, epub, or odt), the media is extracted from the container and the original filenames are used. Otherwise the media is read from the file system or downloaded, and new filenames are constructed based on SHA1 hashes of the contents.
--abbreviations=FILE Specifies a custom abbreviations file, with abbreviations one to a line. If this option is not specified, pandoc will read the data file abbreviations from the user data directory or fall back on a system default. To see the system default, use pandoc --print-default-data-file=abbreviations. The only use pandoc makes of this list is in the Markdown reader. Strings ending in a period that are found in this list will be followed by a nonbreaking space, so that the period will not produce sentence-ending space in formats like LaTeX.

General writer options

-s, --standalone Produce output with an appropriate header and footer (e.g. a standalone HTML, LaTeX, TEI, or RTF file, not a fragment). This option is set automatically for pdf, epub, epub3, fb2, docx, and odt output. For native output, this option causes metadata to be included; otherwise, metadata is suppressed.

--template=FILE|URL Use the specified file as a custom template for the generated document. Implies --standalone. See Templates, below, for a description of template syntax. If no extension is specified, an extension corresponding to the writer will be added, so that --template=special looks for special.html for HTML output. If the template is not found, pandoc will search for it in the templates subdirectory of the user data directory (see --data-dir). If this option is not used, a default template appropriate for the output format will be used (see -D/--print-default-template).

-V KEY[=VAL], --variable=KEY[:VAL] Set the template variable KEY to the value VAL when rendering the document in standalone mode. This is generally only useful when the --template option is used to specify a custom template, since pandoc automatically sets the variables used in the default templates. If no VAL is specified, the key will be given the value true.

-D FORMAT, --print-default-template=FORMAT Print the system default template for an output FORMAT. (See -t for a list of possible FORMATs.) Templates in the user data directory are ignored. This option may be used with -o/--output to redirect output to a file, but -o/--output must come before --print-default-template on the command line.

--print-default-data-file=FILE Print a system default data file. Files in the user data directory are ignored. This option may be used with -o/--output to redirect output to a file, but -o/--output must come before --print-default-data-file on the command line.

--eol=crlf|lf|native Manually specify line endings: crlf (Windows), lf (macOS/Linux/UNIX), or native (line endings appropriate to the OS on which pandoc is being run). The default is native.
Options

--dpi=NUMBER Specify the dpi (dots per inch) value for conversion from pixels to inch/centimeters and vice versa. The default is 96dpi. Technically, the correct term would be ppi (pixels per inch).

--wrap=auto|none|preserve Determine how text is wrapped in the output (the source code, not the rendered version). With auto (the default), pandoc will attempt to wrap lines to the column width specified by --columns (default 72). With none, pandoc will not wrap lines at all. With preserve, pandoc will attempt to preserve the wrapping from the source document (that is, where there are nonsemantic newlines in the source, there will be nonsemantic newlines in the output as well). Automatic wrapping does not currently work in HTML output. In ipynb output, this option affects wrapping of the contents of markdown cells.

--columns=NUMBER Specify length of lines in characters. This affects text wrapping in the generated source code (see --wrap). It also affects calculation of column widths for plain text tables (see Tables below).

--toc, --table-of-contents Include an automatically generated table of contents (or, in the case of latex, context, docx, odt, opendocument, rst, or ms, an instruction to create one) in the output document. This option has no effect unless --s/--standalone is used, and it has no effect on man, docbook4, docbook5, or jats output.

Note that if you are producing a PDF via ms, the table of contents will appear at the beginning of the document, before the title. If you would prefer it to be at the end of the document, use the option --pdf-engine-opt=--no-toc-relocation.

--toc-depth=NUMBER Specify the number of section levels to include in the table of contents. The default is 3 (which means that level-1, 2, and 3 headings will be listed in the contents).

--strip-comments Strip out HTML comments in the Markdown or Textile source, rather than passing them on to Markdown, Textile or HTML output as raw HTML. This does not apply to HTML comments inside raw HTML blocks when the markdown_in_html_blocks extension is not set.

--no-highlight Disables syntax highlighting for code blocks and inlines, even when a language attribute is given.

--highlight-style=STYLE|FILE Specifies the coloring style to be used in highlighted source code. Options are pygments (the default), kate, monochrome, breezeDark, espresso, zenburn, haddock, and tango. For more information on syntax highlighting in pandoc, see Syntax highlighting, below. See also --list-highlight-styles.

Instead of a STYLE name, a JSON file with extension .theme may be supplied. This will be parsed as a KDE syntax highlighting theme and (if valid) used as the highlighting style.

To generate the JSON version of an existing style, use --print-highlight-style.
Options affecting specific writers

--print-highlight-style=STYLE|FILE Prints a JSON version of a highlighting style, which can be modified, saved with a .theme extension, and used with --highlight-style. This option may be used with -o/--output to redirect output to a file, but -o/--output must come before --print-highlight-style on the command line.

--syntax-definition=FILE Instructs pandoc to load a KDE XML syntax definition file, which will be used for syntax highlighting of appropriately marked code blocks. This can be used to add support for new languages or to use altered syntax definitions for existing languages.

-H FILE, --include-in-header=FILE|URL Include contents of FILE, verbatim, at the end of the header. This can be used, for example, to include special CSS or JavaScript in HTML documents. This option can be used repeatedly to include multiple files in the header. They will be included in the order specified. Implies --standalone.

-B FILE, --include-before-body=FILE|URL Include contents of FILE, verbatim, at the beginning of the document body (e.g. after the <body> tag in HTML, or the \begin{document} command in LaTeX). This can be used to include navigation bars or banners in HTML documents. This option can be used repeatedly to include multiple files. They will be included in the order specified. Implies --standalone.

-A FILE, --include-after-body=FILE|URL Include contents of FILE, verbatim, at the end of the document body (before the </body> tag in HTML, or the \end{document} command in LaTeX). This option can be used repeatedly to include multiple files. They will be included in the order specified. Implies --standalone.

--resource-path=SEARCHPATH List of paths to search for images and other resources. The paths should be separated by : on Linux, UNIX, and macOS systems, and by ; on Windows. If --resource-path is not specified, the default resource path is the working directory. Note that, if --resource-path is specified, the working directory must be explicitly listed or it will not be searched. For example: --resource-path=.:test will search the working directory and the test subdirectory, in that order.

--resource-path only has an effect if (a) the output format embeds images (for example, docx, pdf, or html with --self-contained) or (b) it is used together with --extract-media.

--request-header=NAME:VAL Set the request header NAME to the value VAL when making HTTP requests (for example, when a URL is given on the command line, or when resources used in a document must be downloaded). If you’re behind a proxy, you also need to set the environment variable http_proxy to http://....

Options affecting specific writers

--self-contained Produce a standalone HTML file with no external dependencies, using data: URIs to incorporate the contents of linked scripts, stylesheets, images, and
options

Videos. Implies --standalone. The resulting file should be "self-contained," in the sense that it needs no external files and no net access to be displayed properly by a browser. This option works only with HTML output formats, including html4, html5, html+lhs, html5+lhs, s5, slidy, slideous, dzslides, and reveal.js. Scripts, images, and stylesheets at absolute URLs will be downloaded; those at relative URLs will be sought relative to the working directory (if the first source file is local) or relative to the base URL (if the first source file is remote). Elements with the attribute data-external="1" will be left alone; the documents they link to will not be incorporated in the document. Limitation: resources that are loaded dynamically through JavaScript cannot be incorporated; as a result, --self-contained does not work with --mathjax, and some advanced features (e.g. zoom or speaker notes) may not work in an offline "self-contained" reveal.js slideshow.

--html-q-tags Use <q> tags for quotes in HTML.

--ascii Use only ASCII characters in output. Currently supported for XML and HTML formats (which use entities instead of UTF-8 when this option is selected), CommonMark, gfm, and Markdown (which use entities), roff ms (which use hexadecimal escapes), and to a limited degree LaTeX (which uses standard commands for accented characters when possible). roffman output uses ASCII by default.

--reference-links Use reference-style links, rather than inline links, in writing Markdown or reStructuredText. By default inline links are used. The placement of link references is affected by the --reference-location option.

--reference-location = block|section|document Specify whether footnotes (and references, if reference-links is set) are placed at the end of the current (top-level) block, the current section, or the document. The default is document. Currently only affects the markdown writer.

--atx-headers Use ATX-style headings in Markdown output. The default is to use setext-style headings for levels 1 to 2, and then ATX headings. (Note: for gfm output, ATX headings are always used.) This option also affects markdown cells in ipynb output.

--top-level-division=[default|section|chapter|part] Treat top-level headings as the given division type in LaTeX, ConTeXt, DocBook, and TEI output. The hierarchy order is part, chapter, then section; all headings are shifted such that the top-level heading becomes the specified type. The default behavior is to determine the best division type via heuristics: unless other conditions apply, section is chosen. When the LaTeX document class is set to report, book, or memoir (unless the article option is specified), chapter is implied as the setting for this option. If beamer is the output format, specifying either chapter or part will cause top-level headings to become \part{...}, while second-level headings remain as their default type.

-N, --number-sections Number section headings in LaTeX, ConTeXt, HTML, or EPUB output. By default, sections are not numbered. Sections with class unnumbered will never be numbered, even if --number-sections is specified.
Options affecting specific writers

--number-offset=NUMBER[,NUMBER,...] Offset for section headings in HTML output (ignored in other output formats). The first number is added to the section number for top-level headings, the second for second-level headings, and so on. So, for example, if you want the first top-level heading in your document to be numbered “6”, specify --number-offset=5. If your document starts with a level-2 heading which you want to be numbered “1.5”, specify --number-offset=1,4. Offsets are 0 by default. Implies --number-sections.

--listings Use the listings package for LaTeX code blocks. The package does not support multi-byte encoding for source code. To handle UTF-8 you would need to use a custom template. This issue is fully documented here: Encoding issue with the listings package.

-i, --incremental Make list items in slide shows display incrementally (one by one). The default is for lists to be displayed all at once.

--slide-level=NUMBER Specifies that headings with the specified level create slides (for beamer, s5, slidy, slideous, dzslides). Headings above this level in the hierarchy are used to divide the slide show into sections; headings below this level create subheads within a slide. Note that content that is not contained under slide-level headings will not appear in the slide show. The default is to set the slide level based on the contents of the document; see Structuring the slide show.

--section-divs Wrap sections in <section> tags (or <div> tags for html4), and attach identifiers to the enclosing <section> (or <div>) rather than the heading itself. See Heading identifiers, below.

--email-obfuscation=none|javascript|references Specify a method for obfuscating mailto: links in HTML documents. none leaves mailto: links as they are. javascript obfuscates them using JavaScript. references obfuscates them by printing their letters as decimal or hexadecimal character references. The default is none.

--id-prefix=STRING Specify a prefix to be added to all identifiers and internal links in HTML and DocBook output, and to footnote numbers in Markdown and Haddock output. This is useful for preventing duplicate identifiers when generating fragments to be included in other pages.

-T STRING, --title-prefix=STRING Specify STRING as a prefix at the beginning of the title that appears in the HTML header (but not in the title as it appears at the beginning of the HTML body). Implies --standalone.

-c URL, --css=URL Link to a CSS style sheet. This option can be used repeatedly to include multiple files. They will be included in the order specified.

A stylesheet is required for generating EPUB. If none is provided using this option (or the css or stylesheet metadata fields), pandoc will look for a file epub.css in the
user data directory (see \texttt{--data-dir}). If it is not found there, sensible defaults will be used.

\texttt{--reference-doc=FILE} Use the specified file as a style reference in producing a docx or ODT file.

\textbf{Docx} For best results, the reference docx should be a modified version of a docx file produced using pandoc. The contents of the reference docx are ignored, but its stylesheets and document properties (including margins, page size, header, and footer) are used in the new docx. If no reference docx is specified on the command line, pandoc will look for a file \texttt{reference.docx} in the user data directory (see \texttt{--data-dir}). If this is not found either, sensible defaults will be used.

To produce a custom \texttt{reference.docx}, first get a copy of the default \texttt{reference.docx}: \texttt{pandoc -o custom-reference.docx --print-default-data-file reference.docx}. Then open \texttt{custom-reference.docx} in Word, modify the styles as you wish, and save the file. For best results, do not make changes to this file other than modifying the styles used by pandoc:

Paragraph styles:

- Normal
- Body Text
- First Paragraph
- Compact
- Title
- Subtitle
- Author
- Date
- Abstract
- Bibliography
- Heading 1
- Heading 2
- Heading 3
- Heading 4
- Heading 5
- Heading 6
- Heading 7
- Heading 8
- Heading 9
- Block Text
- Footnote Text
- Definition Term
- Definition
- Caption
- Table Caption
Options affecting specific writers

- Image Caption
- Figure
- Captioned Figure
- TOC Heading

Character styles:

- Default Paragraph Font
- Body Text Char
- Verbatim Char
- Footnote Reference
- Hyperlink

Table style:

- Table

**ODT** For best results, the reference ODT should be a modified version of an ODT produced using pandoc. The contents of the reference ODT are ignored, but its stylesheets are used in the new ODT. If no reference ODT is specified on the command line, pandoc will look for a file reference.odt in the user data directory (see `--data-dir`). If this is not found either, sensible defaults will be used.


**PowerPoint** Templates included with Microsoft PowerPoint 2013 (either with .pptx or .potx extension) are known to work, as are most templates derived from these.

The specific requirement is that the template should begin with the following first four layouts:

1. Title Slide
2. Title and Content
3. Section Header
4. Two Content

All templates included with a recent version of MS PowerPoint will fit these criteria. (You can click on Layout under the Home menu to check.)

You can also modify the default reference.pptx: first run pandoc --o custom-reference.pptx --print-default-data-file reference.pptx, and then modify custom-reference.pptx in MS PowerPoint (pandoc will use the first four layout slides, as mentioned above).
Options

--epub-cover-image=FILE Use the specified image as the EPUB cover. It is recommended that the image be less than 1000px in width and height. Note that in a Markdown source document you can also specify `cover-image` in a YAML metadata block (see EPUB Metadata, below).

--epub-metadata=FILE Look in the specified XML file for metadata for the EPUB. The file should contain a series of Dublin Core elements. For example:

```
<dc:rights>Creative Commons</dc:rights>
<dc:language>es-AR</dc:language>
```

By default, pandoc will include the following metadata elements: `<dc:title>` (from the document title), `<dc:creator>` (from the document authors), `<dc:date>` (from the document date, which should be in ISO 8601 format), `<dc:language>` (from the `lang` variable, or, if it is not set, the locale), and `<dc:identifier id="BookId">` (a randomly generated UUID). Any of these may be overridden by elements in the metadata file.

Note: if the source document is Markdown, a YAML metadata block in the document can be used instead. See below under EPUB Metadata.

--epub-embed-font=FILE Embed the specified font in the EPUB. This option can be repeated to embed multiple fonts. Wildcards can also be used: for example, `DejaVuSans-*.ttf`. However, if you use wildcards on the command line, be sure to escape them or put the whole filename in single quotes, to prevent them from being interpreted by the shell. To use the embedded fonts, you will need to add declarations like the following to your CSS (see --css):

```
@font-face {
  font-family: DejaVuSans;
  font-style: normal;
  font-weight: normal;
  src:url("DejaVuSans-Regular.ttf");
}
@font-face {
  font-family: DejaVuSans;
  font-style: normal;
  font-weight: bold;
  src:url("DejaVuSans-Bold.ttf");
}
@font-face {
  font-family: DejaVuSans;
  font-style: italic;
  font-weight: normal;
  src:url("DejaVuSans-Oblique.ttf");
}
```
font-family: DejaVuSans;
font-style: italic;
font-weight: bold;
src:url("DejaVuSans-BoldOblique.ttf");
}
body { font-family: "DejaVuSans"; }

--epub-chapter-level=NUMBER Specify the heading level at which to split the EPUB into separate “chapter” files. The default is to split into chapters at level-1 headings. This option only affects the internal composition of the EPUB, not the way chapters and sections are displayed to users. Some readers may be slow if the chapter files are too large, so for large documents with few level-1 headings, one might want to use a chapter level of 2 or 3.

--epub-subdirectory=DIRNAME Specify the subdirectory in the OCF container that is to hold the EPUB-specific contents. The default is EPUB. To put the EPUB contents in the top level, use an empty string.

--ipynb-output=all|none|best Determines how ipynb output cells are treated. all means that all of the data formats included in the original are preserved. none means that the contents of data cells are omitted. best causes pandoc to try to pick the richest data block in each output cell that is compatible with the output format. The default is best.

--pdf-engine=PROGRAM Use the specified engine when producing PDF output. Valid values are pdflatex, lualatex, xelatex, latexmk, tectonic, wkhtmltopdf, weasyprint, prince, context, and pdfroff. The default is pdflatex. If the engine is not in your PATH, the full path of the engine may be specified here.

--pdf-engine-opt=STRING Use the given string as a command-line argument to the pdf-engine. For example, to use a persistent directory foo for latexmk’s auxiliary files, use --pdf-engine-opt=-outdir=foo. Note that no check for duplicate options is done.

Citation rendering

--bibliography=FILE Set the bibliography field in the document’s metadata to FILE, overriding any value set in the metadata, and process citations using pandoc-citeproc. (This is equivalent to --metadata bibliography=FILE --filter pandoc-citeproc.) If --natbib or --biblatex is also supplied, pandoc-citeproc is not used, making this equivalent to --metadata bibliography=FILE. If you supply this argument multiple times, each FILE will be added to bibliography.
Options

**--csl=FILE** Set the `csl` field in the document’s metadata to `FILE`, overriding any value set in the metadata. (This is equivalent to `--metadata csl=FILE`.) This option is only relevant with `pandoc-citeproc`.

**--citation-abbreviations=FILE** Set the `citation-abbreviations` field in the document’s metadata to `FILE`, overriding any value set in the metadata. (This is equivalent to `--metadata citation-abbreviations=FILE`. This option is only relevant with `pandoc-citeproc`.

**--natbib** Use `natbib` for citations in LaTeX output. This option is not for use with the `pandoc-citeproc` filter or with PDF output. It is intended for use in producing a LaTeX file that can be processed with `biblatex`.

**--biblatex** Use `biblatex` for citations in LaTeX output. This option is not for use with the `pandoc-citeproc` filter or with PDF output. It is intended for use in producing a LaTeX file that can be processed with `biblatex` or `biber`.

### Math rendering in HTML

The default is to render TeX math as far as possible using Unicode characters. Formulas are put inside a `span` with `class="math"`, so that they may be styled differently from the surrounding text if needed. However, this gives acceptable results only for basic math, usually you will want to use **--mathjax** or another of the following options.

**--mathjax[=URL]** Use MathJax to display embedded TeX math in HTML output. TeX math will be put between `\(...\)` (for inline math) or `\[...\]` (for display math) and wrapped in `</span>` tags with class `math`. Then the MathJax JavaScript will render it. The `URL` should point to the MathJax.js load script. If a `URL` is not provided, a link to the Cloudflare CDN will be inserted.

**--mathml** Convert TeX math to MathML (in `epub3`, `docbook4`, `docbook5`, `jats`, `html4` and `html5`). This is the default in `odt` output. Note that currently only Firefox and Safari (and select e-book readers) natively support MathML.

**--webtex[=URL]** Convert TeX formulas 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. For SVG images you can for example use **--webtex** `https://latex.codecogs.com/svg.latex?`. If no URL is specified, the CodeCogs URL generating PNGs will be used (https://latex.codecogs.com/png.latex?). Note: the **--webtex** option will affect Markdown output as well as HTML, which is useful if you’re targeting a version of Markdown without native math support.

**--katex[=URL]** Use KaTeX to display embedded TeX math in HTML output. The `URL` is the base URL for the KaTeX library. That directory should contain a `katex.min.js` and a `katex.min.css` file. If a `URL` is not provided, a link to the KaTeX CDN will be inserted.
--gładtex Enclose TeX math in `<eq>` tags in HTML output. The resulting HTML can then be processed by GladTeX to produce images of the typeset formulas and an HTML file with links to these images. So, the procedure is:

pandoc -s --gladtex input.md -o myfile.htex
gladtex -d myfile-images myfile.htex
# produces myfile.html and images in myfile-images

Options for wrapper scripts

--dump-args Print information about command-line arguments to `stdout`, then exit. This option is intended primarily for use in wrapper scripts. The first line of output contains the name of the output file specified with the `-o` option, or `-` (for `stdout`) if no output file was specified. The remaining lines contain the command-line arguments, one per line, in the order they appear. These do not include regular pandoc options and their arguments, but do include any options appearing after a `--` separator at the end of the line.

--ignore-args Ignore command-line arguments (for use in wrapper scripts). Regular pandoc options are not ignored. Thus, for example,

pandoc --ignore-args -o foo.html -s foo.txt -- -e latin1

is equivalent to

pandoc -o foo.html -s
**Templates**

When the `--standalone` option is used, pandoc uses a template to add header and footer material that is needed for a self-standing document. To see the default template that is used, just type

```
pandoc -D *FORMAT*
```

where `FORMAT` is the name of the output format. A custom template can be specified using the `--template` option. You can also override the system default templates for a given output format `FORMAT` by putting a file `templates/default.*FORMAT*` in the user data directory (see `--data-dir`, above). **Exceptions:**

- For `odt` output, customize the default `opendocument` template.
- For `pdf` output, customize the default `latex` template (or the default `context` template, if you use `--t context`, or the default `ms` template, if you use `--t ms`, or the default `html` template, if you use `--t html`).
- `docx` and `pptx` have no template (however, you can use `--reference-doc` to customize the output).

Templates contain *variables*, which allow for the inclusion of arbitrary information at any point in the file. They may be set at the command line using the `--variable` option. If a variable is not set, pandoc will look for the key in the document’s metadata – which can be set using either YAML metadata blocks or with the `--metadata` option.

**Metadata variables**

*title, author, date* allow identification of basic aspects of the document. Included in PDF metadata through LaTeX and ConTeXt. These can be set through a pandoc title block, which allows for multiple authors, or through a YAML metadata block:

```---
author:
  - Aristotle
  - Peter Abelard
...```
Note that if you just want to set PDF or HTML metadata, without including a title block in the document itself, you can set the title-meta, author-meta, and date-meta variables. (By default these are set automatically, based on title, author, and date.)

**subtitle** document subtitle, included in HTML, EPUB, LaTeX, ConTeXt, and docx documents

**abstract** document summary, included in LaTeX, ConTeXt, AsciiDoc, and docx documents

**keywords** list of keywords to be included in HTML, PDF, ODT, ppx, docx and AsciiDoc metadata; repeat as for author, above

**subject** document subject, included in ODT, PDF, docx and ppx metadata

**description** document description, included in ODT, docx and ppx metadata. Some applications show this as Comments metadata.

**category** document category, included in docx and ppx metadata

Additionally, any root-level string metadata, not included in ODT, docx or ppx metadata is added as a *custom property*. The following YAML metadata block for instance:

```yaml
---
title: 'This is the title'
subtitle: "This is the subtitle"
author:
- Author One
- Author Two
description: |
    This is a long
description.

    It consists of two paragraphs
...
```

will include title, author and description as standard document properties and subtitle as a custom property when converting to docx, ODT or ppx.

**Language variables**

**lang** identifies the main language of the document using IETF language tags (following the BCP 47 standard), such as en or en-GB. The Language subtag lookup tool can look
up or verify these tags. This affects most formats, and controls hyphenation in PDF output when using \LaTeX{} (through `babel` and `polyglossia`) or ConText.

Use native pandoc Divs and Spans with the `lang` attribute to switch the language:

```---
lang: en-GB
...```

Text in the main document language (British English).

```::: {lang=fr-CA}
> Cette citation est écrite en français canadien.
:::````

More text in English. ['Zitat auf Deutsch.'][lang=de]

`dir` the base script direction, either `rtl` (right-to-left) or `ltr` (left-to-right).

For bidirectional documents, native pandoc spans and divs with the `dir` attribute (value `rtl` or `ltr`) can be used to override the base direction in some output formats. This may not always be necessary if the final renderer (e.g., the browser, when generating HTML) supports the Unicode Bidirectional Algorithm.

When using \LaTeX{} for bidirectional documents, only the `xelatex` engine is fully supported (use `--pdf-engine=xelatex`).

### Variables for HTML slides

These affect HTML output when producing slide shows with pandoc. All reveal.js configuration options are available as variables.

- `revealjs-url` base URL for reveal.js documents (defaults to `reveal.js`)
- `s5-url` base URL for S5 documents (defaults to `s5/default`)
- `slidy-url` base URL for Slidy documents (defaults to `https://www.w3.org/Talks/Tools/Slidy2`)
- `slideous-url` base URL for Slideous documents (defaults to `slideous`)

### Variables for Beamer slides

These variables change the appearance of PDF slides using beamer.

- `aspectratio` slide aspect ratio (43 for 4:3 [default], 169 for 16:9, 1610 for 16:10, 149 for 14:9, 141 for 1.41:1, 54 for 5:4, 32 for 3:2)
- `beamerarticle` produce an article from Beamer slides
**Templates**

- **beameroption**: add extra beamer option with \setbeameroption{}
- **institute**: author affiliations: can be a list when there are multiple authors
- **logo**: logo image for slides
- **navigation**: controls navigation symbols (default is empty for no navigation symbols; other valid values are frame, vertical, and horizontal)
- **section-titles**: enables “title pages” for new sections (default is true)
- **theme, colortheme, fonttheme, innertheme, outertheme**: beamer themes:
- **themeoptions**: options for LaTeX beamer themes (a list).
- **titlegraphic**: image for title slide

**Variables for LaTeX**

Pandoc uses these variables when creating a PDF with a LaTeX engine.

**Layout**

- **block-headings**: make \paragraph and \subparagraph (fourth- and fifth-level headings, or fifth- and sixth-level with book classes) free-standing rather than run-in; requires further formatting to distinguish from \subsection (third- or fourth-level headings). Instead of using this option, KOMA-Script can adjust headings more extensively:

```latex
---

documentclass: scrartcl
header-includes: |
\RedeclareSectionCommand[
  beforeskip=-10pt plus -2pt minus -1pt,
  afterskip=1sp plus -1sp minus 1sp,
  font=\normalfont\itshape]{paragraph}
\RedeclareSectionCommand[
  beforeskip=-10pt plus -2pt minus -1pt,
  afterskip=1sp plus -1sp minus 1sp,
  font=\normalfont\scshape, indent=0pt]{subparagraph}
...
```

- **classoption**: option for document class, e.g. oneside; repeat for multiple options:

```latex
---
classoption:
  - twocolumn
  - landscape
...
**documentclass** document class: usually one of the standard classes, article, book, and report; the KOMA-Script equivalents, scrartcl, scrbook, and scrreprt, which default to smaller margins; or memoir

**geometry** option for geometry package, e.g. margin=1in; repeat for multiple options:

```latex
---
geometry:
- top=30mm
- left=20mm
- height=rounded
...
```

**indent** uses document class settings for indentation (the default LaTeX template otherwise removes indentation and adds space between paragraphs)

**linestretch** adjusts line spacing using the setspace package, e.g. 1.25, 1.5

**margin-left, margin-right, margin-top, margin-bottom** sets margins if geometry is not used (otherwise geometry overrides these)

**pagemode** control \pagemode{}: the default article class supports plain (default), empty (no running heads or page numbers), and headings (section titles in running heads)

**papersize** paper size, e.g. letter, a4

**secnumdepth** numbering depth for sections (with --number-sections option or number-sections variable)

---

**Fonts**

**fontenc** allows font encoding to be specified through fontenc package (with pdflatex); default is T1 (see LaTeX font encodings guide)

**fontfamily** font package for use with pdflatex: TeX Live includes many options, documented in the LaTeX Font Catalogue. The default is Latin Modern.

**fontfamilyoptions** options for package used as fontfamily; repeat for multiple options. For example, to use the Libertine font with proportional lowercase (old-style) figures through the libertinus package:

```latex
---
fontfamily: libertinus
fontfamilyoptions:
- osf
- p
...
```
**Templates**

**fontsize** font size for body text. The standard classes allow 10pt, 11pt, and 12pt. To use another size, set `documentclass` to one of the KOMA-Script classes, such as `scrartcl` or `scrbook`.

**mainfont, sansfont, monofont, mathfont, CJKmainfont** font families for use with `xelatex` or `lualatex`: take the name of any system font, using the `fontspec` package. `CJKmainfont` uses the `xecjk` package.

**mainfontoptions, sansfontoptions, monofontoptions, mathfontoptions, CJKoptions** options to use with `mainfont`, `sansfont`, `monofont`, `mathfont`, `CJKmainfont` in `xelatex` and `lualatex`. Allow for any choices available through `fontspec`; repeat for multiple options. For example, to use the TeX Gyre version of Palatino with lowercase figures:

```---
mainfont: TeX Gyre Pagella
mainfontoptions:
- Numbers=Lowercase
- Numbers=Proportional
...```

**microtypeoptions** options to pass to the `microtype` package

**Links**

**colorlinks** add color to link text; automatically enabled if any of `linkcolor`, `filecolor`, `citecolor`, `urlcolor`, or `toccolor` are set.

**linkcolor, filecolor, citecolor, urlcolor, toccolor** color for internal links, external links, citation links, linked URLs, and links in table of contents, respectively: uses options allowed by `xcolor`, including the `dvipsnames`, `svgnames`, and `x11names` lists.

**links-as-notes** causes links to be printed as footnotes

**Front matter**

**lof, lot** include list of figures, list of tables.

**thanks** contents of acknowledgments footnote after document title.

**toc** include table of contents (can also be set using `---toc`/`---table-of-contents`).

**toc-depth** level of section to include in table of contents.

**BibLaTeX Bibliographies**

These variables function when using BibLaTeX for citation rendering.

**biblatexoptions** list of options for `biblatex`
Variables for ConTeXt

\texttt{biblio-style} bibliography style, when used with \texttt{--natbib} and \texttt{--bibtex}.
\texttt{biblio-title} bibliography title, when used with \texttt{--natbib} and \texttt{--bibtex}.
\texttt{bibliography} bibliography to use for resolving references
\texttt{natbiboptions} list of options for natbib

\textbf{Variables for ConTeXt}

Pandoc uses these variables when creating a PDF with ConTeXt.

\texttt{fontsize} font size for body text (e.g. 10pt, 12pt)
\texttt{headertext}, \texttt{footertext} text to be placed in running header or footer (see ConTeXt Headers and Footers); repeat up to four times for different placement
\texttt{indenting} controls indentation of paragraphs, e.g. yes, small, next (see ConTeXt Indentation); repeat for multiple options
\texttt{interlinespace} adjusts line spacing, e.g. 4ex (using setupinterlinespace); repeat for multiple options
\texttt{layout} options for page margins and text arrangement (see ConTeXt Layout); repeat for multiple options
\texttt{linkcolor}, \texttt{contrastcolor} color for links outside and inside a page, e.g. red, blue (see ConTeXt Color)
\texttt{linkstyle} typeface style for links, e.g. normal, bold, slanted, boldslanted, type, cap, small
\texttt{lof}, \texttt{lot} include list of figures, list of tables
\texttt{mainfont}, \texttt{sansfont}, \texttt{monofont}, \texttt{mathfont} font families: take the name of any system font (see ConTeXt Font Switching)
\texttt{margin-left}, \texttt{margin-right}, \texttt{margin-top}, \texttt{margin-bottom} sets margins, if layout is not used (otherwise layout overrides these)
\texttt{pagenumbering} page number style and location (using setuppagenumbering); repeat for multiple options
\texttt{papersize} paper size, e.g. letter, A4, landscape (see ConTeXt Paper Setup); repeat for multiple options
\texttt{pdfa} adds to the preamble the setup necessary to generate PDF/A-1b:2005. To successfully generate PDF/A the required ICC color profiles have to be available and the content and all included files (such as images) have to be standard conforming. The ICC profiles can be obtained from ConTeXt ICC Profiles. See also ConTeXt PDFA for more details.
\texttt{toc} include table of contents (can also be set using \texttt{--toc/--table-of-contents})
\texttt{whitespace} spacing between paragraphs, e.g. none, small (using setupwhitespace)
Variables for wkhtmltopdf

Pandoc uses these variables when creating a PDF with wkhtmltopdf. The --css option also affects the output.

*footer-html, header-html* add information to the header and footer

*margin-left, margin-right, margin-top, margin-bottom* set the page margins

*papersize* sets the PDF paper size

Variables for man pages

*adjusting* adjusts text to left (l), right (r), center (c), or both (b) margins

*footer* footer in man pages

*header* header in man pages

*hyphenate* if true (the default), hyphenation will be used

*section* section number in man pages

Variables for ms

*fontfamily* font family (e.g. T or P)

*indent* paragraph indent (e.g. 2m)

*lineheight* line height (e.g. 12p)

*pagesize* point size (e.g. 10p)

Structural variables

Pandoc sets these variables automatically in response to options or document contents; users can also modify them. These vary depending on the output format, and include the following:

*body* body of document

*date-meta* the date variable converted to ISO 8601 YYYY-MM-DD, included in all HTML based formats (dzslides, epub, html4, html5, revealjs, s5, slideous, slidy). The recognized formats for date are: mm/dd/yyyy, mm/dd/yy, yyyy-mm-dd (ISO 8601), dd MM yyyy (e.g. either 02 Apr 2018 or 02 April 2018), MM dd, yyyy (e.g. Apr. 02, 2018 or April 02, 2018), yyyy[mm[dd]](e.g. 20180402, 201804 or 2018).

*header-includes* contents specified by -H/--include-in-header (may have multiple values)
Using variables in templates

`include-before` contents specified by `-B/--include-before-body` (may have multiple values)

`include-after` contents specified by `-A/--include-after-body` (may have multiple values)

`meta-json` JSON representation of all of the document’s metadata. Field values are transformed to the selected output format.

`numbersections` non-null value if `-N/--number-sections` was specified

`sourcefile`, `outputfile` source and destination filenames, as given on the command line.

sourcefile can also be a list if input comes from multiple files, or empty if input is from stdin. You can use the following snippet in your template to distinguish them:

```
$if(sourcefile)$$
$for(sourcefile)$
$sourcefile$
$endfor$
$else$
(stdin)
$endif$
```

Similarly, `outputfile` can be – if output goes to the terminal.

If you need absolute paths, use e.g. `$curdir$/sourcefile$`.

`curdir` working directory from which pandoc is run.

`toc` non-null value if `--toc/--table-of-contents` was specified

`toc-title` title of table of contents (works only with EPUB, oendocument, odt, docx, pptx, beamer, LaTeX)

Using variables in templates

Variable names are sequences of alphanumerics, -, and _ starting with a letter. A variable name surrounded by $ signs will be replaced by its value. For example, the string $title$ in

```<title>$title$</title>```

will be replaced by the document title.

To write a literal $ in a template, use $$.

Templates may contain conditionals. The syntax is as follows:
$if(variable)$
X
$else$
Y
$endif$

This will include X in the template if variable has a truthy value; otherwise it will include Y. Here a truthy value is any of the following:

- a string that is not entirely white space,
- a non-empty array where the first value is truthy,
- any number (including zero),
- any object,
- the boolean true (to specify the boolean true value using YAML metadata or the --metadata flag, use true, True, or TRUE; with the --variable flag, simply omit a value for the variable, e.g. --variable draft).

X and Y are placeholders for any valid template text, and may include interpolated variables or other conditionals. The $else$ section may be omitted.

When variables can have multiple values (for example, author in a multi-author document), you can use the $for$ keyword:

$for(author)$$author$$sep$, $endfor$

You can optionally specify a separator to be used between consecutive items:

$for(author)$$author$$sep$$, $endfor$

Note that the separator needs to be specified immediately before the $endfor$ keyword.

A dot can be used to select a field of a variable that takes an object as its value. So, for example:

$author.name$ ($author.affiliation$)

The value of a variable will be indented to the same level as the variable.

If you use custom templates, you may need to revise them as pandoc changes. We recommend tracking the changes in the default templates, and modifying your custom templates accordingly. An easy way to do this is to fork the pandoc-templates repository and merge in changes after each pandoc release.

Templates may contain comments: anything on a line after $--$ will be treated as a comment and ignored.
Extensions

The behavior of some of the readers and writers can be adjusted by enabling or disabling various extensions.

An extension can be enabled by adding +EXTENSION to the format name and disabled by adding -EXTENSION. For example, --from markdown_strict+footnotes is strict Markdown with footnotes enabled, while --from markdown-footnotes-pipe_tables is pandoc’s Markdown without footnotes or pipe tables.

The markdown reader and writer make by far the most use of extensions. Extensions only used by them are therefore covered in the section Pandoc’s Markdown below (See Markdown variants for commonmark and gfm.) In the following, extensions that also work for other formats are covered.

Note that markdown extensions added to the ipynb format affect Markdown cells in Jupyter notebooks (as do command-line options like --atx-headers).

Typography

Extension: smart

Interpret straight quotes as curly quotes, --- as em-dashes, -- as en-dashes, and ... as ellipses. Nonbreaking spaces are inserted after certain abbreviations, such as “Mr.”

This extension can be enabled/disabled for the following formats:

input formats markdown, commonmark, latex, mediawiki, org, rst, twiki
output formats markdown, latex, context, rst
enabled by default in markdown, latex, context (both input and output)

Note: If you are writing Markdown, then the smart extension has the reverse effect: what would have been curly quotes comes out straight.

In LaTeX, smart means to use the standard TeX ligatures for quotation marks (’` and ’ for double quotes, ` and ’ for single quotes) and dashes (-- for en-dash and --- for em-dash). If smart is disabled, then in reading LaTeX pandoc will parse these characters literally. In writing LaTeX, enabling smart tells pandoc to use the ligatures when possible; if smart is disabled pandoc will use unicode quotation mark and dash characters.
**Headings and sections**

**Extension: auto_identifiers**

A heading without an explicitly specified identifier will be automatically assigned a unique identifier based on the heading text.

This extension can be enabled/disabled for the following formats:

**input formats** markdown, latex, rst, mediawiki, textile
**output formats** markdown, muse
**enabled by default in** markdown, muse

The default algorithm used to derive the identifier from the heading text is:

- Remove all formatting, links, etc.
- Remove all footnotes.
- Remove all non-alphanumeric characters, 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,

<table>
<thead>
<tr>
<th>Heading</th>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heading identifiers in HTML</td>
<td>heading-identifiers-in-html</td>
</tr>
<tr>
<td>Maître d'hôtel</td>
<td>maitre-dhôtel</td>
</tr>
<tr>
<td><em>Dogs</em>?--in <em>my</em> house?</td>
<td>dogs--in-my-house</td>
</tr>
<tr>
<td>[HTML], [S5], or [RTF]?</td>
<td>html-s5-or-rtf</td>
</tr>
<tr>
<td>3. Applications</td>
<td>applications</td>
</tr>
<tr>
<td>33</td>
<td>section</td>
</tr>
</tbody>
</table>

These rules should, in most cases, allow one to determine the identifier from the heading text. The exception is when several headings 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.

(However, a different algorithm is used if `gfm_auto_identifiers` is enabled; see below.)

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 [heading identifiers](#heading-identifiers-in-html-latex-and-context).

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

If the `--section-divs` option is specified, then each section will be wrapped in a section (or a div, if html4 was specified), and the identifier will be attached to the enclosing `<section>` (or `<div>`) tag rather than the heading itself. This allows entire sections to be manipulated using JavaScript or treated differently in CSS.

**Extension: ascii_identifiers**

Causes the identifiers produced by auto_identifiers to be pure ASCII. Accents are stripped off of accented Latin letters, and non-Latin letters are omitted.

**Extension: gfm_auto_identifiers**

Changes the algorithm used by auto_identifiers to conform to GitHub’s method. Spaces are converted to dashes (–), uppercase characters to lowercase characters, and punctuation characters other than – and _ are removed.

**Math Input**

The extensions `tex_math_dollars`, `tex_math_single_backslash`, and `tex_math_double_backslash` are described in the section about Pandoc’s Markdown.

However, they can also be used with HTML input. This is handy for reading web pages formatted using MathJax, for example.

**Raw HTML/TeX**

The following extensions (especially how they affect Markdown input/output) are also described in more detail in their respective sections of Pandoc’s Markdown.

**Extension: raw_html**

When converting from HTML, parse elements to raw HTML which are not representable in pandoc’s AST. By default, this is disabled for HTML input.
Extensions

Extension: raw_tex

Allows raw LaTeX, TeX, and ConTeXt to be included in a document.

This extension can be enabled/disabled for the following formats (in addition to markdown):

**input formats** latex, org, textile, html (environments, \ref, and \eqref only), ipynb
**output formats** textile, commonmark

Note: as applied to ipynb, raw_html and raw_tex affect not only raw TeX in markdown cells, but data with mime type text/html in output cells. Since the ipynb reader attempts to preserve the richest possible outputs when several options are given, you will get best results if you disable raw_html and raw_tex when converting to formats like docx which don’t allow raw html or tex.

Extension: native_divs

This extension is enabled by default for HTML input. This means that divs are parsed to pandoc native elements. (Alternatively, you can parse them to raw HTML using -f html-native_divs+raw_html.)

When converting HTML to Markdown, for example, you may want to drop all divs and spans:

```
pandoc -f html-native_divs-native_spans -t markdown
```

Extension: native_spans

Analogous to native_divs above.

Literate Haskell support

Extension: literate_haskell

Treat the document as literate Haskell source.

This extension can be enabled/disabled for the following formats:

**input formats** markdown, rst, latex
**output formats** markdown, rst, latex, html

If you append +lhs (or +literate_haskell) to one of the formats above, pandoc will treat the document as literate Haskell source. This means that
Other extensions

- In Markdown input, “bird track” sections will be parsed as Haskell code rather than block quotations. Text between `$begin{code}` and `$end{code}` will also be treated as Haskell code. For ATX-style headings the character ‘=’ will be used instead of ‘#’.

- In Markdown output, code blocks with classes `haskell` and `literate` will be rendered using bird tracks, and block quotations will be indented one space, so they will not be treated as Haskell code. In addition, headings will be rendered setext-style (with underlines) rather than ATX-style (with ‘#’ characters). (This is because ghc treats ‘#’ characters in column 1 as introducing line numbers.)

- In restructured text input, “bird track” sections will be parsed as Haskell code.

- In restructured text output, code blocks with class `haskell` will be rendered using bird tracks.

- In LaTeX input, text in `code` environments will be parsed as Haskell code.

- In LaTeX output, code blocks with class `haskell` will be rendered inside `code` environments.

- In HTML output, code blocks with class `haskell` will be rendered with class `literatehaskell` and bird tracks.

Examples:

```
pandoc --f markdown+lhs --t html
```
reads literate Haskell source formatted with Markdown conventions and writes ordinary HTML (without bird tracks).

```
pandoc --f markdown+lhs --t html+lhs
```
writes HTML with the Haskell code in bird tracks, so it can be copied and pasted as literate Haskell source.

Note that GHC expects the bird tracks in the first column, so indented literate code blocks (e.g. inside an itemized environment) will not be picked up by the Haskell compiler.

Other extensions

**Extension: empty_paragraphs**

Allows empty paragraphs. By default empty paragraphs are omitted.
This extension can be enabled/disabled for the following formats:

**Input formats** docx, html
**Output formats** docx, odt, opendocument, html
Extensions

Extension: styles

When converting from docx, read all docx styles as divs (for paragraph styles) and spans (for character styles) regardless of whether pandoc understands the meaning of these styles. This can be used with docx custom styles. Disabled by default.

input formats docx

Extension: amuse

In the muse input format, this enables Text::Amuse extensions to Emacs Muse markup.

Extension: citations

Some aspects of Pandoc’s Markdown citation syntax are also accepted in org input.

Extension: ntbd

In the context output format this enables the use of Natural Tables (TABLE) instead of the default Extreme Tables (xtables). Natural tables allow more fine-grained global customization but come at a performance penalty compared to extreme tables.
Pandoc’s Markdown

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 using the `markdown_strict` format instead of `markdown`. Extensions can be enabled or disabled to specify the behavior more granularly. They are described in the following. See also Extensions above, for extensions that work also on other formats.

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 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 above.

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
Pandoc’s Markdown

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}

**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,

```java
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)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```
Pandoc's Markdown

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:

```html
<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, below.


**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:
Pandoc's Markdown

* 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
Lists

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.¹

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:

¹The 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) 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.²

Term 1

: Definition 1

Term 2 with *inline markup*

: Definition 2

    { some code, part of Definition 2 }

    Third paragraph of definition 2.

²I have been influenced by the suggestions of David Wheeler.
Pandoc’s Markdown

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:
Lists

(@) 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.)
**Pandoc’s Markdown**

**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:

<table>
<thead>
<tr>
<th>Right</th>
<th>Left</th>
<th>Center</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>123</td>
<td>123</td>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

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: 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.

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

3This scheme is due to Michel Fortin, who proposed it on the Markdown discussion list.
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:

<table>
<thead>
<tr>
<th>Centered Header Aligned</th>
<th>Default Aligned</th>
<th>Right Aligned</th>
<th>Left Aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>First row</td>
<td>12.0</td>
<td>Example of a row that spans multiple lines.</td>
<td></td>
</tr>
<tr>
<td>Second row</td>
<td>5.0</td>
<td>Here's another one. Note the blank line between rows.</td>
<td></td>
</tr>
</tbody>
</table>

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:
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.

```
<table>
<thead>
<tr>
<th>Fruit</th>
<th>Price</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bananas</td>
<td>$1.34</td>
<td>- built-in wrapper</td>
</tr>
<tr>
<td>Oranges</td>
<td>$2.10</td>
<td>- cures scurvy</td>
</tr>
</tbody>
</table>
```

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.

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

```
<table>
<thead>
<tr>
<th>Right</th>
<th>Left</th>
<th>Centered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bananas</td>
<td>$1.34</td>
<td>built-in wrapper</td>
</tr>
</tbody>
</table>
```
For headerless tables, the colons go on the top line instead:

```
<table>
<thead>
<tr>
<th align="right">Right</th>
<th align="left">Left</th>
<th align="right">Centered</th>
</tr>
</thead>
<tbody>
<tr>
<td align="right">123</td>
<td align="left">123</td>
<td align="right">123</td>
</tr>
<tr>
<td align="right">1</td>
<td align="left">1</td>
<td align="right">1</td>
</tr>
</tbody>
</table>
```

**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:

```
<table>
<thead>
<tr>
<th align="right">Right</th>
<th align="left">Left</th>
<th>Default</th>
<th align="center">Center</th>
</tr>
</thead>
<tbody>
<tr>
<td align="right">------:</td>
<td align="left">:-----</td>
<td>---------</td>
<td align="center">:------:</td>
</tr>
<tr>
<td align="right">12</td>
<td align="left">12</td>
<td>12</td>
<td align="center"></td>
</tr>
<tr>
<td align="right">123</td>
<td align="left">123</td>
<td>123</td>
<td align="center"></td>
</tr>
<tr>
<td align="right">1</td>
<td align="left">1</td>
<td>1</td>
<td align="center"></td>
</tr>
</tbody>
</table>
```

: 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:

```bash
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. The metadata fields will be combined through a left-biased union: if two metadata blocks attempt to set the same field, the value from the first 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:

```yaml
---
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.
...`
Pandoc’s Markdown

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:

```html
<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:

```yaml
---
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:

```bash
$for(author)$
$if(author.name)$
$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:

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

64
Backslash escapes

Extension: all_symbols_escaped

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.
**Pandoc's Markdown**

**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,

\[ H_{2}O \text{ is a liquid. } 2^{10} \text{ is 1024.} \]

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 \(P_{a\text{ cat}}\), not \(P_{a\text{ cat}}\).

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}`
Pandoc's Markdown

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 inlinem–equation 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 above.

---

**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's Markdown

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

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

into

```html
<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 `\textless a>html</a>`{=html}
Pandoc’s Markdown

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 \texttt{--t/to}, above, for a list, or use \texttt{pandoc \--list-output-formats}). Use \texttt{openxml} for DOCX output, \texttt{oendocument} for ODT output, \texttt{html5} for EPUB3 output, \texttt{html4} for EPUB2 output, and \texttt{latex}, \texttt{beamer}, \texttt{ms}, or \texttt{html5} for PDF output (depending on what you use for \texttt{--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, \texttt{backtick_code_blocks} must be enabled.

The raw attribute cannot be combined with regular attributes.

\section*{LaTeX macros}

\textbf{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:

\begin{verbatim}
\newcommand{\tuple}[1]{\langle #1 \rangle}
\end{verbatim}

\begin{verbatim}$\tuple{a, b, c}$\end{verbatim}

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

When \texttt{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.

Whether or not \texttt{latex_macros} is enabled, the macro definitions will still be passed through as raw LaTeX.
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:
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:
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 attributes except width and height (but including srcset and sizes) are passed through as is. The other writers ignore attributes that are not 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.jpg}`
    (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.
**Pandoc’s Markdown**

**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:

<table>
<thead>
<tr>
<th>Format</th>
<th>File extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>BibLaTeX</td>
<td>.bib</td>
</tr>
<tr>
<td>BibTeX</td>
<td>.bibtex</td>
</tr>
<tr>
<td>Copac</td>
<td>.copac</td>
</tr>
<tr>
<td>CSL JSON</td>
<td>.json</td>
</tr>
<tr>
<td>CSL YAML</td>
<td>.yaml</td>
</tr>
<tr>
<td>EndNote</td>
<td>.enl</td>
</tr>
<tr>
<td>EndNote XML</td>
<td>.xml</td>
</tr>
<tr>
<td>ISI</td>
<td>.wos</td>
</tr>
<tr>
<td>MEDLINE</td>
<td>.medline</td>
</tr>
<tr>
<td>MODS</td>
<td>.mods</td>
</tr>
<tr>
<td>RIS</td>
<td>.ris</td>
</tr>
</tbody>
</table>

Note that .bib can be used with both BibTeX and BibLaTeX files; use .bibtex to force
Pandoc's Markdown

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
<span style="font-variant:small-caps;">...</span> or <sc>...</sc> small capitals
<s>...</s> subscript
<sup>...</sup> superscript
<sup class="nocase">...</sup> 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:

```yaml
---
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
```
Citations

(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]
Pandoc’s Markdown

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
Non-pandoc extensions

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.
Pandoc’s Markdown

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 \( \text{ 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 `<...>`.
Pandoc’s Markdown

**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.⁴

**Markdown variants**

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

---

⁴To see why laziness is incompatible with relaxing the requirement of a blank line between items, consider the following example:

```markdown
bar : definition
foo : definition
```

Is this a single list item with two definitions of “bar,” the first of which is lazily wrapped, or two list items? To remove the ambiguity we must either disallow lazy wrapping or require a blank line between list items.
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_reference_links.

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.
Producing slide shows with pandoc

You can use pandoc to produce an HTML + JavaScript slide presentation that can be viewed via a web browser. There are five ways to do this, using S5, DZSlides, Slidy, Slideous, or reveal.js. You can also produce a PDF slide show using LaTeX beamer, or slides shows in Microsoft PowerPoint format.

Here’s the Markdown source for a simple slide show, habits.txt:

```markdown
% Habits
% John Doe
% March 22, 2005

# In the morning

## Getting up

- Turn off alarm
- Get out of bed

## Breakfast

- Eat eggs
- Drink coffee

# In the evening

## Dinner

- Eat spaghetti
- Drink wine

------------------

![picture of spaghetti](images/spaghetti.jpg)

## Going to sleep
```
Producing slide shows with pandoc

- Get in bed
- Count sheep

To produce an HTML/JavaScript slide show, simply type

```
pandoc -t FORMAT -s habits.txt -o habits.html
```

where `FORMAT` is either `s5`, `slidy`, `slideous`, `dzslides`, or `revealjs`.

For Slidy, Slideous, reveal.js, and S5, the file produced by pandoc with the `-s/--standalone` option embeds a link to JavaScript and CSS files, which are assumed to be available at the relative path `s5/default` (for S5), `slideous` (for Slideous), `reveal.js` (for reveal.js), or at the Slidy website at `w3.org` (for Slidy). (These paths can be changed by setting the `slidy-url`, `slideous-url`, `revealjs-url`, or `s5-url` variables; see Variables for HTML slides, above.) For DZSlides, the (relatively short) JavaScript and CSS are included in the file by default.

With all HTML slide formats, the `--self-contained` option can be used to produce a single file that contains all of the data necessary to display the slide show, including linked scripts, stylesheets, images, and videos.

To produce a PDF slide show using beamer, type

```
pandoc -t beamer habits.txt -o habits.pdf
```

Note that a reveal.js slide show can also be converted to a PDF by printing it to a file from the browser.

To produce a Powerpoint slide show, type

```
pandoc habits.txt -o habits.pptx
```

**Structuring the slide show**

By default, the *slide level* is the highest heading level in the hierarchy that is followed immediately by content, and not another heading, somewhere in the document. In the example above, level-1 headings are always followed by level-2 headings, which are followed by content, so the slide level is 2. This default can be overridden using the `--slide-level` option.

The document is carved up into slides according to the following rules:

- A horizontal rule always starts a new slide.
- A heading at the slide level always starts a new slide.
• Headings below the slide level in the hierarchy create headings within a slide.

• Headings above the slide level in the hierarchy create “title slides,” which just contain the section title and help to break the slide show into sections. Non-slide content under these headings will be included on the title slide (for HTML slide shows) or in a subsequent slide with the same title (for beamer).

• A title page is constructed automatically from the document’s title block, if present. (In the case of beamer, this can be disabled by commenting out some lines in the default template.)

These rules are designed to support many different styles of slide show. If you don’t care about structuring your slides into sections and subsections, you can just use level-1 headings for all each slide. (In that case, level-1 will be the slide level.) But you can also structure the slide show into sections, as in the example above.

Note: in reveal.js slide shows, if slide level is 2, a two-dimensional layout will be produced, with level-1 headings building horizontally and level-2 headings building vertically. It is not recommended that you use deeper nesting of section levels with reveal.js.

**Incremental lists**

By default, these writers produce lists that display “all at once.” If you want your lists to display incrementally (one item at a time), use the -i option. If you want a particular list to depart from the default, put it in a div block with class incremental or nonincremental. So, for example, using the fenced div syntax, the following would be incremental regardless of the document default:

```plaintext
::: incremental
- Eat spaghetti
- Drink wine

:::
```

or

```plaintext
::: nonincremental
- Eat spaghetti
- Drink wine

:::
```
Producing slide shows with pandoc

While using incremental and nonincremental divs are the recommended method of setting incremental lists on a per-case basis, an older method is also supported: putting lists inside a blockquote will depart from the document default (that is, it will display incrementally without the -i option and all at once with the -i option):

```
> - Eat spaghetti
> - Drink wine
```

Both methods allow incremental and nonincremental lists to be mixed in a single document.

**Inserting pauses**

You can add “pauses” within a slide by including a paragraph containing three dots, separated by spaces:

```
# Slide with a pause

content before the pause

...  

content after the pause
```

**Styling the slides**

You can change the style of HTML slides by putting customized CSS files in $DATADIR/s5/default (for S5), $DATADIR/slidy (for Slidy), or $DATADIR/slideous (for Slideous), where $DATADIR is the user data directory (see --data-dir, above). The originals may be found in pandoc’s system data directory (generally $CABALDIR/pandoc-VERSION/s5/default). Pandoc will look there for any files it does not find in the user data directory.

For dzslides, the CSS is included in the HTML file itself, and may be modified there.

All reveal.js configuration options can be set through variables. For example, themes can be used by setting the theme variable:

```
-V theme=moon
```
Or you can specify a custom stylesheet using the --css option.

To style beamer slides, you can specify a theme, colortheme, fonttheme, innertheme, and outertheme, using the -V option:

```
pandoc -t beamer habits.txt -V theme:Warsaw -o habits.pdf
```

Note that heading attributes will turn into slide attributes (on a <div> or <section>) in HTML slide formats, allowing you to style individual slides. In beamer, the only heading attribute that affects slides is the allowframebreaks class, which sets the allowframebreaks option, causing multiple slides to be created if the content overfills the frame. This is recommended especially for bibliographies:

```
# References {.allowframebreaks}
```

**Speaker notes**

Speaker notes are supported in reveal.js and PowerPoint (pptx) output. You can add notes to your Markdown document thus:

```::: notes
This is my note.
- It can contain Markdown
  - like this list
:::
```

To show the notes window in reveal.js, press s while viewing the presentation. Speaker notes in PowerPoint will be available, as usual, in handouts and presenter view.

Notes are not yet supported for other slide formats, but the notes will not appear on the slides themselves.

**Columns**

To put material in side by side columns, you can use a native div container with class columns, containing two or more div containers with class column and a width attribute:
Frame attributes in beamer

Sometimes it is necessary to add the LaTeX \texttt{[fragile]} option to a frame in beamer (for example, when using the \texttt{minted} environment). This can be forced by adding the \texttt{fragile} class to the heading introducing the slide:

```
# Fragile slide {.fragile}
```

All of the other frame attributes described in Section 8.1 of the Beamer User’s Guide may also be used: \texttt{allowdisplaybreaks}, \texttt{allowframebreaks}, \texttt{b}, \texttt{c}, \texttt{t}, \texttt{environment}, \texttt{label}, \texttt{plain}, \texttt{shrink}, \texttt{standout}, \texttt{noframenumbering}.

Background in reveal.js and beamer

Background images can be added to self-contained reveal.js slideshows and to beamer slideshows.

For the same image on every slide, use the configuration option \texttt{background-image} either in the YAML metadata block or as a command-line variable. (There are no other options in beamer and the rest of this section concerns reveal.js slideshows.)

For reveal.js, you can instead use the reveal.js-native option \texttt{parallaxBackgroundImage}. You can also set \texttt{parallaxBackgroundHorizontal} and \texttt{parallaxBackgroundVertical} the same way and must also set \texttt{parallaxBackgroundSize} to have your values take effect.

To set an image for a particular reveal.js slide, add \texttt{\{data-background-image="/path/to/image"\}} to the first slide-level heading on the slide (which may even be empty).

In reveal.js’s overview mode, the \texttt{parallaxBackgroundImage} will show up only on the first slide.

Other reveal.js background settings also work on individual slides, including \texttt{data-background-size}, \texttt{data-background-repeat}, \texttt{data-background-color}, \texttt{data-transition}, and \texttt{data-transition-speed}. 
See the reveal.js documentation for more details.

For example in reveal.js:

```---
title: My Slideshow
parallaxBackgroundImage: /path/to/my/background_image.png
---
```

## Slide One

Slide 1 has background_image.png as its background.

## {data-background-image="/path/to/special_image.jpg"}

Slide 2 has a special image for its background, even though the heading has no content.
Creating EPUBs with pandoc

EPUB Metadata

EPUB metadata may be specified using the --epub-metadata option, but if the source document is Markdown, it is better to use a YAML metadata block. Here is an example:

```yaml
---
title:
  - type: main
text: My Book
  - type: subtitle
text: An investigation of metadata
creator:
  - role: author
text: John Smith
  - role: editor
text: Sarah Jones
identifier:
  - scheme: DOI
text: doi:10.234234.234/33
publisher: My Press
rights: © 2007 John Smith, CC BY-NC
ibooks:
  version: 1.3.4
...```

The following fields are recognized:


**title** Either a string value, or an object with fields file-as and type, or a list of such objects. Valid values for type are main, subtitle, short, collection, edition, extended.
Creating EPUBs with pandoc

**creator** Either a string value, or an object with fields role, file-as, and text, or a list of such objects. Valid values for role are MARC relators, but pandoc will attempt to translate the human-readable versions (like “author” and “editor”) to the appropriate marc relators.

**contributor** Same format as creator.

**date** A string value in YYYY-MM-DD format. (Only the year is necessary.) Pandoc will attempt to convert other common date formats.

**lang (or legacy: language)** A string value in BCP 47 format. Pandoc will default to the local language if nothing is specified.

**subject** A string value or a list of such values.

**description** A string value.

**type** A string value.

**format** A string value.

**relation** A string value.

**coverage** A string value.

**rights** A string value.

**cover-image** A string value (path to cover image).

**css (or legacy: stylesheet)** A string value (path to CSS stylesheet).

**page-progression-direction** Either ltr or rtl. Specifies the page-progression-direction attribute for the spine element.

**ibooks** iBooks-specific metadata, with the following fields:

- version: (string)
- specified-fonts: true|false (default false)
- ipad-orientation-lock: portrait-only|landscape-only
- iphone-orientation-lock: portrait-only|landscape-only
- binding: true|false (default true)
- scroll-axis: vertical|horizontal|default

**The epub:type attribute**

For epub3 output, you can mark up the heading that corresponds to an EPUB chapter using the epub:type attribute. For example, to set the attribute to the value prologue, use this markdown:

```
# My chapter {epub:type=prologue}
```
Which will result in:

```xml
<body epub:type="frontmatter">
  <section epub:type="prologue">
    <h1>My chapter</h1>
  </section>
</body>
```

Pandoc will output `<body epub:type="bodymatter">`, unless you use one of the following values, in which case either `frontmatter` or `backmatter` will be output.

<table>
<thead>
<tr>
<th><code>epub:type</code> of first section</th>
<th><code>epub:type</code> of body</th>
</tr>
</thead>
<tbody>
<tr>
<td>prologue</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>abstract</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>acknowledgments</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>copyright-page</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>dedication</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>credits</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>keywords</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>imprint</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>contributors</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>other-credits</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>errata</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>revision-history</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>titlepage</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>halftitlepage</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>seriespage</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>foreword</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>preface</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>seriespage</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>titlepage</td>
<td><code>frontmatter</code></td>
</tr>
<tr>
<td>appendix</td>
<td><code>backmatter</code></td>
</tr>
<tr>
<td>colophon</td>
<td><code>backmatter</code></td>
</tr>
<tr>
<td>bibliography</td>
<td><code>backmatter</code></td>
</tr>
<tr>
<td>index</td>
<td><code>backmatter</code></td>
</tr>
</tbody>
</table>

**Linked media**

By default, pandoc will download media referenced from any `<img>`, `<audio>`, `<video>` or `<source>` element present in the generated EPUB, and include it in the EPUB container, yielding a completely self-contained EPUB. If you want to link to external media resources instead, use raw HTML in your source and add `data-external="1"` to the tag with the `src` attribute. For example:
Creating EPUBs with pandoc

<audio controls="1">
  <source src="http://example.com/music/toccata.mp3"
data-external="1" type="audio/mpeg">
</source>
</audio>
Creating Jupyter notebooks with pandoc

When creating a Jupyter notebook, pandoc will try to infer the notebook structure. Code blocks with the class code will be taken as code cells, and intervening content will be taken as Markdown cells. Attachments will automatically be created for images in Markdown cells. Metadata will be taken from the jupyter metadata field. For example:

```yaml
---
title: My notebook
jupyter:
  nbformat: 4
  nbformat_minor: 5
  kernelspec:
    display_name: Python 2
    language: python
    name: python2
  language_info:
    codemirror_mode:
      name: ipython
      version: 2
    file_extension: "".py"
    mimetype: "text/x-python"
    name: "python"
    nbconvert_exporter: "python"
    pygments_lexer: "ipython2"
    version: "2.7.15"
---

# Lorem ipsum

**Lorem ipsum** dolor sit amet, consectetur adipiscing elit. Nunc luctus bibendum felis dictum sodales.

``` code
print("hello")
```
Creating Jupyter notebooks with pandoc

## Pyout

``` code
from IPython.display import HTML
HTML(""
<script>
console.log("hello");
</script>
<b>HTML</b>
"")
```

## Image

This image ![image](myimage.png) will be included as a cell attachment.

If you want to add cell attributes, group cells differently, or add output to code cells, then you need to include divs to indicate the structure. You can use either fenced divs or native divs for this. Here is an example:

:::::: {.cell .markdown}
# Lorem

**Lorem ipsum** dolor sit amet, consectetur adipiscing elit. Nunc luctus bibendum felis dictum sodales.
::::::

:::::: {.cell .code execution_count=1}
``` {.python}
print("hello")
```
::: {.output .stream .stdout}
```
```
:::

:::::: {.cell .code execution_count=2}
``` {.python}
from IPython.display import HTML
```

102
If you include raw HTML or TeX in an output cell, use the [raw attribute][Extension: fenced_attribute], as shown in the last cell of the example above. Although pandoc can process “bare” raw HTML and TeX, the result is often interspersed raw elements and normal textual elements, and in an output cell pandoc expects a single, connected raw block. To avoid using raw HTML or TeX except when marked explicitly using raw attributes, we recommend specifying the extensions -raw_html-raw_tex+raw_attribute when translating between Markdown and ipynb notebooks.

Note that options and extensions that affect reading and writing of Markdown will also affect Markdown cells in ipynb notebooks. For example, --wrap=preserve will preserve soft line breaks in Markdown cells; --atx-headers will cause ATX-style headings to be used; and --preserve-tabs will prevent tabs from being turned to spaces.
Syntax highlighting

Pandoc will automatically highlight syntax in fenced code blocks that are marked with a language name. The Haskell library skylighting is used for highlighting. Currently highlighting is supported only for HTML, EPUB, Docx, Ms, and LaTeX/PDF output. To see a list of language names that pandoc will recognize, type `pandoc --list-highlight-languages`.

The color scheme can be selected using the `--highlight-style` option. The default color scheme is `pygments`, which imitates the default color scheme used by the Python library `pygments` (though `pygments` is not actually used to do the highlighting). To see a list of highlight styles, type `pandoc --list-highlight-styles`.

If you are not satisfied with the predefined styles, you can use `--print-highlight-style` to generate a JSON `.theme` file which can be modified and used as the argument to `--highlight-style`. To get a JSON version of the `pygments` style, for example:

```
pandoc --print-highlight-style pygments > my.theme
```

Then edit `my.theme` and use it like this:

```
pandoc --highlight-style my.theme
```

If you are not satisfied with the built-in highlighting, or you want highlight a language that isn't supported, you can use the `--syntax-definition` option to load a KDE-style XML syntax definition file. Before writing your own, have a look at KDE’s repository of syntax definitions.

To disable highlighting, use the `--no-highlight` option.
Custom Styles

Custom styles can be used in the docx and ICML formats.

Output

By default, pandoc’s docx and ICML output applies a predefined set of styles for blocks such as paragraphs and block quotes, and uses largely default formatting (italics, bold) for inlines. This will work for most purposes, especially alongside a reference.docx file. However, if you need to apply your own styles to blocks, or match a preexisting set of styles, pandoc allows you to define custom styles for blocks and text using divs and spans, respectively.

If you define a div or span with the attribute custom-style, pandoc will apply your specified style to the contained elements. So, for example using the bracketed_spans syntax,

[Get out]{custom-style="Emphatically"}, he said.

would produce a docx file with “Get out” styled with character style Emphatically. Similarly, using the fenced_divs syntax,

Dickinson starts the poem simply:

::: {custom-style="Poetry"}
| A Bird came down the Walk---
| He did not know I saw---
:::

would style the two contained lines with the Poetry paragraph style.

For docx output, styles will be defined in the output file as inheriting from normal text, if the styles are not yet in your reference.docx. If they are already defined, pandoc will not alter the definition.

This feature allows for greatest customization in conjunction with pandoc filters. If you want all paragraphs after block quotes to be indented, you can write a filter to apply the styles necessary. If you want all italics to be transformed to the Emphasis character style
(perhaps to change their color), you can write a filter which will transform all italicized inlines to inlines within an **Emphasis** custom-style span.

For docx output, you don’t need to enable any extensions for custom styles to work.

**Input**

The docx reader, by default, only reads those styles that it can convert into pandoc elements, either by direct conversion or interpreting the derivation of the input document’s styles.

By enabling the `styles` extension in the docx reader (`-f docx+styles`), you can produce output that maintains the styles of the input document, using the custom-style class. Paragraph styles are interpreted as divs, while character styles are interpreted as spans.

For example, using the `custom-style-reference.docx` file in the test directory, we have the following different outputs:

Without the `+styles` extension:

```
$ pandoc test/docx/custom-style-reference.docx -f docx -t markdown
This is some text.

This is text with an *emphasized* text style. And this is text with a **strengthened** text style.

> Here is a styled paragraph that inherits from Block Text.
```

And with the extension:

```
$ pandoc test/docx/custom-style-reference.docx -f docx+styles -t markdown

::: {custom-style="FirstParagraph"}
This is some text.
:::

::: {custom-style="BodyText"}
This is text with an [emphasized]{custom-style="Emphatic"} text style. And this is text with a [strengthened]{custom-style="Strengthened"} text style.
:::

::: {custom-style="MyBlockStyle"}
> Here is a styled paragraph that inherits from Block Text.
:::
```
With these custom styles, you can use your input document as a reference-doc while creating docx output (see below), and maintain the same styles in your input and output files.
Custom writers

Pandoc can be extended with custom writers written in lua. (Pandoc includes a lua interpreter, so lua need not be installed separately.)

To use a custom writer, simply specify the path to the lua script in place of the output format. For example:

```
pandoc -t data/sample.lua
```

Creating a custom writer requires writing a lua function for each possible element in a pandoc document. To get a documented example which you can modify according to your needs, do

```
pandoc --print-default-data-file sample.lua
```
A note on security

If you use pandoc to convert user-contributed content in a web application, here are some things to keep in mind:

1. Although pandoc itself will not create or modify any files other than those you explicitly ask it create (with the exception of temporary files used in producing PDFs), a filter or custom writer could in principle do anything on your file system. Please audit filters and custom writers very carefully before using them.

2. If your application uses pandoc as a Haskell library (rather than shelling out to the executable), it is possible to use it in a mode that fully isolates pandoc from your file system, by running the pandoc operations in the PandocPure monad. See the document Using the pandoc API for more details.

3. Pandoc’s parsers can exhibit pathological performance on some corner cases. It is wise to put any pandoc operations under a timeout, to avoid DOS attacks that exploit these issues. If you are using the pandoc executable, you can add the command line options +RTS -M512M -RTS (for example) to limit the heap size to 512MB.

4. The HTML generated by pandoc is not guaranteed to be safe. If raw_html is enabled for the Markdown input, users can inject arbitrary HTML. Even if raw_html is disabled, users can include dangerous content in attributes for headings, spans, and code blocks. To be safe, you should run all the generated HTML through an HTML sanitizer.
Authors

Copyright 2006–2019 John MacFarlane (jgm@berkeley.edu). Released under the GPL, version 2 or greater. This software carries no warranty of any kind. (See COPYRIGHT for full copyright and warranty notices.) For a full list of contributors, see the file AUTHORS.md in the pandoc source code.