Pandoc User's Guide

John MacFarlane

2026-11-10

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 . . . 12
- General writer options . . . 15
- Options affecting specific writers . . . 19
- Citation rendering . . . 27
- Math rendering . . . 28
- Options for wrapper scripts . . . 29

Exit codes 31

Defaults files 33
- General options . . . 34
- Reader options . . . 35
- General writer options . . . 35
- Options affecting specific writers . . . 36
- Citation rendering . . . 37
- Math rendering in HTML . . . 38
- Options for wrapper scripts . . . 38

Templates 39
- Template syntax . . . 40
  - Comments . . . 40
  - Delimiters . . . 40
  - Interpolated variables . . . 40
  - Conditionals . . . 41
  - For loops . . . 42
  - Partials . . . 43
  - Nesting . . . 44
  - Breakable spaces . . . 44
  - Pipes . . . 45

Variables . . . 46
Metadata variables . . . 46
Language variables . . . 47
Variables for HTML . . . 48
Variables for HTML math . . . 50
Variables for HTML slides . . . 50
Variables for Beamer slides . . . 50
Variables for PowerPoint . . . 51
Variables for LaTeX . . . 51
Variables for ConTeXt . . . 55
Variables for `wkhtmltopdf` . . . 56
Variables for man pages . . . 56
Variables for Texinfo . . . 57
Variables for Typst . . . 57
Variables for ms . . . 57
Variables for RTF . . . 58
Variables set automatically . . . 58

Extensions **61**
Typography . . . 61
Extension: `smart` . . . 61
Headings and sections . . . 62
Extension: `auto_identifiers` . . . 62
Extension: `ascii_identifiers` . . . 63
Extension: `gfm_auto_identifiers` . . . 63
Math Input . . . 63
Raw HTML/TeX . . . 63
Literate Haskell support . . . 64
Extension: `literate_haskell` . . . 64
Other extensions . . . 65
Extension: `empty_paragraphs` . . . 65
Extension: `native_numbering` . . . 65
Extension: `xrefs_name` . . . 66
Extension: `xrefs_number` . . . 66
Extension: `styles` . . . 66
Extension: `amuse` . . . 67
Extension: `raw_markdown` . . . 67
Extension: `citations` (typst) . . . 67
Extension: `citations` (org) . . . 67
Extension: `citations` (docx) . . . 67
Extension: `fancy_lists` (org) . . . 67
Extension: `element_citations` . . . 68
Extension: `ntb` . . . 68
Extension: `smart_quotes` (org) . . . 68
Extension: `special_strings` (org) . . . 68
Extension: `tagging` . . . 68

Pandoc's Markdown **69**
Philosophy . . . 69
Paragraphs . . . 69
Extension: `escaped_line_breaks` . . . 70
Headings . . . 70
Setext-style headings . . . 70
ATX-style headings . . . 70
Extension: `blank_before_header` . . . 71
Extension: `space_in_atx_header` . . . 71
Heading identifiers . . . 71
Extension: `header_attributes` . . . 71
Extension: `implicit_header_references` . . . 72
Block quotations . . . 73
Extension: `blank_before_blockquote` . . . 74
Verbatim (code) blocks . . . 74
Indented code blocks . . . 74
Fenced code blocks . . . 75
Extension: `fenced_code_blocks` . . . 75
Extension: `backtick_code_blocks` . . . 75
Extension: `fenced_code_attributes` . . . 75
Line blocks . . . 77
Extension: `line_blocks` . . . 77
Lists . . . 78
Bullet lists . . . 78
Block content in list items . . . 79
Ordered lists . . . 80
Extension: `fancy_lists` . . . 80
Extension: `startnum` . . . 81
Extension: `task_lists` . . . 82
Definition lists . . . 82
Extension: `definition_lists` . . . 82
Numbered example lists . . . 83
Extension: `example_lists` . . . 83
Ending a list . . . 84
Horizontal rules . . . 85
Tables . . . 85
Extension: `table_captions` . . . 85
Extension: `simple_tables` . . . 86
Extension: `multiline_tables` . . . 87
Extension: `grid_tables` . . . 88
Extension: `pipe_tables` . . . 90
Extension: `table_attributes` . . . 91
Table placement in LaTeX . . . 91
Metadata blocks . . . 91
Extension: `pandoc_title_block` . . . 91
Extension: `yaml_metadata_block` . . . 93
Backslash escapes . . . 96
Extension: `all_symbols_escapable` . . . 96

Inline formatting . . . 97
Emphasis . . . 97
Extension: intraword_underscores . . . 97
Strikeout . . . 97
Extension: strikeout . . . 97
Superscripts and subscripts . . . 98
Extension: superscript, subscript . . . 98
Verbatim . . . 98
Extension: inline_code_attributes . . . 98
Underline . . . 99
Small caps . . . 99
Highlighting . . . 99
Math . . . 100
Extension: tex_math_dollars . . . 100
Raw HTML . . . 101
Extension: raw_html . . . 101
Extension: markdown_in_html_blocks . . . 101
Extension: native_divs . . . 102
Extension: native_spans . . . 102
Extension: raw_tex . . . 102
Generic raw attribute . . . 103
Extension: raw_attribute . . . 103
LaTeX macros . . . 104
Extension: latex_macros . . . 104
Links . . . 104
Automatic links . . . 104
Inline links . . . 105
Reference links . . . 105
Extension: shortcut_reference_links . . . 106
Internal links . . . 107
Images . . . 107
Extension: implicit_figures . . . 107
Extension: link_attributes . . . 108
Divs and Spans . . . 109
Extension: fenced_divs . . . 109
Extension: bracketed_spans . . . 110
Footnotes . . . 110
Extension: footnotes . . . 110
Extension: inline_notes . . . 111
Citation syntax . . . 111
Extension: citations . . . 111
Non-default extensions . . . 113
Extension: rebase_relative_paths . . . 113
Extension: mark . . . 114
Extension: attributes . . . 114
Extension: old_dashes . . . 115
Extension: angle_brackets_escapable . . . 115
Extension: lists_without_preceding_blankline . . . 115

Extension: four_space_rule . . . 115
Extension: spaced_reference_links . . . 115
Extension: hard_line_breaks . . . 115
Extension: ignore_line_breaks . . . 115
Extension: east_asian_line_breaks . . . 116
Extension: emoji . . . 116
Extension: tex_math_gfm . . . 116
Extension: tex_math_single_backslash . . . 116
Extension: tex_math_double_backslash . . . 116
Extension: markdown_attribute . . . 116
Extension: mmd_title_block . . . 117
Extension: abbreviations . . . 117
Extension: alerts . . . 117
Extension: autolink_bare_uris . . . 117
Extension: mmd_link_attributes . . . 117
Extension: mmd_header_identifiers . . . 118
Extension: gutenberg . . . 118
Extension: sourcepos . . . 118
Extension: short_subsuperscripts . . . 118
Extension: wikilinks_title_after_pipe . . . 118
Markdown variants . . . 119

Citations **121**
Specifying bibliographic data . . . 121
Capitalization in titles . . . 123
Conference papers, published vs. unpublished . . . 124
Specifying a citation style . . . 124
Citations in note styles . . . 125
Placement of the bibliography . . . 125
Including uncited items in the bibliography . . . 126
Other relevant metadata fields . . . 127

Slide shows **129**
Structuring the slide show . . . 130
PowerPoint layout choice . . . 131
Incremental lists . . . 132
Inserting pauses . . . 133
Styling the slides . . . 133
Speaker notes . . . 134
Speaker notes on the title slide (PowerPoint) . . . 134
Columns . . . 135
Additional columns attributes in beamer . . . 135
Frame attributes in beamer . . . 136
Background in reveal.js, beamer, and pptx . . . 137
On all slides (beamer, reveal.js, pptx) . . . 137
On individual slides (reveal.js, pptx) . . . 137
On the title slide (reveal.js, pptx) . . . 138
Example (reveal.js) . . . 138

EPUBs **139**
EPUB Metadata . . . 139
The `epub:type` attribute . . . 141
Linked media . . . 142
EPUB styling . . . 142

Chunked HTML **145**

Jupyter notebooks **147**

Vimdoc **151**

Syntax highlighting **153**

Custom Styles **155**
Output . . . 155
Input . . . 156

Custom readers and writers **159**

Reproducible builds **161**

Accessible PDFs and PDF archiving standards **163**
LaTeX . . . 163
ConTeXt . . . 164
WeasyPrint . . . 164
Prince XML . . . 164
Typst . . . 164
Word Processors . . . 165

Running pandoc as a web server **167**

Running pandoc as a Lua interpreter **169**

A note on security **171**

Authors **173**

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`/`--output` 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:

```
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, roff ms, or HTML as an intermediate format. 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. The tool used to generate the PDF from the intermediate format may be specified using `--pdf-engine`.

You can control the PDF style using variables, depending on the intermediate format used: see variables for LaTeX, variables for ConTeXt, variables for `wkhtmltopdf`, variables for ms. When HTML is used as an intermediate format, the output can be styled using `--css`.

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`, `iftex`, `listings` (if the `--listings` option is used), `fancyvrb`, `longtable`, `caption`, `booktabs`, `multirow` (if the document contains a table with cells that cross multiple rows), `graphicx` (if the document contains images), `bookmark`, `xcolor`, `soul`, `geometry` (with the `geometry` variable set), `setspace` (with `linestretch`), and `babel` (with `lang`). If `CJKmainfont` is set, `xeCJK` is needed if `xelatex` is used, else `luatexja` is needed if `lualatex` is used. `framed` is required if code is highlighted in a scheme that uses a colored background. The use of `xelatex` or `lualatex` as the PDF engine requires `fontspec`. `lualatex` uses `selnolig` and `lua-ul`. `xelatex` uses `bidi` (with the `dir` variable set). If the `mathspec` variable is set, `xelatex` will use `mathspec` instead of `unicode-math`. The `csquotes` package will be used for typography if the `csquotes` variable or metadata field is set to a true value. The `natbib`, `biblatex`, `bibtex`, and `biber` packages can optionally be used for citation rendering. If math with `\cancel`, `\bcancel`, or `\xcancel` is used, the `cancel` package is needed. 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), 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:

```
pandoc -f html -t markdown https://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" \
  https://www.fsf.org
```

Options

General options

`-f` *FORMAT*, `-r` *FORMAT*, `--from=`*FORMAT*, `--read=`*FORMAT* Specify input format. *FORMAT* can be:

- `asciidoc` (AsciiDoc markup)
- `bibtex` (BibTeX bibliography)
- `biblatex` (BibLaTeX bibliography)
- `bits` (BITS XML, alias for `jats`)
- `commonmark` (CommonMark Markdown)
- `commonmark_x` (CommonMark Markdown with extensions)
- `creole` (Creole 1.0)
- `csljson` (CSL JSON bibliography)
- `csv` (CSV table)
- `tsv` (TSV table)
- `djot` (Djot markup)
- `docbook` (DocBook)
- `docx` (Word docx)
- `dokuwiki` (DokuWiki markup)
- `endnotexml` (EndNote XML bibliography)
- `epub` (EPUB)
- `fb2` (FictionBook2 e-book)
- `fodt` (flat OpenDocument text document)
- `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)
- `jira` (Jira/Confluence wiki markup)
- `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)
- `mdoc` (mdoc manual page markup)
- `muse` (Muse)
- `native` (native Haskell)
- `odt` (OpenDocument text document)
- `opml` (OPML)
- `org` (Emacs Org mode)
- `pod` (Perl's Plain Old Documentation)
- `pptx` (PowerPoint)
- `ris` (RIS bibliography)
- `rtf` (Rich Text Format)
- `rst` (reStructuredText)
- `t2t` (txt2tags)
- `textile` (Textile)
- `tikiwiki` (TikiWiki markup)
- `twiki` (TWiki markup)
- `typst` (typst)
- `vimwiki` (Vimwiki)
- `xlsx` (Excel spreadsheet)
- `xml` (XML version of native AST)
- the path of a custom Lua reader, see Custom readers and writers below

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:

- `ansi` (text with ANSI escape codes, for terminal viewing)
- `asciidoc` (modern AsciiDoc as interpreted by Asciidoctor)
- `asciidoc_legacy` (AsciiDoc as interpreted by `asciidoc-py`).
- `asciidoctor` (deprecated synonym for `asciidoc`)
- `bbcode` BBCode
- `bbcode_fluxbb` BBCode (FluxBB)
- `bbcode_phpbb` BBCode (phpBB)
- `bbcode_steam` BBCode (Steam)
- `bbcode_hubzilla` BBCode (Hubzilla)
- `bbcode_xenforo` BBCode (xenForo)
- `beamer` (LaTeX beamer slide show)
- `bibtex` (BibTeX bibliography)
- `biblatex` (BibLaTeX bibliography)

- `chunkedhtml` (zip archive of multiple linked HTML files)
- `commonmark` (CommonMark Markdown)
- `commonmark_x` (CommonMark Markdown with extensions)
- `context` (ConTeXt)
- `csljson` (CSL JSON bibliography)
- `djot` (Djot markup)
- `docbook` or `docbook4` (DocBook 4)
- `docbook5` (DocBook 5)
- `docx` (Word docx)
- `dokuwiki` (DokuWiki markup)
- `epub` or `epub3` (EPUB v3 book)
- `epub2` (EPUB v2)
- `fb2` (FictionBook2 e-book)
- `fodt` (flat OpenDocument text document)
- `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_archiving` (JATS XML, Archiving and Interchange Tag Set)
- `jats_articleauthoring` (JATS XML, Article Authoring Tag Set)
- `jats_publishing` (JATS XML, Journal Publishing Tag Set)
- `jats` (alias for `jats_archiving`)
- `jira` (Jira/Confluence 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)
- `markua` (Markua)
- `mediawiki` (MediaWiki markup)
- `ms` (roff ms)
- `muse` (Muse)
- `native` (native Haskell)
- `odt` (OpenDocument text document)
- `opml` (OPML)
- `opendocument` (OpenDocument XML)
- `org` (Emacs Org mode)

- `pdf` (PDF)
- `plain` (plain text)
- `pptx` (PowerPoint slide show)
- `rst` (reStructuredText)
- `rtf` (Rich Text Format)
- `texinfo` (GNU Texinfo)
- `textile` (Textile)
- `t2t` (txt2tags)
- `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)
- `typst` (typst)
- `vimdoc` (Vimdoc)
- `xml` (XML version of native AST)
- `xwiki` (XWiki markup)
- `zimwiki` (ZimWiki markup)
- the path of a custom Lua writer, see Custom readers and writers below

Note that `odt`, `docx`, `epub`, and `pdf` 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. If the output format is `chunkedhtml` and *FILE* has no extension, then instead of producing a `.zip` file pandoc will create a directory *FILE* and unpack the zip archive there (unless *FILE* already exists, in which case an error will be raised).

`--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 directory (by default, `$HOME/.local/share`, overridable by setting the `XDG_DATA_HOME` environment variable). If that directory does not exist and `$HOME/.pandoc` exists, it will be used (for backwards compatibility). On Windows the default user data directory is `%APPDATA%\pandoc`. You can find the default user data directory on your system by looking at the output of `pandoc --version`. Data files placed in this directory (for example, `reference.odt`, `reference.docx`, `epub.css`, `templates`) will override pandoc's normal defaults. (Note that the user data directory is not created by pandoc, so you will need to create it yourself if you want to make use of it.)

If the `PANDOC_DATA_DIRS` environment variable is set to a list of directories (separated by `:`, or `;` on Windows), these directories will be searched, in order, for data files not found in the user data directory, before pandoc falls back to its default data files. This applies to all data files that are looked up in the user data directory, including templates, filters, defaults files, metadata files, and CSL styles. Only absolute paths in `PANDOC_DATA_DIRS` are used. This makes it possible for a package manager to install, for example, Lua filters into `/some/prefix/share/pandoc/filters` and make them available by adding `/some/prefix/share/pandoc` to `PANDOC_DATA_DIRS`.

`-d` *`FILE`*, `--defaults=`*`FILE`* Specify a set of default option settings. *FILE* is a YAML or JSON file whose fields correspond to command-line option settings. All options for document conversion, including input and output files, can be set using a defaults file. The file will be searched for first in the working directory, and then in the `defaults` subdirectory of the user data directory (see `--data-dir`). If *FILE* lacks an extension, pandoc will also try adding the `.yaml` extension if *FILE* is not found. See the section Defaults files for more information on the file format. Settings from the defaults file may be overridden or extended by subsequent options on the command line.

`--completion=`*`SHELL`* Generate a shell completion script for the given shell, one of `bash`, `zsh`, or `fish`. To enable completion with pandoc, evaluate the output of this command in your shell's startup file. For example, for bash, add this to your `.bashrc`:

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

For zsh, you will need to create a directory to store zsh completions (e.g., `~/.local/share/zsh/site-functions`) or use one of the existing directories listed by `zsh --pathdirs`. Create the pandoc completion script in that directory:

```
pandoc --completion=zsh > ~/.local/share/zsh/site-functions/_pandoc
```

Then add these lines to your `.zshrc`:

```
fpath=(~/.local/share/zsh $fpath)
autoload -Uz compinit && compinit
```

`--bash-completion` *Deprecated. Use `--completion=bash` instead.*

`--sandbox[=true|false]` Run pandoc in a sandbox, limiting IO operations in readers and writers to reading the files specified on the command line. Note that this option does not limit IO operations by filters or in the production of PDF documents. But it does offer security against, for example, disclosure of files through the use of `include` directives. Anyone using pandoc on untrusted user input should use this option.

Note: some readers and writers (e.g., `docx`) need access to data files. If these are stored on the file system, then pandoc will not be able to find them when run in `--sandbox` mode and will raise an error. For these applications, we recommend using a pandoc binary compiled with the `embed_data_files` option, which causes the data files to be baked into the binary instead of being stored on the file system.

--verbose Give verbose debugging output.

--quiet Suppress warning messages.

--fail-if-warnings[=true|false] 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 for *FORMAT*, 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 `--syntax-highlighting`.

-v, --version Print version.

-h, --help Show usage message.

Reader options

--shift-heading-level-by=*NUMBER* Shift heading levels by a positive or negative integer. For example, with `--shift-heading-level-by=-1`, level 2 headings become level 1 headings, and level 3 headings become level 2 headings. Headings cannot have a level less than 1, so a heading that would be shifted below level 1 becomes a regular paragraph. Exception: with a shift of -N, a level-N heading at the beginning of the document replaces the metadata title. `--shift-heading-level-by=-1` is a good choice when converting HTML or Markdown documents that use an initial level-1 heading for the document title and level-2+ headings for sections. `--shift-heading-level-by=1` may be a good choice for converting Markdown documents that use level-1 headings for sections to HTML, since pandoc uses a level-1 heading to render the document title.

--base-header-level=*NUMBER* *Deprecated. Use `--shift-heading-level-by`=X instead, where X = NUMBER - 1.* Specify the base level for headings (defaults to 1).

--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[=true|false]` 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`.

If two or more files are processed using `--file-scope`, prefixes based on the filenames will be added to identifiers in order to disambiguate them, and internal links will be adjusted accordingly. For example, a header with identifier `foo` in `subdir/file1.txt` will have its identifier changed to `subdir__file1.txt__foo`.

`-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, Lua-filters, and citeproc processing are applied in the order specified on the command line.

`-L` *SCRIPT*, `--lua-filter=`*SCRIPT* Transform the document in a similar fashion as JSON filters (see `--filter`), but use pandoc's built-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.

See the Lua filters documentation for further details.

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

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

Filters, Lua filters, and citeproc processing are applied in the order specified on the command line.

`-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 metadata file will always be parsed as Markdown. (If the input format is Markdown or a Markdown variant, then the same variant will be used to parse the metadata file; if it is a non-Markdown format, pandoc's default Markdown extensions will be used.) This option can be used repeatedly to include multiple metadata files; values in files specified later on the command line will be preferred over those specified in earlier files. Metadata values specified inside the document, or by using `-M`, overwrite values specified with this option. The file will be searched for first in the working directory, and then in the `metadata` subdirectory of the user data directory (see `--data-dir`).

`-p`, `--preserve-tabs[=true|false]` Preserve tabs instead of converting them to spaces. (By default, pandoc converts tabs to spaces before parsing its input.) Note that this will only affect tabs in literal code spans and code blocks. Tabs in regular text are always 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) processes all the insertions and deletions. `reject` ignores them. Both `accept` and `reject` ignore comments. `all` includes all insertions, deletions, and comments, wrapped in spans with `insertion`, `deletion`, `comment-start`, and `comment-end` classes, respectively. The author and time of change are 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`*`|`*`FILE`*`.zip` Extract images and other media contained in or linked from the source document to the path *DIR*, creating it if necessary, and adjust the image references in the document so they point to the extracted files. Media are downloaded, read from the file system, or extracted from a binary container (e.g. docx), as needed. The original file paths are used if they are relative paths not containing `..`. Otherwise filenames are constructed from the SHA1 hash of the contents.

If the path given ends in `.zip`, then instead of creating a directory, pandoc will create a zip archive containing the media files.

`--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 found in this list will be followed by a nonbreaking space, and the period will not produce sentence-ending space in formats like LaTeX. The strings may not contain spaces.

`--typst-input=`*`KEY`*`[=`*`VAL`*`]` Set a parameter value that will be made available to the typst parser in `sys.inputs`, like `--input` in the `typst` CLI. Either `:` or `=` may be used to separate *KEY* from *VAL*. Values containing spaces must be quoted.

`--trace[=true|false]` Print diagnostic output tracing parser progress to stderr. This option is intended for use by developers in diagnosing performance issues.

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 the template is not found, pandoc will search for it in the `templates` subdirectory of the user data directory (see `--data-dir`). If no extension is specified and an extensionless template is not found, pandoc will look for a template with an extension corresponding to the writer, so that `--template=special` looks for `special.html` for HTML output. 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 string value *VAL* when rendering the document in standalone mode. Either `:` or `=` may be used to separate *KEY* from *VAL*. If no *VAL* is specified, the key will be given the value `true`. Structured values (lists, maps) cannot be assigned using this option, but they can be assigned in the `variables` section of a defaults file or using the `--variable-json` option. If the variable already has a *list* value, the value will be added to the list. If it already has another kind of value, it will be made into a list containing the previous and the new value. For example, `-V author=Joe -V author=Sue` makes `author` contain a list of strings: `Joe` and `Sue`.

`--variable-json=`*`KEY`*`[=`*`JSON`*`]` Set the template variable *KEY* to the value specified by a JSON string (this may be a boolean, a string, a list, or a mapping; a number will be treated as a string). For example, `--variable-json foo=false` will give `foo` the boolean false value, while `--variable-json foo='"false"'` will give it the string value `"false"`. Either `:` or `=` may be used to separate *KEY* from *VAL*. If the variable already has a value, this value will be replaced.

`-D` *`FORMAT`*, `--print-default-template=`*`FORMAT`* Print the system default template for an output *FORMAT*. (See `-t` for a list of possible *FORMAT*s.) 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.

Note that some of the default templates use partials, for example `styles.html`. To print the partials, use `--print-default-data-file`: for example, `--print-default-data-file=templates/styles.html`.

`--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`.

`--dpi=`*`NUMBER`* Specify the default dpi (dots per inch) value for conversion from pixels to inch/centimeters and vice versa. (Technically, the correct term would be ppi: pixels per inch.) The default is 96dpi. When images contain information about dpi internally, the encoded value is used instead of the default specified by this option.

`--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). 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[=true|false]`, `--table-of-contents[=true|false]` 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` and using (the default) `groff` as a `--pdf-engine`, the table of contents will appear at the end of the document. If you would prefer it to be at the beginning of the document, before the title, you can use `--pdf-engine=pdfroff`.

Because the entries in a table of contents need heading identifiers to link to, this option forces the `auto_identifiers` extension on for the input format, even when it is off by default (as it is for `typst`).

`--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).

`--lof[=true|false]`, `--list-of-figures[=true|false]` Include an automatically generated list of figures (or, in some formats, an instruction to create one) in the output document. This option has no effect unless `-s/--standalone` is used, and it only has an effect on `latex`, `context`, and `docx` output.

`--lot[=true|false]`, `--list-of-tables[=true|false]` Include an automatically generated list of tables (or, in some formats, an instruction to create one) in the output document. This option has no effect unless `-s/--standalone` is used, and it only has an effect on `latex`, `context`, and `docx` output.

`--strip-comments[=true|false]` 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.

`--syntax-highlighting=default|none|idiomatic|`*`STYLE`*`|`*`FILE`* The method to use for code syntax highlighting. Setting a specific *STYLE* causes highlighting to be performed

with the internal highlighting engine, using KDE syntax definitions and styles. The `idiomatic` method uses a format-specific highlighter if one is available, or the default style if the target format has no idiomatic highlighting method. Setting this option to `none` disables all syntax highlighting. The `default` method uses a format-specific default.

The default for HTML, EPUB, Docx, Ms, Man, and LaTeX output is to use the internal highlighter with the default style; for Typst it is to use Typst's own syntax highlighting system.

Style 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`.

`--no-highlight` *Deprecated, use `--syntax-highlighting=none` instead.*

Disables syntax highlighting for code blocks and inlines, even when a language attribute is given.

`--highlight-style=`*`STYLE|FILE`* *Deprecated, use `--syntax-highlighting=`*STYLE|FILE *instead.*

Specifies the coloring style to be used in highlighted source code.

`--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 `--syntax-highlighting`. 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. This option may be repeated to add multiple syntax definitions.

`-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`.

Note that if the output format is `odt`, this file must be in OpenDocument XML format suitable for insertion into the body of the document, and if the output is `docx`, this file must be in appropriate OpenXML format.

`-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`. Note that if the output format is `odt`, this file must be in OpenDocument XML format suitable for insertion into the body of the document, and if the output is `docx`, this file must be in appropriate OpenXML format.

`--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. This option can be used repeatedly. Search path components that come later on the command line will be searched before those that come earlier, so `--resource-path foo:bar --resource-path baz:bim` is equivalent to `--resource-path baz:bim:foo:bar`. Note that this option only has an effect when pandoc itself needs to find an image (e.g., in producing a PDF or docx, or when `--embed-resources` is used.) It will not cause image paths to be rewritten in other cases (e.g., when pandoc is generating LaTeX or HTML).

`--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://...`.

`--no-check-certificate[=true|false]` Disable the certificate verification to allow access to unsecure HTTP resources (for example when the certificate is no longer valid or self signed).

Options affecting specific writers

`--self-contained[=true|false]` *Deprecated synonym for `--embed-resources --standalone`.*

`--embed-resources[=true|false]` Produce a standalone HTML file with no external dependencies, using `data:` URIs to incorporate the contents of linked scripts, stylesheets, images, and videos. 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 `revealjs`. 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, fonts may be missing when `--math-method=mathjax` is used, and some advanced features (e.g. zoom or speaker notes) may not work in an offline "self-contained" `reveal.js` slide show.

For SVG images, `img` tags with `data:` URIs are used, unless the image has the class `inline-svg`, in which case an inline SVG element is inserted. This approach is recommended when there are many occurrences of the same SVG in a document, as `<use>` elements will be used to reduce duplication.

`--link-images[=true|false]` Include links to images instead of embedding the images in ODT. (This option currently only affects `odt` and `fodt` output.) This is worth considering for `fodt`, where images have to be base64-encoded and so take about a third more space than in the compressed `odt` package.

`--html-q-tags[=true|false]` Use `<q>` tags for quotes in HTML. (This option only has an effect if the `smart` extension is enabled for the input format used.)

`--ascii[=true|false]` 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 man and ms (which use hexadecimal escapes), and to a limited degree LaTeX (which uses standard commands for accented characters when possible).

`--reference-links[=true|false]` 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 this option only affects the `markdown`, `muse`, `html`, `epub`, `slidy`, `s5`, `slideous`, `dzslides`, and `revealjs` writers. In slide formats, specifying `--reference-location=section` will cause notes to be rendered at the bottom of a slide.

`--figure-caption-position=above|below` Specify whether figure captions go above or below figures (default is `below`). This option only affects HTML, LaTeX, Docx, ODT, and Typst output.

`--table-caption-position=above|below` Specify whether table captions go above or below tables (default is `above`). This option only affects HTML, LaTeX, Docx, ODT, and Typst output.

`--markdown-headings=setext|atx` Specify whether to use ATX-style (#-prefixed) or Setext-style (underlined) headings for level 1 and 2 headings in Markdown output. (The default is `atx`.) ATX-style headings are always used for levels 3+. This option also affects Markdown cells in `ipynb` output.

`--list-tables[=true|false]` Render tables as list tables in RST 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 `documentclass` variable 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.

In Docx output, this option adds section breaks before first-level headings if `chapter` is selected, and before first- and second-level headings if `part` is selected. Footnote numbers will restart with each section break unless the reference doc modifies this.

`-N`, `--number-sections[=true|false]` Number section headings in LaTeX, ConTeXt, HTML, Docx, ms, or EPUB output. By default, sections are not numbered. Sections with class `unnumbered` will never be numbered, even if `--number-sections` is specified.

`--number-offset=`*`NUMBER`*`[,`*`NUMBER`*`,...]` Offsets for section heading numbers. The first number is added to the section number for level-1 headings, the second for level-2 headings, and so on. So, for example, if you want the first level-1 heading in your document to be numbered "6" instead of "1", 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`. `--number-offset` only directly affects the number of the first section heading in a document; subsequent numbers increment in the normal way. Implies `--number-sections`. Currently this feature only affects HTML and Docx output.

`--listings[=true|false]` *Deprecated, use `--syntax-highlighting=idiomatic` or `--syntax-highlighting=default` instead.*

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[=true|false]` 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`, `revealjs`, `pptx`, `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. Valid values are 0-6. If a slide level of 0 is specified, slides will not be split automatically on headings, and horizontal rules must be used to indicate slide boundaries. If a slide level is not specified explicitly, the slide level will be set automatically based on the contents of the document; see Structuring the slide show.

`--section-divs[=true|false]` 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). This option only affects HTML output (and does not affect HTML slide formats).

`--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. This option only affects HTML (including HTML slide shows) and EPUB output. It should be used together with `-s/--standalone`, because the link to the stylesheet goes in the document header.

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 `--data-dir`). If it is not found there, sensible defaults will be used.

`--reference-doc=`*FILE*|*URL* Use the specified file as a style reference in producing a docx or ODT file.

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 `reference.docx` in the user data directory (see `--data-dir`). If this is not found either, sensible defaults will be used.

To produce a custom `reference.docx`, first get a copy of the default `reference.docx`: `pandoc -o custom-reference.docx --print-default-data-file reference.docx`. Then open `custom-reference.docx` in Word or LibreOffice, 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
- AbstractTitle
- Bibliography
- Heading 1
- Heading 2
- Heading 3
- Heading 4
- Heading 5
- Heading 6
- Heading 7
- Heading 8
- Heading 9
- Block Text [for block quotes]
- Footnote Block Text [for block quotes in footnotes]
- Source Code
- Footnote Text
- Definition Term
- Definition
- Caption
- Table Caption
- Image Caption
- Figure
- Captioned Figure
- TOC Heading

Character styles:

- Default Paragraph Font
- Verbatim Char
- Footnote Reference
- Hyperlink
- Section Number

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.

To produce a custom `reference.odt`, first get a copy of the default `reference.odt`: `pandoc -o custom-reference.odt --print-default-data-file reference.odt`. Then open `custom-reference.odt` in LibreOffice, modify the styles as you wish, and save the file.

`fodt` output uses the same reference documents as `odt`; the reference is still a zipped `.odt`, which is flattened along with the generated content. Note that an automatic style of the reference document is dropped, with a warning, if the generated content defines a style of the same name.

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 contain layouts with the following names (as seen within PowerPoint):

- Title Slide
- Title and Content
- Section Header
- Two Content
- Comparison
- Content with Caption
- Blank

For each name, the first layout found with that name will be used. If no layout is found with one of the names, pandoc will output a warning and use the layout with that name from the default reference doc instead. (How these layouts are used is described in PowerPoint layout choice.)

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 layouts with the names listed above).

`--split-level=`*`NUMBER`* Specify the heading level at which to split an EPUB or chunked HTML document into separate files. The default is to split into chapters at level-1 headings. In the case of EPUB, 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. For chunked HTML, this option determines how much content goes in each "chunk."

`--chunk-template=`*`PATHTEMPLATE`* Specify a template for the filenames in a `chunkedhtml` document. In the template, `%n` will be replaced by the chunk number (padded with leading 0s to 3 digits), `%s` with the section number of the chunk, `%h` with the heading text (with formatting removed), `%i` with the section identifier. For example, `section-%s-%i.html` might be resolved to `section-1.1-introduction.html`. The characters `/` and `\` are not allowed in chunk templates and will be ignored. The default is `%s-%i.html`.

`--epub-chapter-level=`*`NUMBER`* *Deprecated synonym for `--split-level`.*

`--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-title-page=true|false` Determines whether a the title page is included in the EPUB (default is `true`).

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

--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`,

`pagedjs-cli`, `prince`, `context`, `groff`, `pdfroff`, and `typst`. If the engine is not in your PATH, the full path of the engine may be specified here. If this option is not specified, pandoc uses the following defaults depending on the output format specified using `-t/--to`:

- `-t latex` or none: `pdflatex` (other options: `xelatex`, `lualatex`, `tectonic`, `latexmk`)
- `-t context`: `context`
- `-t html`: `weasyprint` (other options: `prince`, `wkhtmltopdf`, `pagedjs-cli`; see print-css.rocks for a good introduction to PDF generation from HTML/CSS)
- `-t ms`: `groff`
- `-t typst`: `typst`

This option is normally intended to be used when a PDF file is specified as `-o/--output`. However, it may still have an effect when other output formats are requested. For example, `ms` output will include `.pdfhref` macros only if a `--pdf-engine` is selected, and the macros will be differently encoded depending on whether `groff` or `pdfroff` is specified.

`--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

`-C`, `--citeproc` Process the citations in the file, replacing them with rendered citations and adding a bibliography. Citation processing will not take place unless bibliographic data is supplied, either through an external file specified using the `--bibliography` option or the `bibliography` field in metadata, or via a `references` section in metadata containing a list of citations in CSL YAML format with Markdown formatting. The style is controlled by a CSL stylesheet specified using the `--csl` option or the `csl` field in metadata. (If no stylesheet is specified, the `chicago-author-date` style will be used by default.) The citation processing transformation may be applied before or after filters or Lua filters (see `--filter`, `--lua-filter`): these transformations are applied in the order they appear on the command line. For more information, see the section on Citations.

Note: if this option is specified, the `citations` extension will be disabled automatically in the writer, to ensure that the citeproc-generated citations will be rendered instead of the format's own citation syntax.

`--bibliography=`*`FILE`* Set the `bibliography` field in the document's metadata to *FILE*, overriding any value set in the metadata. If you supply this argument multiple times, each *FILE* will be added to the bibliography. If *FILE* is a URL, it will be fetched via

HTTP. If *FILE* is not found relative to the working directory, it will be sought in the resource path (see `--resource-path`).

`--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`.) If *FILE* is a URL, it will be fetched via HTTP. If *FILE* is not found relative to the working directory, it will be sought in the resource path (see `--resource-path`) and finally in the `csl` subdirectory of the pandoc user data directory.

`--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`.) If *FILE* is a URL, it will be fetched via HTTP. If *FILE* is not found relative to the working directory, it will be sought in the resource path (see `--resource-path`) and finally in the `csl` subdirectory of the pandoc user data directory.

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

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

Math rendering

`--math-method=`*`METHOD`*`[:`*`URL`*`]` Specify the method used to display TeX math. The following methods are possible (some take an optional URL, separated from the method name by a colon). The default value is `mathml`.

`plain` Render math using Unicode characters, to the extent possible. When this is not possible, fall back to plain TeX.

`mathjax[:`*`URL`*`]` Render math using MathJax. 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. This method works in HTML and HTML-derived formats.

`mathml` Convert TeX math to MathML. MathML is supported natively by the main web browsers and most e-book readers. This option works in HTML and XML formats.

`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 `--math-method=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: this method works not just in HTML but in Markdown, 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. This method works in HTML and HTML-derived formats.

`gladtex` Enclose TeX math in `<eq>` tags in HTML output. The resulting HTML can then be processed by GladTeX to produce SVG images of the typeset formulas and an HTML file with these images embedded.

```
pandoc -s --math-method=gladtex input.md -o myfile.htex
gladtex -d image_dir myfile.htex
# produces myfile.html and images in image_dir
```

`--mathjax[=`*`URL`*`]` *Deprecated. Use `--math-method=mathjax[:`*URL*`]` instead.*

`--mathml` *Deprecated. Use `--math-method=mathml` instead.*

`--webtex[=`*`URL`*`]` *Deprecated. Use `--math-method=webtex[:`*URL*`]` instead.*

`--katex[=`*`URL`*`]` *Deprecated. Use `--math-method=katex[:`*URL*`]` instead.*

`--gladtex` *Deprecated. Use `--math-method=gladtex` instead.*

Options for wrapper scripts

`--dump-args[=true|false]` 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[=true|false]` 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
```

Exit codes

If pandoc completes successfully, it will return exit code 0. Nonzero exit codes have the following meanings:

Code	Error
1	PandocIOError
3	PandocFailOnWarningError
4	PandocAppError
5	PandocTemplateError
6	PandocOptionError
21	PandocUnknownReaderError
22	PandocUnknownWriterError
23	PandocUnsupportedExtensionError
24	PandocCiteprocError
25	PandocBibliographyError
31	PandocEpubSubdirectoryError
43	PandocPDFError
44	PandocXMLError
47	PandocPDFProgramNotFoundError
61	PandocHttpError
62	PandocShouldNeverHappenError
63	PandocSomeError
64	PandocParseError
66	PandocMakePDFError
67	PandocSyntaxMapError
83	PandocFilterError
84	PandocLuaError
89	PandocNoScriptingEngine
91	PandocMacroLoop
92	PandocUTF8DecodingError
93	PandocIpynbDecodingError
94	PandocUnsupportedCharsetError
95	PandocInputNotTextError
97	PandocCouldNotFindDataFileError
98	PandocCouldNotFindMetadataFileError
99	PandocResourceNotFound

Defaults files

The `--defaults` option may be used to specify a package of options, in the form of a YAML or JSON file. Examples in this section will be given in YAML, but the equivalent forms in JSON will also work.

Fields that are omitted will just have their regular default values. So a defaults file can be as simple as one line:

```yaml
verbosity: INFO
```

or in JSON:

```json
{ "verbosity": "INFO" }
```

In fields that expect a file path (or list of file paths), the following syntax may be used to interpolate environment variables:

```yaml
csl:  ${HOME}/mycsldir/special.csl
```

`${USERDATA}` may also be used; this will always resolve to the user data directory that is current when the defaults file is parsed, regardless of the setting of the environment variable `USERDATA`.

`${.}` will resolve to the directory containing the defaults file itself. This allows you to refer to resources contained in that directory:

```yaml
epub-cover-image: ${.}/cover.jpg
epub-metadata: ${.}/meta.xml
resource-path:
- .             # the working directory from which pandoc is run
- ${.}/images   # the images subdirectory of the directory
                # containing this defaults file
```

This environment variable interpolation syntax *only* works in fields that expect file paths.

Defaults files can be placed in the `defaults` subdirectory of the user data directory and used from any directory. For example, one could create a file specifying defaults for writing letters, save it as `letter.yaml` in the `defaults` subdirectory of the user data directory, and then invoke these defaults from any directory using `pandoc --defaults letter` or `pandoc -dletter`.

When multiple defaults are used, their contents will be combined.

Note that, where command-line arguments may be repeated (`--metadata-file`, `--css`, `--include-in-header`, `--include-before-body`, `--include-after-body`, `--variable`, `--metadata`, `--syntax-definition`), the values specified on the command line will combine with values specified in the defaults file, rather than replacing them.

The following tables show the mapping between the command line and defaults file entries.

command line	defaults file
`foo.md`	`input-file: foo.md`

`foo.md bar.md`	`input-files:`
	`  - foo.md`
	`  - bar.md`

The value of `input-files` may be left empty to indicate input from stdin, and it can be an empty sequence `[]` for no input.

General options

command line	defaults file
`--from markdown+emoji`	`from: markdown+emoji`

`--to markdown+hard_line_breaks`	`reader: markdown+emoji`
	`to: markdown+hard_line_breaks`
	`writer: markdown+hard_line_breaks`

`--output foo.pdf`	`output-file: foo.pdf`
`--output -`	`output-file:`
`--data-dir dir`	`data-dir: dir`
`--defaults file`	`defaults:` `- file`
`--verbose`	`verbosity: INFO`
`--quiet`	`verbosity: ERROR`
`--fail-if-warnings`	`fail-if-warnings: true`
`--sandbox`	`sandbox: true`
`--log=FILE`	`log-file: FILE`

Options specified in a defaults file itself always have priority over those in another file included with a `defaults:` entry.

`verbosity` can have the values `ERROR`, `WARNING`, or `INFO`.

Reader options

command line	defaults file
`--shift-heading-level-by -1`	`shift-heading-level-by: -1`
`--indented-code-classes python`	`indented-code-classes:` `  - python`
`--default-image-extension ".jpg"`	`default-image-extension: '.jpg'`
`--file-scope`	`file-scope: true`

`--citeproc \`
` --lua-filter count-words.lua \`
` --filter special.lua`
	`filters:`
	`  - citeproc`
	`  - count-words.lua`
	`  - type: json`
	`  path: special.lua`

`--metadata key=value \`
` --metadata key2`
	`metadata:`
	`  key: value`
	`  key2: true`

`--metadata-file meta.yaml`	`metadata-files:`
	`  - meta.yaml`
	
	`metadata-file: meta.yaml`

`--preserve-tabs`	`preserve-tabs: true`
`--tab-stop 8`	`tab-stop: 8`
`--track-changes accept`	`track-changes: accept`
`--extract-media dir`	`extract-media: dir`
`--abbreviations abbrevs.txt`	`abbreviations: abbrevs.txt`
`--typst-input foo=bar`	`typst-inputs:` `  foo: bar`
`--trace`	`trace: true`

Metadata values specified in a defaults file are parsed as literal string text, not Markdown.

Filters will be assumed to be Lua filters if they have the `.lua` extension, and JSON filters otherwise. But the filter type can also be specified explicitly, as shown. Filters are run in the order specified. To include the built-in citeproc filter, use either `citeproc` or `{type: citeproc}`.

General writer options

command line	defaults file
`--standalone`	`standalone: true`
`--template letter`	`template: letter`

command line	defaults file

`--variable key=val \`
`  --variable key2`
	`variables:`
	`  key: val`
	`  key2: true`

`--eol nl`	`eol: nl`
`--dpi 300`	`dpi: 300`
`--wrap preserve`	`wrap: "preserve"`
`--columns 72`	`columns: 72`
`--table-of-contents`	`table-of-contents: true`
`--toc`	`toc: true`
`--toc-depth 3`	`toc-depth: 3`
`--strip-comments`	`strip-comments: true`
`--no-highlight`	`syntax-highlighting: 'none'`
`--syntax-highlighting kate`	`syntax-highlighting: kate`

`--syntax-definition mylang.xml`	`syntax-definitions:`
	`  - mylang.xml`
	
	`syntax-definition: mylang.xml`

`--include-in-header inc.tex`	`include-in-header:` `  - inc.tex`
`--include-before-body inc.tex`	`include-before-body:` `  - inc.tex`
`--include-after-body inc.tex`	`include-after-body:` `  - inc.tex`
`--resource-path .:foo`	`resource-path: ['.','foo']`
`--request-header foo:bar`	`request-headers:` `  - ["User-Agent", "Mozilla/5.0"]`
`--no-check-certificate`	`no-check-certificate: true`

Options affecting specific writers

command line	defaults file
`--self-contained`	`self-contained: true`
`--link-images`	`link-images: true`
`--html-q-tags`	`html-q-tags: true`
`--ascii`	`ascii: true`
`--reference-links`	`reference-links: true`
`--reference-location block`	`reference-location: block`
`--figure-caption-position=above`	`figure-caption-position: above`
`--table-caption-position=below`	`table-caption-position: below`
`--markdown-headings atx`	`markdown-headings: atx`
`--list-tables`	`list-tables: true`

command line	defaults file
`--top-level-division chapter`	`top-level-division: chapter`
`--number-sections`	`number-sections: true`
`--number-offset=1,4`	`number-offset: \[1,4\]`
`--listings`	`listings: true`
`--list-of-figures`	`list-of-figures: true`
`--lof`	`lof: true`
`--list-of-tables`	`list-of-tables: true`
`--lot`	`lot: true`
`--incremental`	`incremental: true`
`--slide-level 2`	`slide-level: 2`
`--section-divs`	`section-divs: true`
`--email-obfuscation references`	`email-obfuscation: references`
`--id-prefix ch1`	`identifier-prefix: ch1`
`--title-prefix MySite`	`title-prefix: MySite`

`--css styles/screen.css  \`
`  --css styles/special.css`
	`css:`
	`  - styles/screen.css`
	`  - styles/special.css`

`--reference-doc my.docx`	`reference-doc: my.docx`
`--epub-cover-image cover.jpg`	`epub-cover-image: cover.jpg`
`--epub-title-page=false`	`epub-title-page: false`
`--epub-metadata meta.xml`	`epub-metadata: meta.xml`

`--epub-embed-font special.otf \`
`  --epub-embed-font headline.otf`
	`epub-fonts:`
	`  - special.otf`
	`  - headline.otf`

`--split-level 2`	`split-level: 2`
`--chunk-template="%i.html"`	`chunk-template: "%i.html"`
`--epub-subdirectory=""`	`epub-subdirectory: ''`
`--ipynb-output best`	`ipynb-output: best`
`--pdf-engine xelatex`	`pdf-engine: xelatex`

`--pdf-engine-opt=--shell-escape`	`pdf-engine-opts:`
	`  - '-shell-escape'`
	
	`pdf-engine-opt: '-shell-escape'`

Citation rendering

command line	defaults file
`--citeproc`	`citeproc: true`
`--bibliography logic.bib`	`bibliography: logic.bib`
`--csl ieee.csl`	`csl: ieee.csl`

command line	defaults file
`--citation-abbreviations ab.json`	`citation-abbreviations: ab.json`
`--natbib`	`cite-method: natbib`
`--biblatex`	`cite-method: biblatex`

`cite-method` can be `citeproc`, `natbib`, or `biblatex`. This only affects LaTeX output. If you want to use citeproc to format citations, you should also set 'citeproc: true'.

If you need control over when the citeproc processing is done relative to other filters, you should instead use `citeproc` in the list of `filters` (see Reader options).

Math rendering in HTML

command line	defaults file
`--math-method=mathjax`	`math-method: method: mathjax`
`--math-method=webtex:URL`	`math-method: method: webtex url: mathjax`
`--math-method=mathml`	`math-method: method: mathml`

If the command line option accepts a URL argument, a `url:` field can be added to `math-method:`.

Options for wrapper scripts

command line	defaults file
`--dump-args`	`dump-args: true`
`--ignore-args`	`ignore-args: true`

Templates

When the `-s/--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 `docx` output, customize the `default.openxml` 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`).
- `pptx` has no template.

Note that `docx`, `odt`, and `pptx` output can also be customized using `--reference-doc`. Use a reference doc to adjust the styles in your document; use a template to handle variable interpolation and customize the presentation of metadata, the position of the table of contents, boilerplate text, etc.

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 `-V/--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 `-M/--metadata` option. In addition, some variables are given default values by pandoc. See Variables below for a list of variables used in pandoc's default templates.

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.

Template syntax

Comments

Anything between the sequence `$--` and the end of the line will be treated as a comment and omitted from the output.

Delimiters

To mark variables and control structures in the template, either `$…$` or `${…}` may be used as delimiters. The styles may also be mixed in the same template, but the opening and closing delimiter must match in each case. The opening delimiter may be followed by one or more spaces or tabs, which will be ignored. The closing delimiter may be preceded by one or more spaces or tabs, which will be ignored.

To include a literal `$` in the document, use `$$`.

Interpolated variables

A slot for an interpolated variable is a variable name surrounded by matched delimiters. Variable names must begin with a letter and can contain letters, numbers, `_`, `-`, and `.`. The keywords `it`, `if`, `else`, `endif`, `for`, `sep`, and `endfor` may not be used as variable names. Examples:

```
$foo$
$foo.bar.baz$
$foo_bar.baz-bim$
$ foo $
${foo}
${foo.bar.baz}
${foo_bar.baz-bim}
${ foo }
```

Variable names with periods are used to get at structured variable values. So, for example, `employee.salary` will return the value of the `salary` field of the object that is the value of the `employee` field.

- If the value of the variable is a simple value, it will be rendered verbatim. (Note that no escaping is done; the assumption is that the calling program will escape the strings appropriately for the output format.)
- If the value is a list, the values will be concatenated.
- If the value is a map, the string `true` will be rendered.
- Every other value will be rendered as the empty string.

Conditionals

A conditional begins with `if(variable)` (enclosed in matched delimiters) and ends with `endif` (enclosed in matched delimiters). It may optionally contain an `else` (enclosed in matched delimiters). The `if` section is used if `variable` has a true value, otherwise the `else` section is used (if present). The following values count as true:

- any map
- any array containing at least one true value
- any nonempty string
- boolean True

Note that in YAML metadata (and metadata specified on the command line using `-M/--metadata`), unquoted `true` and `false` will be interpreted as Boolean values. But a variable specified on the command line using `-V/--variable` will always be given a string value. Hence a conditional `if(foo)` will be triggered if you use `-V foo=false`, but not if you use `-M foo=false`.

Examples:

```
$if(foo)$bar$endif$

$if(foo)$
  $foo$
$endif$

$if(foo)$
part one
$else$
part two
$endif$

${if(foo)}bar${endif}

${if(foo)}
  ${foo}
${endif}

${if(foo)}
${ foo.bar }
${else}
no foo!
${endif}
```

The keyword `elseif` may be used to simplify complex nested conditionals:

```
$if(foo)$
XXX
$elseif(bar)$
YYY
$else$
ZZZ
$endif$
```

For loops

A for loop begins with `for(variable)` (enclosed in matched delimiters) and ends with `endfor` (enclosed in matched delimiters).

- If `variable` is an array, the material inside the loop will be evaluated repeatedly, with `variable` being set to each value of the array in turn, and concatenated.
- If `variable` is a map, the material inside will be set to the map.
- If the value of the associated variable is not an array or a map, a single iteration will be performed on its value.

Examples:

```
$for(foo)$$foo$$sep$, $endfor$

$for(foo)$
  - $foo.last$, $foo.first$
$endfor$

${ for(foo.bar) }
  - ${ foo.bar.last }, ${ foo.bar.first }
${ endfor }

$for(mymap)$
$it.name$: $it.office$
$endfor$
```

You may optionally specify a separator between consecutive values using `sep` (enclosed in matched delimiters). The material between `sep` and the `endfor` is the separator.

```
${ for(foo) }${ foo }${ sep }, ${ endfor }
```

Instead of using `variable` inside the loop, the special anaphoric keyword `it` may be used.

```
${ for(foo.bar) }
  - ${ it.last }, ${ it.first }
${ endfor }
```

Partials

Partials (subtemplates stored in different files) may be included by using the name of the partial, followed by `()`, for example:

```
${ styles() }
```

Partials will be sought in the directory containing the main template. The file name will be assumed to have the same extension as the main template if it lacks an extension. When calling the partial, the full name including file extension can also be used:

```
${ styles.html() }
```

(If a partial is not found in the directory of the template and the template path is given as a relative path, it will also be sought in the `templates` subdirectory of the user data directory.)

Partials may optionally be applied to variables using a colon:

```
${ date:fancy() }
```

```
${ articles:bibentry() }
```

If `articles` is an array, this will iterate over its values, applying the partial `bibentry()` to each one. So the second example above is equivalent to

```
${ for(articles) }
${ it:bibentry() }
${ endfor }
```

Note that the anaphoric keyword `it` must be used when iterating over partials. In the above examples, the `bibentry` partial should contain `it.title` (and so on) instead of `articles.title`.

Final newlines are omitted from included partials.

Partials may include other partials.

A separator between values of an array may be specified in square brackets, immediately after the variable name or partial:

```
${months[, ]}

${articles:bibentry()[; ]}
```

The separator in this case is literal and (unlike with `sep` in an explicit `for` loop) cannot contain interpolated variables or other template directives.

Nesting

To ensure that content is "nested," that is, subsequent lines indented, use the `^` directive:

```
$item.number$  $^$$item.description$ ($item.price$)
```

In this example, if `item.description` has multiple lines, they will all be indented to line up with the first line:

```
00123  A fine bottle of 18-year old
       Oban whiskey. ($148)
```

To nest multiple lines to the same level, align them with the `^` directive in the template. For example:

```
$item.number$  $^$$item.description$ ($item.price$)
               (Available til $item.sellby$.)
```

will produce

```
00123  A fine bottle of 18-year old
       Oban whiskey. ($148)
       (Available til March 30, 2020.)
```

If a variable occurs by itself on a line, preceded by whitespace and not followed by further text or directives on the same line, and the variable's value contains multiple lines, it will be nested automatically.

Breakable spaces

Normally, spaces in the template itself (as opposed to values of the interpolated variables) are not breakable, but they can be made breakable in part of the template by using the `~` keyword (ended with another `~`).

```
$~$This long line may break if the document is rendered
with a short line length.$~$
```

Pipes

A pipe transforms the value of a variable or partial. Pipes are specified using a slash (`/`) between the variable name (or partial) and the pipe name. Example:

```
$for(name)$
$name/uppercase$
$endfor$

$for(metadata/pairs)$
- $it.key$: $it.value$
$endfor$

$employee:name()/uppercase$
```

Pipes may be chained:

```
$for(employees/pairs)$
$it.key/alpha/uppercase$. $it.name$
$endfor$
```

Some pipes take parameters:

```
|----------------------|------------|
$for(employee)$
$it.name.first/uppercase/left 20 "| "$$it.name.salary/right 10 " | " |"$
$endfor$
|----------------------|------------|
```

Currently the following pipes are predefined:

- `pairs`: Converts a map or array to an array of maps, each with `key` and `value` fields. If the original value was an array, the `key` will be the array index, starting with 1.
- `uppercase`: Converts text to uppercase.
- `lowercase`: Converts text to lowercase.
- `length`: Returns the length of the value: number of characters for a textual value, number of elements for a map or array.
- `reverse`: Reverses a textual value or array, and has no effect on other values.
- `first`: Returns the first value of an array, if applied to a non-empty array; otherwise returns the original value.

- `last`: Returns the last value of an array, if applied to a non-empty array; otherwise returns the original value.
- `rest`: Returns all but the first value of an array, if applied to a non-empty array; otherwise returns the original value.
- `allbutlast`: Returns all but the last value of an array, if applied to a non-empty array; otherwise returns the original value.
- `chomp`: Removes trailing newlines (and breakable space).
- `nowrap`: Disables line wrapping on breakable spaces.
- `alpha`: Converts textual values that can be read as an integer into lowercase alphabetic characters `a..z` (mod 26). This can be used to get lettered enumeration from array indices. To get uppercase letters, chain with `uppercase`.
- `roman`: Converts textual values that can be read as an integer into lowercase roman numerals. This can be used to get roman-numeral enumeration from array indices. To get uppercase roman, chain with `uppercase`.
- `left n "leftborder" "rightborder"`: Renders a textual value in a block of width `n`, aligned to the left, with an optional left and right border. Has no effect on other values. This can be used to align material in tables. Widths are positive integers indicating the number of characters. Borders are strings inside double quotes; literal `"` and `\` characters must be backslash-escaped.
- `right n "leftborder" "rightborder"`: Renders a textual value in a block of width `n`, aligned to the right, and has no effect on other values.
- `center n "leftborder" "rightborder"`: Renders a textual value in a block of width `n`, aligned to the center, and has no effect on other values.

Variables

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`.) The page title in HTML is set by `pagetitle`, which is equal to `title` by default.

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

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

`abstract-title` title of abstract, currently used only in HTML, EPUB, docx, and Typst. This will be set automatically to a localized value, depending on `lang`, but can be manually overridden.

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

`subject` document subject, included in ODT, PDF, docx, EPUB, and pptx metadata

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

`category` document category, included in docx and pptx metadata

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

```
---
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 pptx.

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 `span`s and `div`s 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

`document-css` Enables inclusion of most of the CSS in the `styles.html` partial (have a look with `pandoc --print-default-data-file=templates/styles.html`). Unless you use `--css`, this variable is set to `true` by default. You can disable it with e.g. `pandoc -M document-css=false`.

`mainfont` sets the CSS `font-family` property on the `html` element.

`fontsize` sets the base CSS `font-size`, which you'd usually set to e.g. `20px`, but it also accepts `pt` (12pt = 16px in most browsers).

`fontcolor`, **`fontcolordark`** sets the CSS `color` property on the `html` element.

`linkcolor`, **`linkcolordark`** sets the CSS `color` property on all links.

`quotebordercolor`, **`quotebordercolordark`** sets the CSS `border-color` property for the left-border of `blockquote` elements.

`quotecolor`, **`quotecolordark`** sets the CSS `color` property for `blockquote` elements.

`monofont` sets the CSS `font-family` property on `code` and `pre` elements.

`monobackgroundcolor`, **`monobackgroundcolordark`** sets the CSS `background-color` property on `code` and `pre` elements and adds extra padding.

`linestretch` sets the CSS `line-height` property on the `html` element, which is preferred to be unitless.

`maxwidth` sets the CSS `max-width` property (default is 36em).

`backgroundcolor`, **`backgroundcolordark`** sets the CSS `background-color` property on the `html` element.

`margin-left`, **`margin-right`**, **`margin-top`**, **`margin-bottom`** sets the corresponding CSS `padding` properties on the `body` element.

To override or extend some CSS for just one document, include for example:

```
---
header-includes: |
  <style>
  blockquote {
    font-style: italic;
  }
  tr.even {
    background-color: #f0f0f0;
  }
  td, th {
    padding: 0.5em 2em 0.5em 0.5em;
  }
  tbody {
    border-bottom: none;
  }
  </style>
---
```

Note on `color-scheme`

By default, the browser's chosen `color-scheme` (either `light` or `dark`) will affect the color of various unstyled elements like input fields, which are generally only applicable if your document is specifically targeting HTML.

By default, Pandoc emits `color-scheme: light dark` to allow the system to choose between a default light-mode or dark-mode scheme, but it alters this if variables specifically override colors, under the assumption that the user wishes for that scheme to always be used.

Currently, this is only per a basic heuristic that checks whether the following variables are provided:

- `fontcolor`
- `fontcolordark`
- `backgroundcolor`
- `backgroundcolordark`

And if only one "mode" is provided (light or dark), `color-scheme` is set so that this chosen mode is used always, regardless of the system's preference for light or dark mode.

Note that there are a few caveats to this, namely that:

- If the browser does not support `light-dark` (older than ~2024), then it will always choose light mode, regardless of this override.
- When printing, light mode is always chosen, and color variables are ignored.
- Syntax highlighting is currently unaffected by color scheme.

In general, if you wish to override the theming for a single scheme, you should choose to override either the normal `*color` variables (light mode) or the `*colordark` variables (dark mode) to ensure that the styles do not conflict with other styles added by new versions of Pandoc or the browser itself.

Variables for HTML math

`classoption` when using `--math-method=katex`, you can render display math equations flush left using YAML metadata or with `-M classoption=fleqn`.

Variables for HTML slides

These affect HTML output when producing slide shows with pandoc.

`institute` author affiliations: can be a list when there are multiple authors
`revealjs-url` base URL for reveal.js documents (defaults to `https://unpkg.com/reveal.js@^5`)
`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`)
`title-slide-attributes` additional attributes for the title slide of reveal.js slide shows. See background in reveal.js, beamer, and pptx for an example.
`highlightjs-theme` highlight.js theme for code highlighting when using `--syntax-highlighting=idiomatic` with reveal.js (defaults to `monokai`). See the highlight.js demo page for available themes.

All reveal.js configuration options are available as variables. To turn off boolean flags that default to true in reveal.js, use `0`.

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)
`beameroption` add extra beamer option with `\setbeameroption{}`
`institute` author affiliations: can be a list when there are multiple authors
`logo` logo image for slides
`logooptions` options for logo image (e.g., `width`, `height`)

`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`, `colorthemeoptions`, `fontthemeoptions`, `innerthemeoptions`, `outerthemeoptions` options for LaTeX beamer themes (lists)
`titlegraphic` image for title slide: can be a list
`titlegraphicoptions` options for title slide image (e.g., `width`, `height`)
`shorttitle`, `shortsubtitle`, `shortauthor`, `shortinstitute`, `shortdate` some beamer themes use short versions of the title, subtitle, author, institute, date

Variables for PowerPoint

These variables control the visual aspects of a slide show that are not easily controlled via templates.

`monofont` font to use for code.

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 `\subsubsection` (third- or fourth-level headings). Instead of using this option, KOMA-Script can adjust headings more extensively:

```
---
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:

```
---
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:

```
---
geometry:
- top=30mm
- left=20mm
- heightrounded
...
```

shorthands Enable language-specific shorthands when loading `babel`. (By default, pandoc includes `shorthands=off` when loading `babel`, disabling language-specific shorthands.)

hyperrefoptions option for `hyperref` package, e.g. `linktoc=all`; repeat for multiple options:

```
---
hyperrefoptions:
- linktoc=all
- pdfwindowui
- pdfpagemode=FullScreen
...
```

indent if true, pandoc will use 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)

pagestyle control `\pagestyle{}`: 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 `numbersections` variable)

beamerarticle produce an article from Beamer slides. Note: if you set this variable, you must specify the beamer writer but use the default *LaTeX* template: for example, `pandoc -Vbeamerarticle -t beamer --template default.latex`.

handout produce a handout version of Beamer slides (with overlays condensed into single slides)

csquotes load `csquotes` package and use `\enquote` or `\enquote*` for quoted text.

csquotesoptions options to use for `csquotes` package (repeat for multiple options).

babeloptions options to pass to the babel package (may be repeated for multiple options). This defaults to `provide=*` if the main language isn't a European language written with Latin or Cyrillic script or Vietnamese. Most users will not need to adjust the default setting.

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:

```
---
fontfamily: libertinus
fontfamilyoptions:
- osf
- p
...
```

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**, **CJKsansfont**, **CJKmonofont** font families for use with `xelatex` or `lualatex`: take the name of any system font, using the `fontspec` package. `CJKmainfont` uses the `xecjk` package if `xelatex` is used, or the `luatexja` package if `lualatex` is used.

mainfontoptions, **sansfontoptions**, **monofontoptions**, **mathfontoptions**, **CJKoptions**, **luatexjapresetoptions** 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
...
```

mainfontfallback, **sansfontfallback**, **monofontfallback** fonts to try if a glyph isn't found in `mainfont`, `sansfont`, or `monofont` respectively. These are lists. The font name must be followed by a colon and optionally a set of options, for example:

```
---
mainfontfallback:
```

```
  - "FreeSans:"
  - "NotoColorEmoji:mode=harf"
...
```

Font fallbacks currently only work with `lualatex`.

`babelfonts` a map of Babel language names (e.g. `chinese`) to the font to be used with the language:

```
---
babelfonts:
  chinese-hant: "Noto Serif CJK TC"
  russian: "Noto Serif"
...
```

`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

`boxlinks` add visible box around links (has no effect if `colorlinks` is 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

`urlstyle` style for URLs (e.g., `tt`, `rm`, `sf`, and, the default, `same`)

Front matter

`lof`, **`lot`** include list of figures, list of tables (can also be set using `--lof/--list-of-figures`, `--lot/--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

`biblio-style` bibliography style, when used with `--natbib` and `--biblatex`

`biblio-title` bibliography title, when used with `--natbib` and `--biblatex`

`bibliography` bibliography to use for resolving references

`natbiboptions` list of options for natbib

Other

`pdf-trailer-id` the PDF trailer ID; must be two PDF byte strings if set, conventionally with 16 bytes each. E.g., `<00112233445566778899aabbccddeeff> <00112233445566778899aabbccddeeff>`.
See the section on reproducible builds.

`pdfstandard` PDF standard(s) for the document, e.g. `ua-2`, `a-4f`. Supports PDF/A, PDF/X, and PDF/UA variants. Requires LuaLaTeX and LaTeX 2023+. Repeat for multiple standards:

```
---
pdfstandard:
- ua-2
- a-4f
...
```

Variables for ConTeXt

Pandoc uses these variables when creating a PDF with ConTeXt.

`fontsize` font size for body text (e.g. `10pt`, `12pt`)

`headertext`, **`footertext`** text to be placed in running header or footer (see ConTeXt Headers and Footers); repeat up to four times for different placement

`indenting` controls indentation of paragraphs, e.g. `yes,small,next` (see ConTeXt Indentation); repeat for multiple options

`interlinespace` adjusts line spacing, e.g. `4ex` (using `setupinterlinespace`); repeat for multiple options

`layout` options for page margins and text arrangement (see ConTeXt Layout); repeat for multiple options

`linkcolor`, **`contrastcolor`** color for links outside and inside a page, e.g. `red`, `blue` (see ConTeXt Color)

`linkstyle` typeface style for links, e.g. `normal`, `bold`, `slanted`, `boldslanted`, `type`, `cap`, `small`

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

`mainfont`, **`sansfont`**, **`monofont`**, **`mathfont`** font families: take the name of any system font (see ConTeXt Font Switching)

`mainfontfallback`, **`sansfontfallback`**, **`monofontfallback`** list of fonts to try, in order, if a glyph is not found in the main font. Use `\definefallbackfamily`-compatible font name syntax. Emoji fonts are unsupported.

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

`pagenumbering` page number style and location (using `setuppagenumbering`); repeat for multiple options

papersize paper size, e.g. `letter`, `A4`, `landscape` (see ConTeXt Paper Setup); repeat for multiple options

pdfa adds to the preamble the setup necessary to generate PDF/A of the type specified, e.g. `1a:2005`, `2a`. If no type is specified (i.e. the value is set to True, by e.g. `--metadata=pdfa` or `pdfa: true` in a YAML metadata block), `1b:2005` will be used as default, for reasons of backwards compatibility. Using `--variable=pdfa` without specified value is not supported. 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 and output intent may be specified using the variables `pdfaiccprofile` and `pdfaintent`. See also ConTeXt PDFA for more details.

pdfaiccprofile when used in conjunction with `pdfa`, specifies the ICC profile to use in the PDF, e.g. `default.cmyk`. If left unspecified, `sRGB.icc` is used as default. May be repeated to include multiple profiles. Note that the profiles have to be available on the system. They can be obtained from ConTeXt ICC Profiles.

pdfaintent when used in conjunction with `pdfa`, specifies the output intent for the colors, e.g. `ISO coated v2 300\letterpercent\space (ECI)`. If left unspecified, `sRGB IEC61966-2.1` is used as default.

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

urlstyle typeface style for links without link text, e.g. `normal`, `bold`, `slanted`, `boldslanted`, `type`, `cap`, `small`

whitespace spacing between paragraphs, e.g. `none`, `small` (using `setupwhitespace`)

includesource include all source documents as file attachments in the PDF file

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

section section number in man pages

Variables for Texinfo

`version` version of software (used in title and title page)
`filename` name of info file to be generated (defaults to a name based on the texi filename)

Variables for Typst

`template` Typst template to use (relative path only).
`margin` A dictionary with the fields defined in the Typst documentation: `x`, `y`, `top`, `bottom`, `left`, `right`.
`papersize` Paper size: `a4`, `us-letter`, etc.
`mainfont` Name of system font to use for the main font.
`fontsize` Font size (e.g., `12pt`).
`section-numbering` Schema to use for numbering sections, e.g. `1.A.1`.
`page-numbering` Schema to use for numbering pages, e.g. `1` or `i`, or an empty string to omit page numbering.
`columns` Number of columns for body text.
`thanks` contents of acknowledgments footnote after document title. (Note: as of typst 0.15, this does not work properly, due to an upstream issue; see jgm/pandoc#11807.)
`mathfont`, **`codefont`** Name of system font to use for math and code, respectively.
`linestretch` adjusts line spacing, e.g. `1.25`, `1.5`
`linkcolor`, **`filecolor`**, **`citecolor`** color for external links, internal links, and citation links, respectively: expects a hexadecimal color code
`csl-entry-spacing` entry spacing between citeproc-generated bibliography entries (with a unit, e.g. `1em`). This will be set automatically based on the CSL style but can be overridden.
`csl-hanging-indent` amount of hanging indent in citeproc-generated bibliography entries (with a unit, e.g. `1.5em`). This will be set automatically based on the CSL style but can be overridden.

Variables for ms

`fontfamily` `A` (Avant Garde), `B` (Bookman), `H` (Helvetica), `HN` (Helvetica Narrow), `P` (Palatino), or `T` (Times New Roman). This setting does not affect source code, which is always displayed using monospace Courier. These built-in fonts are limited in their coverage of characters. Additional fonts may be installed using the script `install-font.sh` provided by Peter Schaffter and documented in detail on his web site.
`indent` paragraph indent (e.g. `2m`)
`lineheight` line height (e.g. `12p`)
`pointsize` point size (e.g. `10p`)

Variables for RTF

`fontsize` font size for body text, e.g. `12pt` or `12.5pt`.

Variables set automatically

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, html, 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)

`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. `$curdir$/$sourcefile$`.

`pdf-engine` name of PDF engine if provided using `--pdf-engine`, or the default engine for the format if PDF output is requested.

`curdir` working directory from which pandoc is run.

`pandoc-version` pandoc version.

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

`toc-title` title of table of contents (works only with EPUB, HTML, revealjs, opendocument, odt, docx, pptx, beamer, LaTeX). Note that in docx and pptx a custom `toc-title` will be picked up from metadata, but cannot be set as a variable.

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 `--markdown-headings`).

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`, `html`
output formats `markdown`, `latex`, `context`, `org`, `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`, `man`, `typst`
output formats `markdown`, `muse`
enabled by default in `markdown`, `muse`, `man`

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,

Heading	Identifier
`Heading identifiers in HTML`	`heading-identifiers-in-html`
`Maître d'hôtel`	`maître-dhôtel`
`*Dogs*?--in *my* house?`	`dogs--in-my-house`
`[HTML], [S5], or [RTF]?`	`html-s5-or-rtf`
`3. Applications`	`applications`
`33`	`section`

These rules should, in most cases, allow one to determine the identifier from the 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. Emojis are replaced by their names.

Math Input

The extensions `tex_math_dollars`, `tex_math_gfm`, `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 are described in more detail in their respective sections of Pandoc's Markdown:

- `raw_html` allows HTML elements which are not representable in pandoc's AST to be parsed as raw HTML. By default, this is disabled for HTML input.
- `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`, `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`.

- `native_divs` causes HTML `div` elements to be parsed as native pandoc Div blocks. If you want them to be parsed as raw HTML, use `-f html-native_divs+raw_html`.
- `native_spans` causes HTML `span` elements to be parsed as native pandoc Span inlines. If you want them to be parsed as raw HTML, use `-f html-native_spans+raw_html`. If you want to drop all `div`s and `span`s when converting HTML to Markdown, you can use `pandoc -f html-native_divs-native_spans -t markdown`.

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

- 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`, `latex`

Extension: `native_numbering`

Enables numbering of figures and tables via counter fields specific to the target format. This extension can be enabled/disabled for the following formats:

output formats `odt`, `opendocument`, `docx`

If this extension is not enabled, tables and figures in these formats will not be numbered.

Extension: `xrefs_name`

Links to headings, figures and tables inside the document are substituted with cross-references that will use the name or caption of the referenced item. The original link text is replaced once the generated document is refreshed. This extension can be combined with `xrefs_number` in which case numbers will appear before the name.

Text in cross-references is only made consistent with the referenced item once the document has been refreshed.

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

output formats `odt`, `opendocument`

Extension: `xrefs_number`

Links to headings, figures and tables inside the document are substituted with cross-references that will use the number of the referenced item. The original link text is discarded. This extension can be combined with `xrefs_name` in which case the name or caption will appear after the number.

For the `xrefs_number` extension to be useful, heading numbers must be enabled in the generated document, also table and figure captions must be enabled using for example the `native_numbering` extension.

Numbers in cross-references are only visible in the final document once it has been refreshed.

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

output formats `odt`, `opendocument`

Extension: `styles`

When converting from docx, add `custom-styles` attributes for all docx styles, regardless of whether pandoc understands the meanings of these styles. Because attributes cannot be added directly to paragraphs or text in the pandoc AST, paragraph styles will cause Divs to be created and character styles will cause Spans to be created to hold the attributes. (Table styles will be added to the Table elements directly.) This extension can be used with docx custom styles.

When converting from odt or fodt, add `custom-styles` attributes for paragraph and character styles, using the style names shown in the word processor. Automatic styles, which only record direct formatting, are not recorded; instead, the named style they are based on is used. A few styles are not recorded: the default paragraph style (`Default Paragraph Style`, or `Standard`), the default styles of table cells, and styles that pandoc interprets itself

(such as `Emphasis` and `Strong Emphasis`). The paragraphs of list items are not wrapped in Divs. A heading gets a `custom-style` attribute if its style is not the standard one for its level.

input formats `docx`, `odt`, `fodt`

Extension: `amuse`

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

Extension: `raw_markdown`

In the `ipynb` input format, this causes Markdown cells to be included as raw Markdown blocks (allowing lossless round-tripping) rather than being parsed. Use this only when you are targeting `ipynb` or a Markdown-based output format.

Extension: `citations` (typst)

When the `citations` extension is enabled in `typst` (as it is by default), `typst` citations will be parsed as native pandoc citations, and native pandoc citations will be rendered as `typst` citations.

Extension: `citations` (org)

When the `citations` extension is enabled in `org`, org-cite and org-ref style citations will be parsed as native pandoc citations, and org-cite citations will be used to render native pandoc citations.

Extension: `citations` (docx)

When `citations` is enabled in `docx`, citations inserted by Zotero or Mendeley or EndNote plugins will be parsed as native pandoc citations. (Otherwise, the formatted citations generated by the bibliographic software will be parsed as regular text.)

Extension: `fancy_lists` (org)

Some aspects of Pandoc's Markdown fancy lists are also accepted in `org` input, mimicking the option `org-list-allow-alphabetical` in Emacs. As in Org Mode, enabling this extension allows lowercase and uppercase alphabetical markers for ordered lists to be parsed in addition to arabic ones. Note that for Org, this does not include roman numerals or the `#` placeholder that are enabled by the extension in Pandoc's Markdown.

Extension: `element_citations`

In the `jats` output formats, this causes reference items to be replaced with `<element-citation>` elements. These elements are not influenced by CSL styles, but all information on the item is included in tags.

Extension: `ntb`

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.

Extension: `smart_quotes` (org)

Interpret straight quotes as curly quotes during parsing. When *writing* Org, then the `smart_quotes` extension has the reverse effect: what would have been curly quotes comes out straight.

This extension is implied if `smart` is enabled.

Extension: `special_strings` (org)

Interpret `---` as em-dashes, `--` as en-dashes, `\-` as shy hyphen, and `...` as ellipses.

This extension is implied if `smart` is enabled.

Extension: `tagging`

Enabling this extension with `context` output will produce markup suitable for the production of tagged PDFs. This includes additional markers for paragraphs and alternative markup for emphasized text. The `emphasis-command` template variable is set if the extension is enabled.

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 original 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`

Original 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 for labels and link anchors in the LaTeX, ConTeXt, Textile, Jira markup, and AsciiDoc writers.

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

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

is just the same as

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

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

Extension: `implicit_header_references`

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

```
# Heading identifiers in HTML
```

you can simply write

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

or

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

or

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

instead of giving the identifier explicitly:

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

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

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

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

```
# Foo

[foo]: bar

See [foo]
```

Block quotations

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

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

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

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

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

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

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

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

```
>     code
```

Extension: `blank_before_blockquote`

Original 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.
>> Not nested, since `blank_before_blockquote` is enabled by default
```

Verbatim (code) blocks

Indented code blocks

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

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

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

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

Fenced code blocks

Extension: `fenced_code_blocks`

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

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

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

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

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

Extension: `backtick_code_blocks`

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

Extension: `fenced_code_attributes`

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

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

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

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

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

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

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

This is equivalent to:

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

This shortcut form may be combined with attributes:

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

Which is equivalent to:

```
``` {.haskell .numberLines}
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 `--syntax-highlighting=none` option. To set the highlighting style or method, use `--syntax-highlighting`. 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
```

Inline formatting (such as emphasis) is allowed in the content (though it can't cross line boundaries). Block-level formatting (such as block quotes or lists) is not recognized.

This syntax is borrowed from reStructuredText.

Lists

Bullet lists

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

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

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

```
* one

* two

* three
```

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

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

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

But Markdown also allows a "lazy" format:

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

Block content in list items

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

```
  * First paragraph.

    Continued.

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

        { code }
```

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

```
*     code

  continuation paragraph
```

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

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

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

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

+ Another one; this looks
bad but is legal.

    Second paragraph of second
list item.
```

Ordered lists

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

In original 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 original 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-parenthesis 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]

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

```
B. Russell won a Nobel Prize (but not for "On Denoting").
```

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
```

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

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

Note: the '`#`' ordered list marker doesn't work with `commonmark`.

As with `commonmark`, ordered sublists must start with `1` or the equivalent (e.g., `i` or `a`). This helps avoid unintentional lists caused by things like parenthesized dates.

Extension: `startnum`

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

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

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

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

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

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

Extension: `task_lists`

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

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

Definition lists

Extension: `definition_lists`

Pandoc supports definition lists, using the syntax of PHP Markdown Extra with some extensions.[2]

```
Term 1

:   Definition 1

Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

Each term must fit on one line, which may optionally be followed by a blank line, and must be followed by one or more definitions. A definition begins with a colon or tilde, which may be indented one or two spaces.

A term may have multiple definitions, and each definition may consist of one or more indented block elements (paragraph, code block, list, etc.). The blocks in the definition should be indented to the column of the first non-space content after the `:` or `~` marker, or (if the `four_space_rule` extension is enabled) four spaces or one tab stop. As with other Markdown lists, you can "lazily" omit indentation in paragraph continuation lines:

```
Term 1

: Definition
with lazy continuation.

  Second paragraph of the definition.
```

[2] I have been influenced by the suggestions of David Wheeler.

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.

Numbered example lists

Extension: `example_lists`

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

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

Explanation of examples.

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

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

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

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

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

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.

You can repeat an earlier numbered example by re-using its label:

```
(@foo) Sample sentence.

Intervening text...

This theory can explain the case we saw earlier (repeated):

(@foo) Sample sentence.
```

This only works reliably, though, if the repeated item is in a list by itself, because each numbered example list will be numbered continuously from its starting number.

In a longer (book-length) work, one might need to reset the example list counter at the beginning of each chapter. One can do that using the following syntax:

```
(1@foo) This item will be numbered 1, and subsequent example
    list items will be numbered 2, 3, etc.
```

Any number may be used instead of `1`. Note that if there are several example list items in a row, the number only has an effect if placed on the first of them.

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:

```
* * * *

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

We strongly recommend that horizontal rules be separated from surrounding text by blank lines. If a horizontal rule is not followed by a blank line, pandoc may try to interpret the lines that follow as a YAML metadata block or a table.

Tables

Four kinds of tables may be used. The first three kinds (simple, multiline, and grid) require lining up columns and thus presuppose the use of a fixed-width font, such as Courier. The fourth kind (pipe tables) 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 `table:` or just `:`), which will be stripped off. It may appear either before or after the table.

Extension: `simple_tables`

Simple tables look like this:

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

Table:  Demonstration of simple table syntax.
```

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

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

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

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

Extension: `multiline_tables`

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

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

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

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

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

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

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

The header may be omitted in multiline tables as well as simple tables:

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

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

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

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

Extension: `grid_tables`

Grid tables look like this:

```
: Sample grid table.

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

The row of =s separates the header from the table body, and can be omitted for a headerless table. The cells of grid tables may contain arbitrary block elements (multiple paragraphs, code blocks, lists, etc.).

Cells can span multiple columns or rows:

```
+---------------------+----------+
| Property            | Earth    |
+=============+=======+==========+
|             | min   | -89.2 °C |
| Temperature +-------+----------+
| 1961-1990   | mean  | 14 °C    |
|             +-------+----------+
|             | max   | 56.7 °C  |
+-------------+-------+----------+
```

A table header may contain more than one row:

```
+--------------------+-----------------------+
| Location           | Temperature 1961-1990 |
|                    | in degree Celsius     |
|                    +-------+-------+-------+
```

```
|                     | min   | mean  | max   |
+=====================+=======+=======+=======+
| Antarctica          | -89.2 | N/A   | 19.8  |
+---------------------+-------+-------+-------+
| Earth               | -89.2 | 14    | 56.7  |
+---------------------+-------+-------+-------+
```

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

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

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

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

A table foot can be defined by enclosing it with separator lines that use = instead of -:

```
 +---------------+---------------+
 | Fruit         | Price         |
 +===============+===============+
 | Bananas       | $1.34         |
 +---------------+---------------+
 | Oranges       | $2.10         |
 +===============+===============+
 | Sum           | $3.44         |
 +===============+===============+
```

The foot must always be placed at the very bottom of the table.

Grid tables can be created easily using Emacs' table-mode (`M-x table-insert`).

Extension: `pipe_tables`

Pipe tables look like this:

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

  : Demonstration of pipe table syntax.
```

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

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

```
fruit| price
-----|-----:
apple|2.05
pear|1.37
orange|3.09
```

The cells of pipe tables cannot contain block elements like paragraphs and lists, and cannot span multiple lines. If any line of the Markdown source is longer 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 the 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.

Extension: `table_attributes`

Attributes may be attached to tables by including them at the end of the caption. (For the syntax, see `header_attributes`.)

```
  : Here's the caption. {#ident .class key="value"}
```

Table placement in LaTeX

By default, pandoc will render tables in LaTeX using the `longtable` environment. The tables will not "float" and they can span page boundaries. If you prefer standard "floating" tables (which are limited to what can fit on one page), add the `float` class to the table's attributes (which can be specified after the caption). Optionally, the `latex-placement` attribute may be used to pass parameters affecting the float placement:

```
: The caption of the table. {#ident .float latex-placement="ht"}
```

To repeat the caption of a `longtable` on each page, add the `repeat-caption` class to the table's attributes. Repeated captions are not added to the list of tables. This class has no effect on tables with the `float` class, since those tables cannot span pages.

```
: The caption of the table. {.repeat-caption}
```

Note that `longtable` does not work with two-column layouts, so if you are producing a two-column document using LaTeX, you will need to specify the `float` class on every table, or use this small Lua filter to add it automatically:

```lua
function Table(el)
  el.classes:insert("float")
  return el
end
```

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 date 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. The initial line `---` must not be followed by a blank line. 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 JSON may be used as well, because JSON is a subset of YAML.)

Note that, because of the way pandoc concatenates input files when several are provided, you may also keep the metadata in a separate YAML file and pass it to pandoc as an argument, along with your Markdown files:

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

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

Metadata will be taken from the fields of the YAML object and added to any existing document metadata. Metadata can contain lists and objects (nested arbitrarily), but all string scalars will be interpreted as Markdown. Fields with names ending in an underscore will be ignored by pandoc. (They may be given a role by external processors.) Field names must

not be interpretable as YAML numbers or boolean values (so, for example, `yes`, `True`, and `15` cannot be used as field names).

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

Each metadata block is handled internally as an independent YAML document. This means, for example, that any YAML anchors defined in a block cannot be referenced in another block.

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, and if it contains a backslash escape, then it must be ensured that it is not treated as a YAML escape sequence. The pipe character (`|`) can be used to begin an indented block that will be interpreted literally, without need for escaping. This form is necessary when the field contains blank lines or block-level formatting:

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

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

The literal block after the `|` must be indented relative to the line containing the `|`. If it is not, the YAML will be invalid and pandoc will not interpret it as metadata. For an overview of the complex rules governing YAML, see the Wikipedia entry on YAML syntax.

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

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

Variables can contain arbitrary YAML structures, but the template must match this structure. The `author` variable in the default templates expects a simple list or string, but can be

changed to support more complicated structures. The following combination, for example, would add an affiliation to the author if one is given:

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

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

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

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

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

Note: the `yaml_metadata_block` extension works not just with `markdown` but with `commonmark` (and it is enabled by default in `gfm` and `commonmark_x`). However, in `commonmark`, `gfm`, and `commonmark_x`, the following restrictions apply:

- The YAML metadata block must occur at the beginning of the document (and there can be only one). If multiple files are given as arguments to pandoc, only the first can be a YAML metadata block.
- The leaf nodes of the YAML structure are parsed in isolation from each other and from the rest of the document. So, for example, you can't use a reference link in these contexts if the link definition is somewhere else in the document.

Backslash escapes

Extension: `all_symbols_escapable`

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

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

one will get

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

instead of

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

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

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

(However, if the `markdown_strict` format is used, the original Markdown rule will be used.)

A backslash-escaped space is parsed as a nonbreaking space. In TeX output, it will appear as `~`. In HTML and XML output, it will appear as a literal unicode nonbreaking space character (note that it will thus actually look "invisible" in the generated HTML source; you can still use the `--ascii` command-line option to make it appear as an explicit entity).

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

Backslash escapes do not work in verbatim contexts.

Inline formatting

Emphasis

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

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

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

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

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

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

Extension: `intraword_underscores`

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

```
feas∗ible∗, not feas∗able∗.
```

Strikeout

Extension: `strikeout`

To strike out a section of text with a horizontal line, begin and end it with ∼∼. Thus, for example,

```
This ∼∼is deleted text.∼∼
```

Superscripts and subscripts

Extension: `superscript`, `subscript`

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

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

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

Verbatim

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

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

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

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

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

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

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

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

Extension: `inline_code_attributes`

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

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

Underline

To underline text, use the `underline` class:

```
[Underline]{.underline}
```

Or, without the `bracketed_spans` extension (but with `native_spans`):

```
<span class="underline">Underline</span>
```

This will work in all output formats that support underline.

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.

Highlighting

To highlight text, use the `mark` class:

```
[Mark]{.mark}
```

Or, without the `bracketed_spans` extension (but with `native_spans`):

```
<span class="mark">Mark</span>
```

This will work in all output formats that support highlighting.

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.

For display math, use `$$` delimiters. (In this case, the delimiters may be separated from the formula by whitespace. However, there can be no blank lines between the opening and closing `$$` 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, Txt2Tags, 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 math will appear verbatim surrounded by `latexmath:[...]`. For `asciidoc_legacy` the bracketed material will also include inline or display math delimiters.

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 `--math-method=mathml` is used, it will be rendered using MathML in an `inlineequation` or `informalequation` tag. Otherwise it will be rendered, if possible, using Unicode characters.

Docx and PowerPoint It will be rendered using OMML math markup.

FictionBook2 If `--math-method=webtex` is used, formulas are rendered as images using CodeCogs or another 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, Textile, and Txt2Tags 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 commonmark output, if `raw_html` is enabled, superscripts, subscripts, strikeouts and small capitals will be represented as HTML. Otherwise, plain-text fallbacks will be used.

Disabling the `raw_html` extension has no effect on `commonmark` (or `gfm` or `commonmark_x`) *input*.

Extension: `markdown_in_html_blocks`

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

Pandoc behaves this way when the `markdown_strict` format is used; but by default, pandoc interprets material between HTML block tags as Markdown. Thus, for example, pandoc will turn

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

into

```
<table>
<tr>
<td><em>one</em></td>
<td><a href="https://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>`, `<style>`, `<pre>`, and `<textarea>` tags is not interpreted as Markdown.

This departure from original 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` elements for content inside `<span>` tags. For the most part this should give the same output as `raw_html`, but it makes it easier to write pandoc filters to manipulate groups of inlines.

Extension: `raw_tex`

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

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

Note that in LaTeX environments, like

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

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

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

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

Generic raw attribute

Extension: `raw_attribute`

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

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

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

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

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

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

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

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

The raw attribute cannot be combined with regular attributes.

LaTeX macros

Extension: `latex_macros`

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

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

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

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

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

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

Links

Markdown allows links to be specified in several ways.

Automatic links

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

```
<https://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](https://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.) If the label is empty (`[]`), then it will implicitly be taken to be the same as the link text; thus `[foo][]` is equivalent to `[foo][foo]`. (If the `shortcut_reference_links` extension is enabled, the empty `[]` may be omitted.)

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 of reference links and link definitions:

```
See [the website *I* built][my website].

See [my website][] and [the bar page][1] and
the [home page of the FSF][fsf].

[my website]: http://foo.bar.baz
[1]: /foo/bar.html  "My title, optional"
[fsf]: https://fsf.org (The Free Software Foundation)
[special page]: /bar#special  'A title in single quotes'
```

The URL in a link definition 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]: https://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
```

The link definition may come either before or after a reference link that uses the label.

Note: In some 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 all 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 (the part in square brackets) 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 description will be used as the caption.

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

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

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

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

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

To specify an alt text for the image that is different from the caption, you can use an explicit attribute (assuming the `link_attributes` extension is set):

```
![The caption.](image.png){alt="description of image"}
```

For LaTeX output, you can specify a figure's positioning by adding the `latex-placement` attribute.

```
![The caption.](image.png){latex-placement="ht"}
```

Extension: `link_attributes`

Attributes can be set on links and images:

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

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

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

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

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

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

- Dimensions may be converted to a form that is compatible with the output format (for example, dimensions given in pixels will be converted to inches when converting HTML to LaTeX). Conversion between pixels and physical measurements is affected by the `--dpi` option (by default, 96 dpi is assumed, unless the image itself contains dpi information).

- The `%` unit is generally relative to some available space. For example the above example will render to the following.
  - HTML: `<img src="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.

Note: the `commonmark` parser doesn't permit colons after the attributes.

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

Example:

```
::::: {#special .sidebar}
Here is a paragraph.

And another.
:::::
```

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

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

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

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

Extension: `bracketed_spans`

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

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

Footnotes

Extension: `footnotes`

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

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

[^1]: Here is the footnote.

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

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

        { some.code }

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

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

The identifiers in footnote references may not contain spaces, tabs, newlines, or the characters `^`, `[`, or `]`. 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.^[Inline 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.

Citation syntax

Extension: `citations`

To cite a bibliographic item with an identifier foo, use the syntax `@foo`. Normal citations should be included in square brackets, with semicolons separating distinct items:

```
Blah blah [@doe99; @smith2000; @smith2004].
```

How this is rendered depends on the citation style. In an author-date style, it might render as

```
Blah blah (Doe 1999, Smith 2000, 2004).
```

In a footnote style, it might render as

```
Blah blah.[^1]

[^1]:  John Doe, "Frogs," *Journal of Amphibians* 44 (1999);
Susan Smith, "Flies," *Journal of Insects* (2000);
Susan Smith, "Bees," *Journal of Insects* (2004).
```

See the CSL user documentation for more information about CSL styles and how they affect rendering.

Unless a citation key starts with a letter, digit, or `_`, and contains only alphanumerics and single internal punctuation characters (`:.#$%&-+?<>~/`), it must be surrounded by curly braces, which are not considered part of the key. In `@Foo_bar.baz.`, the key is `Foo_bar.baz` because the final period is not *internal* punctuation, so it is not included in the key. In `@{Foo_bar.baz.}`, the key is `Foo_bar.baz.`, including the final period. In `@Foo_bar--baz`, the key is `Foo_bar` because the repeated internal punctuation characters terminate the key. The curly braces are recommended if you use URLs as keys: `[@{https://example.com/bib?name=foobar&date=2000}, p. 33]`.

Citation items may optionally include a prefix, a locator, and a suffix. In

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

the first item (`doe99`) has prefix `see`, locator `pp. 33-35`, and suffix `and *passim*`. The second item (`smith04`) has locator `chap. 1` and no prefix or suffix.

Pandoc uses some heuristics to separate the locator from the rest of the subject. It is sensitive to the locator terms defined 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.

In complex cases, you can force something to be treated as a locator by enclosing it in curly braces or prevent parsing the suffix as a locator by prepending curly braces:

```
[@smith{ii, A, D-Z}, with a suffix]
[@smith, {pp. iv, vi-xi, (xv)-(xvii)} with suffix here]
[@smith{}, 99 years later]
```

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 author-in-text citation, by omitting the square brackets:

```
@smith04 says blah.

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

This will cause the author's name to be rendered, followed by the bibliographical details. Use this form when you want to make the citation the subject of a sentence.

When you are using a note style, it is usually better to let citeproc create the footnotes from citations rather than writing an explicit note. If you do write an explicit note that contains a citation, note that normal citations will be put in parentheses, while author-in-text citations will not. For this reason, it is sometimes preferable to use the author-in-text style inside notes when using a note style.

Many CSL styles will format citations differently when the same source has been cited earlier. In documents with chapters, it is usually desirable to reset this position information at the beginning of every chapter. To do this, add the class `reset-citation-positions` to the heading for each chapter:

```
# The Beginning {.reset-citation-positions}
```

Note that this class only has an effect when placed on top-level headings; it is ignored in nested blocks.

Non-default 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: `rebase_relative_paths`

Rewrite relative paths for Markdown links and images, depending on the path of the file containing the link or image link. For each link or image, pandoc will compute the directory of the containing file, relative to the working directory, and prepend the resulting path to the link or image path.

The use of this extension is best understood by example. Suppose you have a subdirectory for each chapter of a book, `chap1`, `chap2`, `chap3`. Each contains a file `text.md` and a number of images used in the chapter. You would like to have `![image](spider.jpg)` in

`chap1/text.md` refer to `chap1/spider.jpg` and `![image](spider.jpg)` in `chap2/text.md` refer to `chap2/spider.jpg`. To do this, use

```
pandoc chap*/*.md -f markdown+rebase_relative_paths
```

Without this extension, you would have to use `![image](chap1/spider.jpg)` in `chap1/text.md` and `![image](chap2/spider.jpg)` in `chap2/text.md`. Links with relative paths will be rewritten in the same way as images.

Absolute paths and URLs are not changed. Neither are empty paths or paths consisting entirely of a fragment, e.g., `#foo`.

Note that relative paths in reference links and images will be rewritten relative to the file containing the link reference definition, not the file containing the reference link or image itself, if these differ.

Extension: `mark`

To highlight a section of text, begin and end it with ==. Thus, for example,

```
This ==is deleted text.==
```

Extension: `attributes`

Allows attributes to be attached to any inline or block-level element when parsing `commonmark`. The syntax for the attributes is the same as that used in `header_attributes`.

- Attributes that occur immediately after an inline element affect that element. If they follow a space, then they belong to the space. (Hence, this option subsumes `inline_code_attributes` and `link_attributes`.)
- Attributes that occur immediately before a block element, on a line by themselves, affect that element.
- Consecutive attribute specifiers may be used, either for blocks or for inlines. Their attributes will be combined.
- Attributes that occur at the end of the text of a Setext or ATX heading (separated by whitespace from the text) affect the heading element. (Hence, this option subsumes `header_attributes`.)
- Attributes that occur after the opening fence in a fenced code block affect the code block element. (Hence, this option subsumes `fenced_code_attributes`.)
- Attributes that occur at the end of a reference link definition affect links that refer to that definition.

Note that pandoc's AST does not currently allow attributes to be attached to arbitrary elements. Hence a Span or Div container will be added if needed.

Extension: `old_dashes`

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

Extension: `angle_brackets_escapable`

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

Extension: `lists_without_preceding_blankline`

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

Extension: `four_space_rule`

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

Extension: `spaced_reference_links`

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

```
[foo] [bar].
```

Extension: `hard_line_breaks`

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

Extension: `ignore_line_breaks`

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

Extension: `east_asian_line_breaks`

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

Extension: `emoji`

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

Extension: `tex_math_gfm`

Supports two GitHub-specific formats for math. Inline math: `` $`e=mc^2`$ ``.

Display math:

```
``` math
e=mc^2
```
```

Extension: `tex_math_single_backslash`

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

Extension: `tex_math_double_backslash`

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

Extension: `markdown_attribute`

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

Extension: `mmd_title_block`

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

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

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

Extension: `abbreviations`

Parses PHP Markdown Extra abbreviation keys, like

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

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

Extension: `alerts`

Supports GitHub-style Markdown alerts, like

```
> [!TIP]
> Helpful advice for doing things better or more easily.
```

Extension: `autolink_bare_uris`

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

Extension: `mmd_link_attributes`

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

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

[ref]: https://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: `gutenberg`

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

Extension: `sourcepos`

Include source position attributes when parsing `commonmark` or `djot`. For elements that accept attributes, a `data-pos` attribute is added; other elements are placed in a surrounding Div or Span element with a `data-pos` attribute.

Extension: `short_subsuperscripts`

Parse MultiMarkdown-style subscripts and superscripts, which start with a '~' or '^' character, respectively, and include the alphanumeric sequence that follows. For example:

```
x^2 = 4
```

or

```
Oxygen is O~2.
```

Extension: `wikilinks_title_after_pipe`

Pandoc supports multiple Markdown wikilink syntaxes, regardless of whether the title is before or after the pipe.

Using `--from=markdown+wikilinks_title_after_pipe` results in

```
[[URL|title]]
```

while using `--from=markdown+wikilinks_title_before_pipe` results in

```
[[title|URL]]
```

Markdown variants

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

- `markdown_phpextra` (PHP Markdown Extra)
- `markdown_github` (deprecated GitHub-Flavored Markdown)
- `markdown_mmd` (MultiMarkdown)
- `markdown_strict` (Markdown.pl)
- `commonmark` (CommonMark)
- `gfm` (Github-Flavored Markdown)
- `commonmark_x` (CommonMark with many pandoc extensions)

To see which extensions are supported for a given format, and which are enabled by default, you can use the command

```
pandoc --list-extensions=FORMAT
```

where `FORMAT` is replaced with the name of the format.

Note that the list of extensions for `commonmark`, `gfm`, and `commonmark_x` is defined relative to default commonmark. So, for example, `backtick_code_blocks` does not appear as an extension, since it is enabled by default and cannot be disabled.

Citations

When the `--citeproc` option is used, pandoc can automatically generate citations and a bibliography in a number of styles. Basic usage is

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

To use this feature, you will need to have

- a document containing citations (see Citation syntax);
- a source of bibliographic data: either an external bibliography file or a list of `references` in the document's YAML metadata;
- optionally, a CSL citation style.

Specifying bibliographic data

You can specify an external bibliography using the `bibliography` metadata field in a YAML metadata section or the `--bibliography` command line argument. If you want to use multiple bibliography files, you can supply multiple `--bibliography` arguments or set the `bibliography` metadata field to a YAML array. A bibliography may have any of these formats:

Format	File extension
BibLaTeX	.bib
BibTeX	.bibtex
CSL JSON	.json
CSL YAML	.yaml
RIS	.ris

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

In BibTeX and BibLaTeX databases, pandoc parses LaTeX markup inside fields such as `title`; in CSL YAML databases, pandoc Markdown; and in CSL JSON databases, an HTML-like markup:

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

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

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

If both an external bibliography and inline (YAML metadata) references are provided, both will be used. In case of conflicting `id`s, the inline references will take precedence.

Note that pandoc can be used to produce such a YAML metadata section from a BibTeX, BibLaTeX, or CSL JSON bibliography:

```
pandoc chem.bib -s -f biblatex -t markdown
pandoc chem.json -s -f csljson -t markdown
```

Indeed, pandoc can convert between any of these citation formats:

```
pandoc chem.bib -s -f biblatex -t csljson
pandoc chem.yaml -s -f markdown -t biblatex
```

Running pandoc on a bibliography file with the `--citeproc` option will create a formatted bibliography in the format of your choice:

```
pandoc chem.bib -s --citeproc -o chem.html
pandoc chem.bib -s --citeproc -o chem.pdf
```

Capitalization in titles

If you are using a bibtex or biblatex bibliography, then observe the following rules:

- English titles should be in title case. Non-English titles should be in sentence case, and the `langid` field in biblatex should be set to the relevant language. (The following values are treated as English: `american`, `british`, `canadian`, `english`, `australian`, `newzealand`, `USenglish`, or `UKenglish`.)
- As is standard with bibtex/biblatex, proper names should be protected with curly braces so that they won't be lowercased in styles that call for sentence case. For example:

  ```
  title = {My Dinner with {Andre}}
  ```

- In addition, words that should remain lowercase (or camelCase) should be protected:

  ```
  title = {Spin Wave Dispersion on the {nm} Scale}
  ```

  Though this is not necessary in bibtex/biblatex, it is necessary with citeproc, which stores titles internally in sentence case, and converts to title case in styles that require it. Here we protect "nm" so that it doesn't get converted to "Nm" at this stage.

If you are using a CSL bibliography (either JSON or YAML), then observe the following rules:

- All titles should be in sentence case.
- Use the `language` field for non-English titles to prevent their conversion to title case in styles that call for this. (Conversion happens only if `language` begins with `en` or is left empty.)

- Protect words that should not be converted to title case using this syntax:

```
Spin wave dispersion on the <span class="nocase">nm</span> scale
```

Conference papers, published vs. unpublished

For a formally published conference paper, use the biblatex entry type `inproceedings` (which will be mapped to CSL `paper-conference`).

For an unpublished manuscript, use the biblatex entry type `unpublished` without an `eventtitle` field (this entry type will be mapped to CSL `manuscript`).

For a talk, an unpublished conference paper, or a poster presentation, use the biblatex entry type `unpublished` with an `eventtitle` field (this entry type will be mapped to CSL `speech`). Use the biblatex `type` field to indicate the type, e.g. "Paper", or "Poster". `venue` and `eventdate` may be useful too, though `eventdate` will not be rendered by most CSL styles. Note that `venue` is for the event's venue, unlike `location` which describes the publisher's location; do not use the latter for an unpublished conference paper.

Specifying a citation style

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` (or `citation-style`) metadata field. By default, pandoc will use the Chicago Manual of Style author-date format. (You can override this default by copying a CSL style of your choice to `default.csl` in your user data directory.) The CSL project provides further information on finding and editing styles.

The `--citation-abbreviations` option (or the `citation-abbreviations` metadata field) may be used to specify a JSON file containing abbreviations of journals that should be used in formatted bibliographies when `form="short"` is specified. The format of the file can be illustrated with an example:

```
{ "default": {
    "container-title": {
            "Lloyd's Law Reports": "Lloyd's Rep",
            "Estates Gazette": "EG",
            "Scots Law Times": "SLT"
    }
  }
}
```

Citations in note styles

Pandoc's citation processing is designed to allow you to move between author-date, numerical, and note styles without modifying the Markdown source. When you're using a note style, avoid inserting footnotes manually. Instead, insert citations just as you would in an author-date style—for example,

```
Blah blah [@foo, p. 33].
```

The footnote will be created automatically. Pandoc will take care of removing the space and moving the note before or after the period, depending on the setting of `notes-after-punctuation`, as described below in Other relevant metadata fields.

In some cases you may need to put a citation inside a regular footnote. Normal citations in footnotes (such as `[@foo, p. 33]`) will be rendered in parentheses. In-text citations (such as `@foo [p. 33]`) will be rendered without parentheses. (A comma will be added if appropriate.) Thus:

```
[^1]:  Some studies [@foo; @bar, p. 33] show that
frubulicious zoosnaps are quantical.  For a survey
of the literature, see @baz [chap. 1].
```

Placement of the bibliography

If the style calls for a list of works cited, it will be placed in a div with id `refs`, if one exists:[1]

```
::: {#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
```

[1]Note that if `--file-scope` is used, a div written this way will be given an identifier of the form `FILE__refs`, to avoid duplicate identifiers (see `--file-scope`). In view of this possibility, pandoc will place the bibliography in any div whose identifier is `refs` *or* ends with `__refs`.

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 put the bibliography into a variable in your template, one way to do that is to put the div with id `refs` into a metadata field, e.g.

```
---
refs: |
   ::: {#refs}
   :::
...
```

You can then put the variable `$refs$` into your template where you want the bibliography to be placed.

Including uncited items in the bibliography

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

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

@item3
```

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

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

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

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

Other relevant metadata fields

A few other metadata fields affect bibliography formatting:

`link-citations` If true, citations will be hyperlinked to the corresponding bibliography entries (for author-date and numerical styles only). Defaults to false.

`link-bibliography` If true, DOIs, PMCIDs, PMIDs, and URLs in bibliographies will be rendered as hyperlinks. (If an entry contains a DOI, PMCID, PMID, or URL, but none of these fields are rendered by the style, then the title, or in the absence of a title the whole entry, will be hyperlinked.) Defaults to true.

`lang` The `lang` field will affect how the style is localized, for example in the translation of labels, the use of quotation marks, and the way items are sorted. (For backwards compatibility, `locale` may be used instead of `lang`, but this use is deprecated.)
A BCP 47 language tag is expected: for example, `en`, `de`, `en-US`, `fr-CA`, `ug-Cyrl`. The unicode extension syntax (after `-u-`) may be used to specify options for collation (sorting) more precisely. Here are some examples:

- `zh-u-co-pinyin`: Chinese with the Pinyin collation.
- `es-u-co-trad`: Spanish with the traditional collation (with `Ch` sorting after `C`).
- `fr-u-kb`: French with "backwards" accent sorting (with `coté` sorting after `côte`).
- `en-US-u-kf-upper`: English with uppercase letters sorting before lower (default is lower before upper).

`notes-after-punctuation` If true (the default for note styles), pandoc will put footnote references or superscripted numerical citations after following punctuation. For example, if the source contains `blah blah [@jones99].`, the result will look like `blah blah.[^1]`, with the note moved after the period and the space collapsed. If false, the space will still be collapsed, but the footnote will not be moved after the punctuation. The option may also be used in numerical styles that use superscripts for citation numbers (but for these styles the default is not to move the citation).

Slide shows

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 slide shows in Microsoft PowerPoint format.

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

```
% 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
```

```
- 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. (In beamer, a "block" will be created. If the heading has the class `example`, an `exampleblock` environment will be used; if it has the class `alert`, an `alertblock` will be used; otherwise a regular `block` will be used.)
- 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 either just use level-1 headings for all slides (in that case, level 1 will be the slide level) or you can set `--slide-level=0`.

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 unless you set `--slide-level=0` (which lets reveal.js produce a one-dimensional layout and only interprets horizontal rules as slide boundaries).

PowerPoint layout choice

When creating slides, the pptx writer chooses from a number of pre-defined layouts, based on the content of the slide:

Title Slide This layout is used for the initial slide, which is generated and filled from the metadata fields `date`, `author`, and `title`, if they are present.

Section Header This layout is used for what pandoc calls "title slides", i.e. slides which start with a header which is above the slide level in the hierarchy.

Two Content This layout is used for two-column slides, i.e. slides containing a div with class `columns` which contains at least two divs with class `column`.

Comparison This layout is used instead of "Two Content" for any two-column slides in which at least one column contains text followed by non-text (e.g. an image or a table).

Content with Caption This layout is used for any non-two-column slides which contain text followed by non-text (e.g. an image or a table).

Blank This layout is used for any slides which only contain blank content, e.g. a slide containing only speaker notes, or a slide containing only a non-breaking space.

Title and Content This layout is used for all slides which do not match the criteria for another layout.

These layouts are chosen from the default pptx reference doc included with pandoc, unless an alternative reference doc is specified using `--reference-doc`.

When reading pptx, each slide becomes a level-2 heading with the identifier `slide-N`, followed by the slide's content. Slides hidden from the slide show are read too, with the class `hidden` on their heading.

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:

```
::: incremental

- Eat spaghetti
- Drink wine

:::
```

or

```
::: nonincremental

- Eat spaghetti
- Drink wine

:::
```

While using `incremental` and `nonincremental` divs is 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.

If you want to include a block-quoted list, you can work around this behavior by putting the list inside a fenced div, so that it is not the direct child of the block quote:

```
> ::: wrapper
> - a
> - list in a quote
> :::
```

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
```

Note: this feature is not yet implemented for PowerPoint output.

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, a number of heading classes and attributes are recognized as frame options and will be passed through as options to the frame: see Frame attributes in beamer, below.

Speaker notes

Speaker notes are supported in reveal.js, PowerPoint (pptx), and beamer 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.

When reading pptx, the speaker notes of each slide are read into a `notes` div at the end of the slide's content, so a pptx file converted to Markdown and back keeps its notes.

Speaker notes on the title slide (PowerPoint)

For PowerPoint output, the title slide is generated from the document's YAML metadata block. To add speaker notes to this slide, use a `notes` field in the metadata:

```
---
title: My Presentation
author: Jane Doe
notes: |
  Welcome everyone to this presentation.

  Remember to introduce yourself and mention the key topics.
---
```

The `notes` field can contain multiple paragraphs and Markdown formatting.

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:

```
:::::::::::::: {.columns}
::: {.column width="40%"}
contents...
:::
::: {.column width="60%"}
contents...
:::
::::::::::::::
```

Note: Specifying column widths does not currently work for PowerPoint.

Additional columns attributes in beamer

The div containers with classes `columns` and `column` can optionally have an `align` attribute. The class `columns` can optionally have a `totalwidth` attribute or an `onlytextwidth` class.

```
:::::::::::::: {.columns align=center totalwidth=8em}
::: {.column width="40%"}
contents...
:::
::: {.column width="60%" align=bottom}
contents...
:::
::::::::::::::
```

The `align` attributes on `columns` and `column` can be used with the values `top`, `top-baseline`, `center` and `bottom` to vertically align the columns. It defaults to `top` in `columns`.

The `totalwidth` attribute limits the width of the columns to the given value.

```
:::::::::::::: {.columns align=top .onlytextwidth}
::: {.column width="40%" align=center}
contents...
:::
::: {.column width="60%"}
contents...
:::
::::::::::::::
```

The class `onlytextwidth` sets the `totalwidth` to `\textwidth`.

See Section 12.7 of the Beamer User's Guide for more details.

Frame attributes in beamer

Sometimes it is necessary to add the LaTeX `[fragile]` option to a frame in beamer (for example, when using the `minted` environment). This can be forced by adding the `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: `allowdisplaybreaks`, `allowframebreaks`, `b`, `c`, `s`, `t`, `environment`, `label`, `plain`, `shrink`, `standout`, `noframenumbering`, `squeeze`. `allowframebreaks` is recommended especially for bibliographies, as it allows multiple slides to be created if the content overfills the frame:

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

In addition, the `frameoptions` attribute may be used to pass arbitrary frame options to a beamer slide:

```
# Heading {frameoptions="squeeze,shrink,customoption=foobar"}
```

Background in reveal.js, beamer, and pptx

Background images can be added to self-contained reveal.js slide shows, beamer slide shows, and pptx slide shows.

On all slides (beamer, reveal.js, pptx)

With beamer and reveal.js, the configuration option `background-image` can be used either in the YAML metadata block or as a command-line variable to get the same image on every slide.

Note that for reveal.js, the `background-image` will be used as a `parallaxBackgroundImage` (see below).

For pptx, you can use a `--reference-doc` in which background images have been set on the relevant layouts.

`parallaxBackgroundImage` (reveal.js)

For reveal.js, there is also the reveal.js-native option `parallaxBackgroundImage`, which produces a parallax scrolling background. You must also set `parallaxBackgroundSize`, and can optionally set `parallaxBackgroundHorizontal` and `parallaxBackgroundVertical` to configure the scrolling behaviour. See the reveal.js documentation for more details about the meaning of these options.

In reveal.js's overview mode, the parallaxBackgroundImage will show up only on the first slide.

On individual slides (reveal.js, pptx)

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

As the HTML writers pass unknown attributes through, other reveal.js background settings also work on individual slides, including `background-size`, `background-repeat`, `background-color`, `transition`, and `transition-speed`. (The `data-` prefix will automatically be added.)

Note: `data-background-image` is also supported in pptx for consistency with reveal.js – if `background-image` isn't found, `data-background-image` will be checked.

On the title slide (reveal.js, pptx)

To add a background image to the automatically generated title slide for reveal.js, use the `title-slide-attributes` variable in the YAML metadata block. It must contain a map of attribute names and values. (Note that the `data-` prefix is required here, as it isn't added automatically.)

For pptx, pass a `--reference-doc` with the background image set on the "Title Slide" layout.

Example (reveal.js)

```
---
title: My Slide Show
parallaxBackgroundImage: /path/to/my/background_image.png
title-slide-attributes:
    data-background-image: /path/to/title_image.png
    data-background-size: contain
---

## Slide One

Slide 1 has background_image.png as its background.

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

Slide 2 has a special image for its background, even though the heading has no content.
```

EPUBs

EPUB Metadata

There are two ways to specify metadata for an EPUB. The first is to use the `--epub-metadata` option, which takes as its argument an XML file with Dublin Core elements.

The second way is to use YAML, either in a YAML metadata block in a Markdown document, or in a separate YAML file specified with `--metadata-file`. Here is an example of a YAML metadata block with EPUB metadata:

```
---
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: 'urn:doi:10.234234.234/33'
- scheme: ISBN-13
  text: 'urn:isbn:9780000000000'
publisher:  My Press
rights: © 2007 John Smith, CC BY-NC
ibooks:
  version: 1.3.4
...
```

The following fields are recognized:

`identifier` Either a string value, or an object with fields `text` and `scheme` (optional), or a list of such objects. Valid values for `scheme` are `ISBN-10`, `GTIN-13`, `UPC`, `ISMN-10`, `DOI`, `LCCN`, `GTIN-14`, `ISBN-13`, `Legal deposit number`, `URN`, `OCLC number`, `Co-publisher's`

`ISBN-13`, `ISMN-13`, `ISBN-A`, `JP e-code`, `OLCC number`, `JP Magazine ID`, `UPC-12+5`, `BNF Control number`, `ISSN-13`, `ARK`, `Digital file internal version number`. In the `text` field, it is advisable to specify a URN as in the examples above. Note that quotes are needed around a YAML field containing a colon.

`title` Either a string value, or an object with fields `text`, `file-as` (optional) and `type` (optional), or a list of such objects. Valid values for `type` are `main`, `subtitle`, `short`, `collection`, `edition`, `extended`.

`creator` Either a string value, or an object with fields `text`, `role` (optional), and `file-as` (optional), 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` Either a string value, or an object with fields `text`, `authority` (optional), and `term` (optional), or a list of such objects. Valid values for `authority` are either a reserved authority value (currently `AAT`, `BIC`, `BISAC`, `CLC`, `DDC`, `CLIL`, `EuroVoc`, `MEDTOP`, `LCSH`, `NDC`, `Thema`, `UDC`, and `WGS`) or an absolute IRI identifying a custom scheme. Valid values for `term` are defined by the scheme.

`description` A string value.

`type` A string value.

`format` A string value.

`relation` A string value.

`coverage` A string value.

`rights` A string value.

`belongs-to-collection` A string value. Identifies the name of a collection to which the EPUB Publication belongs.

`group-position` The `group-position` field indicates the numeric position in which the EPUB Publication belongs relative to other works belonging to the same `belongs-to-collection` field.

`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.

`accessModes` An array of strings (schema). Defaults to `["textual"]`.

`accessModeSufficient` An array of strings (schema). Defaults to `["textual"]`.

`accessibilityHazards` An array of strings (schema). Defaults to `["none"]`.

`accessibilityFeatures` An array of strings (schema). Defaults to

```
- "alternativeText"
- "readingOrder"
- "structuralNavigation"
```

```
  - "tableOfContents"
```

`accessibilitySummary` A string value.
`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:

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

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.

`epub:type` of first section	`epub:type` of body
prologue	frontmatter
abstract	frontmatter
acknowledgments	frontmatter
copyright-page	frontmatter
dedication	frontmatter
credits	frontmatter
keywords	frontmatter
imprint	frontmatter
contributors	frontmatter
other-credits	frontmatter
errata	frontmatter
revision-history	frontmatter

`epub:type` of first section	`epub:type` of body
titlepage	frontmatter
halftitlepage	frontmatter
seriespage	frontmatter
foreword	frontmatter
preface	frontmatter
frontispiece	frontmatter
appendix	backmatter
colophon	backmatter
bibliography	backmatter
index	backmatter

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:

```
<audio controls="1">
  <source src="https://example.com/music/toccata.mp3"
          data-external="1" type="audio/mpeg">
  </source>
</audio>
```

If the input format already is HTML then `data-external="1"` will work as expected for `<img>` elements. Similarly, for Markdown, external images can be declared with `![img](url){external=1}`. Note that this only works for images; the other media elements have no native representation in pandoc's AST and require the use of raw HTML.

EPUB styling

By default, pandoc will include some basic styling contained in its `epub.css` data file. (To see this, use `pandoc --print-default-data-file epub.css`.) To use a different CSS file, just use the `--css` command line option. A few inline styles are defined in addition; these are essential for correct formatting of pandoc's HTML output.

The `document-css` variable may be set if the more opinionated styling of pandoc's default HTML templates is desired (and in that case the variables defined in Variables for HTML may be used to fine-tune the style).

Chunked HTML

`pandoc -t chunkedhtml` will produce a zip archive of linked HTML files, one for each section of the original document. Internal links will automatically be adjusted to point to the right place, images linked to under the working directory will be incorporated, and navigation links will be added. In addition, a JSON file `sitemap.json` will be included describing the hierarchical structure of the files.

If an output file without an extension is specified, then it will be interpreted as a directory and the zip archive will be automatically unpacked into it (unless it already exists, in which case an error will be raised). Otherwise a `.zip` file will be produced.

The navigation links can be customized by adjusting the template. By default, a table of contents is included only on the top page. To include it on every page, set the `toc` variable manually.

Jupyter notebooks

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:

````
---
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")
```
````

````
## 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}
```
hello
```
:::
::::::

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

````
HTML("""
<script>
console.log("hello");
</script>
<b>HTML</b>
""")
```

::: {.output .execute_result execution_count=2}
```{=html}
<script>
console.log("hello");
</script>
<b>HTML</b>
hello
```
:::
::::::
````

If you include raw HTML or TeX in an output cell, use the raw 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; `--markdown-headings=setext` will cause Setext-style headings to be used; and `--preserve-tabs` will prevent tabs from being turned to spaces.
````

Vimdoc

The Vimdoc writer generates Vim help files and makes use of the following metadata variables:

```
abstract: "A short description"
author: Author
title: Title

# Vimdoc-specific
filename: "definition-lists.txt"
vimdoc-prefix: pandoc
```

A complete header requires `abstract`, `author`, `title` and `filename` to be set. Compiling a file with such metadata produces the following file (assumes `--standalone`, see Templates):

```
*definition-lists.txt*  A short description

                            Title by Author


                                 Type |gO| to see the table of contents.

[...]

 vim:tw=72:sw=4:ts=4:ft=help:norl:et:
```

If `vimdoc-prefix` is set, all non-command tags are prefixed with its value; this is used to prevent tag collision: all headers have a tag (either inferred or explicit) and multiple help pages can have the same header names, therefore collision is to be expected. Let our input be the following markdown file:

```
## Header

`:[range]Fnl {expr}`{#:Fnl}
:   Evaluates {expr} or range

`vim.b`{#vim.b}
```

```
:   Buffer-scoped (`:h b:`) variables for the current buffer. Invalid or unset
    key returns `nil`. Can be indexed with an integer to access variables for a
    specific buffer.

[Span]{#span}
:   generic inline container for phrasing content, which does not inherently
    represent anything.
```

Convert it to vimdoc:

```
------------------------------------------------------------------------------
Header                                                            *header*

:[range]Fnl {expr}                                                  *:Fnl*
    Evaluates {expr} or range
`vim.b`                                                           *vim.b*
    Buffer-scoped (|b:|) variables for the current buffer. Invalid or
    unset key returns `nil`. Can be indexed with an integer to access
    variables for a specific buffer.
Span                                                              *span*
    generic inline container for phrasing content, which does not
    inherently represent anything.
```

Convert it to vimdoc with the metadata variable set (e.g. with `-M vimdoc-prefix=pandoc`)

```
------------------------------------------------------------------------------
Header                                                   *pandoc-header*

:[range]Fnl {expr}                                                  *:Fnl*
    Evaluates {expr} or range
`vim.b`                                                *pandoc-vim.b*
    Buffer-scoped (|b:|) variables for the current buffer. Invalid or
    unset key returns `nil`. Can be indexed with an integer to access
    variables for a specific buffer.
Span                                                    *pandoc-span*
    generic inline container for phrasing content, which does not
    inherently represent anything.
```

`vim.b` and `Span` got their prefixes but not `:Fnl` because ex-commands (those starting with `:`) don't get a prefix, since they are considered unique across help pages.

In both cases `:help b:` became reference `|b:|` (also works with `:h b:`). Links pointing to either https://vimhelp.org/ or https://neovim.io/doc/user become references too.

Vim traditionally wraps at 78, but Pandoc defaults to 72. Use `--columns 78` to match Vim.

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, Man, Typst, 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 `--syntax-highlighting` 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 `--syntax-highlighting`. To get a JSON version of the `pygments` style, for example:

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

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

```
pandoc --syntax-highlighting my.theme
```

If you are not satisfied with the built-in highlighting, or you want to 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.

If you receive an error that pandoc "Could not read highlighting theme", check that the JSON file is encoded with UTF-8 and has no Byte-Order Mark (BOM).

To disable highlighting, use `--syntax-highlighting=none`.

To use a format's idiomatic syntax highlighting instead of pandoc's built-in highlighting, use `--syntax-highlighting=idiomatic`. Currently, `idiomatic` only affects the following formats:

- In reveal.js, it causes reveal.js's highlighting plugin to be used for source code highlighting. The style may be customized by setting the `highlightjs-theme` variable.

- In Typst, it causes Typst's built-in highlighting to be used. (This is also the default for Typst.)
- In LaTeX, it causes the `listings` package to be used. Note that `listings` 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.
- In other formats, `idiomatic` will have the same result as `default`.

Custom Styles

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

Output

By default, pandoc's odt, fodt, 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 doc 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 `div`s and `span`s, respectively.

If you define a Div, Span, or Table with the attribute `custom-style`, pandoc will apply your specified style to the contained elements (with the exception of elements whose function depends on a style, like headings, code blocks, block quotes, or links). So, for example, using the `bracketed_spans` syntax,

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

would produce a 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.

Styles will be defined in the output file as inheriting from normal text (docx) or Default Paragraph Style (odt), if the styles are not yet in your reference doc. 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 or odt output, you don't need to enable any extensions for custom styles to work.

For icml output, you can also set an `object-style` in images:

```
![Image with object style](myImage.jpg){object-style="fixedSizeImage"}
```

In InDesign you'll see that object style given to the image, and you'll be able to customize it, or load its definition from a template of yours.

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. A `custom-style` attribute will be added for each style. Divs will be created to hold the paragraph styles, and Spans to hold the character styles. Table styles will be applied directly to the Table. The odt and fodt readers also support the `styles` extension (`-f odt+styles`), for paragraph and character styles.

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="First Paragraph"}
This is some text.
:::

::: {custom-style="Body Text"}
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="My Block Style"}
> 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 readers and writers

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

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

```
pandoc -t data/sample.lua
pandoc -f my_custom_markup_language.lua -t latex -s
```

If the script is not found relative to the working directory, it will be sought in the `custom` subdirectory of the user data directory (see `--data-dir`).

A custom reader is a Lua script that defines one function, Reader, which takes a string as input and returns a Pandoc AST. See the Lua filters documentation for documentation of the functions that are available for creating pandoc AST elements. For parsing, the lpeg parsing library is available by default. To see a sample custom reader:

```
pandoc --print-default-data-file creole.lua
```

If you want your custom reader to have access to reader options (e.g. the tab stop setting), you give your Reader function a second `options` parameter.

A custom writer is a Lua script that defines a function that specifies how to render each element in a Pandoc AST. See the djot-writer.lua for a full-featured example.

Note that custom writers have no default template. If you want to use `--standalone` with a custom writer, you will need to specify a template manually using `--template` or add a new default template with the name `default.NAME_OF_CUSTOM_WRITER.lua` to the `templates` subdirectory of your user data directory (see Templates).

Reproducible builds

Some of the document formats pandoc targets (such as EPUB, docx, and ODT) include build timestamps in the generated document. That means that the files generated on successive builds will differ, even if the source does not. To avoid this, set the `SOURCE_DATE_EPOCH` environment variable, and the timestamp will be taken from it instead of the current time. `SOURCE_DATE_EPOCH` should contain an integer unix timestamp (specifying the number of seconds since midnight UTC January 1, 1970).

For reproducible builds with LaTeX, you can either specify the `pdf-trailer-id` in the metadata or leave it undefined, in which case pandoc will create a trailer-id based on a hash of the `SOURCE_DATE_EPOCH` and the document's contents.

Some document formats also include a unique identifier. For EPUB, this can be set explicitly by setting the `identifier` metadata field (see EPUB Metadata, above).

Accessible PDFs and PDF archiving standards

PDF is a flexible format, and using PDF in certain contexts requires additional conventions. For example, PDFs are not accessible by default; they define how characters are placed on a page but do not contain semantic information on the content. However, it is possible to generate accessible PDFs, which use tagging to add semantic information to the document.

Pandoc defaults to LaTeX to generate PDF. LaTeX's `\DocumentMetadata` interface supports PDF standards and tagging when using LuaLaTeX; set the `pdfstandard` variable to enable this (see below). For older LaTeX installations, alternative engines must be used.

The PDF standards PDF/A and PDF/UA define further restrictions intended to optimize PDFs for archiving and accessibility. Tagging is commonly used in combination with these standards to ensure best results.

Note, however, that standard compliance depends on many things, including the colorspace of embedded images. Pandoc cannot check this, and external programs must be used to ensure that generated PDFs are in compliance.

LaTeX

Set the `pdfstandard` variable to produce tagged PDFs conforming to PDF/A, PDF/X, or PDF/UA standards. For example:

```
pandoc -V pdfstandard=ua-2 --pdf-engine=lualatex doc.md -o doc.pdf
```

Multiple standards can be combined:

```
---
pdfstandard:
  - ua-2
  - a-4f
---
```

The required PDF version is inferred automatically. This feature requires LuaLaTeX in TeX Live 2025 with LaTeX kernel 2025-06-01 or newer.

ConTeXt

ConTeXt always produces tagged PDFs, but the quality depends on the input. The default ConTeXt markup generated by pandoc is optimized for readability and reuse, not tagging. Enable the `tagging` format extension to force markup that is optimized for tagging. For example:

```
pandoc -t context+tagging doc.md -o doc.pdf
```

A recent `context` version should be used, as older versions contained a bug that led to invalid PDF metadata.

WeasyPrint

The HTML-based engine WeasyPrint includes experimental support for PDF/A and PDF/UA since version 57. Tagged PDFs can be created with

```
pandoc --pdf-engine=weasyprint \
       --pdf-engine-opt=--pdf-variant=pdf/ua-1 ...
```

The feature is experimental and standard compliance should not be assumed.

Prince XML

The non-free HTML-to-PDF converter `prince` has extensive support for various PDF standards as well as tagging. E.g.:

```
pandoc --pdf-engine=prince \
       --pdf-engine-opt=--tagged-pdf ...
```

See the prince documentation for more info.

Typst

Typst 0.12 can produce PDF/A-2b:

```
pandoc --pdf-engine=typst --pdf-engine-opt=--pdf-standard=a-2b ...
```

Word Processors

Word processors like LibreOffice and MS Word can also be used to generate standardized and tagged PDF output. Pandoc does not support direct conversions via these tools. However, pandoc can convert a document to a `docx` or `odt` file, which can then be opened and converted to PDF with the respective word processor. See the documentation for Word and LibreOffice.

Running pandoc as a web server

If you rename (or symlink) the pandoc executable to `pandoc-server`, or if you call pandoc with `server` as the first argument, it will start up a web server with a JSON API. This server exposes most of the conversion functionality of pandoc. For full documentation, see the pandoc-server man page.

If you rename (or symlink) the pandoc executable to `pandoc-server.cgi`, it will function as a CGI program exposing the same API as `pandoc-server`.

`pandoc-server` is designed to be maximally secure; it uses Haskell's type system to provide strong guarantees that no I/O will be performed on the server during pandoc conversions.

Running pandoc as a Lua interpreter

Calling the pandoc executable under the name `pandoc-lua` or with `lua` as the first argument will make it function as a standalone Lua interpreter. The behavior is mostly identical to that of the standalone `lua` executable, version 5.4. All `pandoc.*` packages, as well as the packages `re` and `lpeg`, are available via global variables. Furthermore, the globals `PANDOC_VERSION`, `PANDOC_STATE`, and `PANDOC_API_VERSION` are set at startup. For full documentation, see the pandoc-lua man page.

A note on security

1. Although pandoc itself will not create or modify any files other than those you explicitly ask it to 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 you want to use pandoc in a web application that accepts untrusted user input, you should be aware that running pandoc in this context requires special precautions, or you risk exfiltration of the contents of local files, or worse. In particular:

    - Several input formats (including LaTeX, Org, RST, and Typst) support `include` directives that allow the contents of a file to be included in the output. An untrusted attacker could use these to view the contents of files on the file system. Using the `--sandbox` option will protect against this threat.

    - Several output formats (including RTF, FB2, HTML with `--self-contained`, EPUB, Docx, and ODT) will embed encoded or raw images into the output file. An untrusted attacker could exploit this to view the contents of non-image files on the file system. Using the `--sandbox` option will protect against this threat, but will also prevent including images in these formats.

    - In reading HTML files, pandoc will attempt to include the contents of `iframe` elements by fetching content from the local file or URL specified by `src`. If untrusted HTML is processed on a server, this has the potential to reveal anything readable by the process running the server or to enable server-side request forgery (SSRF) attacks (CVE-2025-51591). To mitigate such attacks, use `--sandbox` or `-f html+raw_html` (which causes the whole `iframe` to be parsed as a raw HTML block).

    - Using pandoc with `--pdf-engine` introduces additional risks, which cannot be mitigated by using `--sandbox`. For example, in `wkhtmltopdf` the `footer-html` and `header-html` metadata fields will be passed directly to `wkhtmltopdf` as `--footer-html` and `--header-html` arguments, and by using a `file:` URI with these, one could exfiltrate local files. In addition, using `--pdf-engine=wkhtmltopdf` with `-f html+raw_html` enables an SSRF vulnerability in wkhtmltopdf (CVE-2022-35583). Note that `wkhtmltopdf` is deprecated and its authors recommend not to use it with any untrusted input. However, there may be similar issues with other pdf-engines. Anyone including a pdf-engine in a

pandoc command run on untrusted input should audit the engine carefully and carefully control the use of `--pdf-engine-opt`.

Here are some alternatives for safe use of pandoc in a web application:

- Use `pandoc server` to spin up a local web server, and have your application make requests to it. The server runs in sandboxed mode and will not be able to read files or use filters.
- Use the WASM version of pandoc. This allows the use of Lua filters, but everything runs in an isolated WASM sandbox.
- Use `pandoc --sandbox`. However, be aware that the sandbox function only affects readers and writers, and security issues can arise if you use it together with `--filter` or `--pdf-engine`.

3. 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 URLs and attributes. To be safe, you should run all HTML generated from untrusted user input through an HTML sanitizer.
4. 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. Note that the `commonmark` parser (including `commonmark_x` and `gfm`) is much less vulnerable to pathological performance than the `markdown` parser, so it is a better choice when processing untrusted input.
5. 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. (This corresponds to the use of the `--sandbox` option on the command line.)

Authors

Copyright 2006–2024 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.