

Pandoc User's Guide

John MacFarlane

December 7, 2009

Pandoc is a Haskell library for converting from one markup format to another, and a command-line tool that uses this library. It can read markdown and (subsets of) reStructuredText, HTML, and LaTeX; and it can write markdown, reStructuredText, HTML, LaTeX, ConTeXt, RTF, DocBook XML, OpenDocument XML, ODT, GNU Texinfo, MediaWiki markup, groff man pages, and S5 HTML slide shows. Pandoc's enhanced version of markdown includes syntax for footnotes, tables, flexible ordered lists, definition lists, delimited code blocks, superscript, subscript, strikeout, title blocks, automatic tables of contents, embedded LaTeX math, and markdown inside HTML block elements. (These enhancements can be disabled if a drop-in replacement for `Markdown.pl` is desired.)

In contrast to most existing tools for converting markdown to HTML, which use regex substitutions, 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, 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.

© 2006–9 John MacFarlane (jgm at berkeley dot 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.) Contributors: Recai Oktaş (build system, debian package, wrapper scripts), Peter Wang (Texinfo writer), Andrea Rossato (OpenDocument writer).

Using Pandoc

If you run `pandoc` without arguments, it will accept input from `stdin`. If you run it with file names as arguments, it will take input from those files. By default, `pandoc` writes its output to `stdout`.¹ If you want to write to a file, use the `-o` option:

¹The exception is for `odt`. Since this is a binary output format, an output file must be specified explicitly.

```
pandoc -o hello.html hello.txt
```

Note that you can specify multiple input files on the command line. `pandoc` will concatenate them all (with blank lines between them) before parsing:

```
pandoc -s ch1.txt ch2.txt refs.txt > book.html
```

(The `-s` option here tells `pandoc` to produce a standalone HTML file, with a proper header, rather than a fragment. For more details on this and many other command-line options, see below.)

The format of the input and output can be specified explicitly using command-line options. The input format can be specified using the `-r/--read` or `-f/--from` options, the output format using the `-w/--write` or `-t/--to` options. 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 output formats include `markdown`, `latex`, `context` (ConTeXt), `html`, `rtf` (rich text format), `rst` (reStructuredText), `docbook` (DocBook XML), `opendocument` (OpenDocument XML), `odt` (OpenOffice text document), `texinfo`, (GNU Texinfo), `mediawiki` (MediaWiki markup), `man` (groff man), and `s5` (which produces an HTML file that acts like powerpoint).

Supported input formats include `markdown`, `html`, `latex`, and `rst`. Note that the `rst` reader only parses a subset of reStructuredText syntax. For example, it doesn't handle tables, option lists, or footnotes. But for simple documents it should be adequate. The `latex` and `html` readers are also limited in what they can do. Because the `html` reader is picky about the HTML it parses, it is recommended that you pipe HTML through HTML Tidy before sending it to `pandoc`, or use the `html2markdown` script described below.

If you don't specify a reader or writer explicitly, `pandoc` will try to determine the input and output format from the extensions of the input and output 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 unless explicitly specified.

Character encodings

All input is assumed to be in the UTF-8 encoding, and all output is in UTF-8. If your local character encoding is not UTF-8 and you use accented or foreign characters, you should pipe the input and output through `iconv`. For example,

```
iconv -t utf-8 source.txt | pandoc | iconv -f utf-8 > output.html
```

will convert `source.txt` from the local encoding to UTF-8, then convert it to HTML, then convert back to the local encoding, putting the output in `output.html`.

The wrapper scripts (described below) automatically convert the input from the local encoding to UTF-8 before running them through `pandoc`, then convert the output back to the local encoding.

Wrappers

Three wrapper scripts, `markdown2pdf`, `html2markdown`, and `hsmarkdown`, are included in the standard Pandoc installation. (The Windows binary package does not include `html2markdown`, which is a POSIX shell script. It does include portable Haskell versions of `markdown2pdf` and `hsmarkdown`.)

1. `markdown2pdf` produces a PDF file from markdown-formatted text, using `pandoc` and `pdflatex`. The default behavior of `markdown2pdf` is to create a file with the same base name as the first argument and the extension `pdf`; thus, for example,

```
markdown2pdf sample.txt endnotes.txt
```

will produce `sample.pdf`. (If `sample.pdf` exists already, it will be backed up before being overwritten.) An output file name can be specified explicitly using the `-o` option:

```
markdown2pdf -o book.pdf chap1 chap2
```

If no input file is specified, input will be taken from `stdin`. All of `pandoc`'s options will work with `markdown2pdf` as well.

`markdown2pdf` assumes that `pdflatex` is in the path. It also assumes that the following LaTeX packages are available: `unicode`, `fancyhdr` (if you have verbatim text in footnotes), `graphicx` (if you use images), `array` (if you use tables), and `ulem` (if you use strikeout text). If they are not already included in your LaTeX distribution, you can get them from CTAN. A full TeX Live or MacTeX distribution will have all of these packages.

2. `html2markdown` grabs a web page from a file or URL and converts it to markdown-formatted text, using `tidy` and `pandoc`.

All of `pandoc`'s options will work with `html2markdown` as well. In addition, the following special options may be used. The special options must be separated from the `html2markdown` command and any regular `Pandoc` options by the delimiter `--`:

```
html2markdown -o out.txt -- -e latin1 -g curl google.com
```

The `-e` or `--encoding` option specifies the character encoding of the HTML input. If this option is not specified, and input is not from `stdin`, `html2markdown` will attempt to determine the page's character encoding from the "Content-type" meta tag. If this is not present, UTF-8 is assumed.

The `-g` or `--grabber` option specifies the command to be used to fetch the contents of a URL:

```
html2markdown -g 'curl --user foo:bar' www.mysite.com
```

If this option is not specified, `html2markdown` searches for an available program (`wget`, `curl`, or a text-mode browser) to fetch the contents of a URL.

`html2markdown` requires HTML Tidy, which must be in the path. It uses `iconv` for character encoding conversions; if `iconv` is absent, it will still work, but it will treat everything as UTF-8.

3. `hsmarkdown` is designed to be used as a drop-in replacement for `Markdown.pl`. It forces `pandoc` to convert from markdown to HTML, and to use the `--strict` flag for maximal compliance with official markdown syntax. (All of `Pandoc`'s syntax extensions and variants, described below, are disabled.) No other command-line options are allowed. (In fact, options will be interpreted as filenames.)

As an alternative to using the `hsmarkdown` script, the user may create a symbolic link to `pandoc` called `hsmarkdown`. When invoked under the name `hsmarkdown`, `pandoc` will behave as if the `--strict` flag had been selected, and no command-line options will be recognized. However, this approach does not work under Cygwin, due to problems with its simulation of symbolic links.

Command-line options

Various command-line options can be used to customize the output. For further documentation, see the `pandoc(1)` man page.

- f, --from, -r, or --read *format*** specifies the input format (the format Pandoc will be converting *from*). *format* can be `native`, `markdown`, `rst`, `html`, or `latex`. (`+lhs` can be appended to indicate that the input should be treated as literate Haskell source. See Literate Haskell support, below.)
- t, --to, -w, or --write *format*** specifies the output format — the format Pandoc will be converting *to*. *format* can be `native`, `html`, `s5`, `docbook`, `opendocument`, `latex`, `context`, `markdown`, `man`, `rst`, and `rtf`. (`+lhs` can be appended to indicate that the output should be treated as literate Haskell source. See Literate Haskell support, below.)
- s or --standalone** indicates that a standalone document is to be produced (with appropriate headers and footers), rather than a fragment.
- o or --output *filename*** sends output to *filename*. If this option is not specified, or if its argument is `-`, output will be sent to stdout. (Exception: if the output format is `odt`, output to stdout is disabled.)
- p or --preserve-tabs** causes tabs in the source text to be preserved, rather than converted to spaces (the default).
- tab-stop *tabstop*** sets the number of spaces per tab to *tabstop* (defaults to 4).
- strict** specifies that strict markdown syntax is to be used, without pandoc’s usual extensions and variants (described below). When the input format is HTML, this means that constructs that have no equivalents in standard markdown (e.g. definition lists or strikethrough text) will be parsed as raw HTML.
- reference-links** causes reference-style links to be used in markdown and reStructuredText output. By default inline links are used.
- R or --parse-raw** causes the HTML and LaTeX readers to parse HTML codes and LaTeX environments that it can’t translate as raw HTML or LaTeX. Raw HTML can be printed in markdown, reStructuredText, HTML, and S5 output; raw LaTeX can be printed in markdown, reStructuredText, LaTeX, and ConTeXt output. The default is for the readers to omit untranslatable HTML codes and LaTeX environments. (The LaTeX reader does pass through untranslatable LaTeX *commands*, even if `-R` is not specified.)
- C or --custom-header *filename*** can be used to specify a custom document header. To see the headers used by default, use the `-D` option: for example, `pandoc -D html` prints the default HTML header. Implies `--standalone`.
- toc or --table-of-contents** includes an automatically generated table of contents (or, in the case of `latex`, `context`, and `rst`, an instruction to create one) in the output document. This option has no effect with `man`, `docbook`, or `s5` output formats.

- template=***file* uses *file* as a custom template for the generated document. Implies `-s`. See Templates below for a description of template syntax. If this option is not used, a default template appropriate for the output format will be used. See also `-D/--print-default-template`.
- v** *key=val*, **--variable=***key:val* sets the template variable *key* to the value *val* when rendering the document in standalone mode. This is only useful when the `--template` option is used to specify a custom template, since pandoc automatically sets the variables used in the default templates.
- c** or **--css** *filename* allows the user to specify a custom stylesheet that will be linked to in HTML and S5 output. This option can be used repeatedly to include multiple stylesheets. They will be included in the order specified. Implies `--standalone`.
- H** or **--include-in-header** *filename* includes the contents of *filename* (verbatim) at the end of the document 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** or **--include-before-body** *filename* includes the contents of *filename* (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.
- A** or **--include-after-body** *filename* includes the contents of *filename* (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.
- reference-odt** *filename* uses the specified file as a style reference in producing an 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

`$HOME/.pandoc`

(on unix) or

`C:\Documents And Settings\USERNAME\Application Data\pandoc`

(on Windows). If this is not found either, sensible defaults will be used.

- D** or **--print-default-template *format*** prints the default template for an output *format*. (See `-t` for a list of possible *formats*.)
- T** or **--title-prefix *string*** includes *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). (See below on Title Blocks.) Implies `--standalone`.
- S** or **--smart** causes `pandoc` to produce typographically correct output, along the lines of John Gruber's Smartypants. Straight quotes are converted to curly quotes, `---` to dashes, and `...` to ellipses. Nonbreaking spaces are inserted after certain abbreviations, such as "Mr." (Note: This option is only significant when the input format is `markdown`. It is selected automatically when the output format is `latex` or `context`.)
- m[*url*]** or **--latexmathml[=*url*]** causes `pandoc` to use the LaTeXMathML script to display TeX math in HTML or S5. If a local copy of `LaTeXMathML.js` is available on the webserver where the page will be viewed, provide a *url* and a link will be inserted in the generated HTML or S5. If no *url* is provided, the contents of the script will be inserted directly; this provides portability at the price of efficiency. If you plan to use math on several pages, it is much better to link to a copy of `LaTeXMathML.js`, which can be cached. (See `--jsmath`, `--gladtex`, and `--mimetex` for alternative ways of dealing with math in HTML.)
- jsmath=[*url*]** causes `pandoc` to use the jsMath script to display TeX math in HTML or S5. The *url* should point to the jsMath load script (e.g. `jsMath/easy/load.js`). If it is provided, a link to it will be included in the header of standalone HTML documents. (See `--latexmathml`, `--mimetex`, and `--gladtex` for alternative ways of dealing with math in HTML.)
- gladtex=[*url*]** causes TeX formulas to be enclosed in `<eq>` tags in HTML or S5 output. This output can then be processed by `gladTeX` to produce links to images with the typeset formulas. (See `--latexmathml`, `--jsmath`, and `--mimetex` for alternative ways of dealing with math in HTML.)
- mimetex=[*url*]** causes TeX formulas to be replaced by `` tags linking to the `mimeTeX` CGI script, which will produce images with the typeset formulas. (See `--latexmathml`, `--jsmath`, and `--gladtex` for alternative ways of dealing with math in HTML.)
- i** or **--incremental** causes all lists in S5 output to be displayed incrementally by default (one item at a time). The normal default is for lists to be displayed all at once.
- xetex** creates LaTeX output suitable for processing by XeTeX.

- N** or **--number-sections** causes sections to be numbered in LaTeX, ConTeXt, or HTML output. By default, sections are not numbered.
- no-wrap** disables text-wrapping in output. By default, text is wrapped appropriately for the output format.
- sanitize-html** sanitizes HTML (in markdown or HTML input) using a whitelist. Unsafe tags are replaced by HTML comments; unsafe attributes are omitted. URIs in links and images are also checked against a whitelist of URI schemes.
- email-obfuscation=*none|javascript|references*** specifies 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. If **--strict** is specified, *references* is used regardless of the presence of this option.
- id-prefix=*string*** specifies a prefix to be added to all automatically generated identifiers in HTML output. This is useful for preventing duplicate identifiers when generating fragments to be included in other pages.
- indented-code-classes=*classes*** specifies classes to use for indented code blocks—for example, `perl`, `numberLines` or `haskell`. Multiple classes may be separated by spaces or commas.
- dump-args** is intended to make it easier to create wrapper scripts that use Pandoc. It causes Pandoc to dump information about the arguments with which it was called to `stdout`, then `exit`. The first line printed is the name of the output file specified using the `-o` or `--output` option, or `-` if output would go to `stdout`. The remaining lines, if any, list command-line arguments. These will include the names of input files and any special options passed after `--` on the command line. So, for example,

```
pandoc --dump-args -o foo.html -s foo.txt \
  appendix.txt -- -e latin1
```

will cause the following to be printed to `stdout`:

```
foo.html foo.txt appendix.txt -e latin1
```

- ignore-args** causes Pandoc to ignore all command-line arguments. 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
```

-v or **--version** prints the version number to STDERR.

-h or **--help** prints a usage message to STDERR.

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 --print-default-template=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 `FORMAT.template` in

```
$HOME/.pandoc/templates
```

(on unix) or

```
C:\Documents And Settings\USERNAME\Application Data\pandoc\templates
```

(on Windows).

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

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

will be replaced by the document title.

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

Some variables are set automatically by pandoc. These vary somewhat depending on the output format, but include:

legacy-header contents specified by `-C/--custom-header`

header-includes contents specified by `-H/--include-in-header` (may have multiple values)

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

body body of document

title title of document, as specified in title block

author author of document, as specified in title block (may have multiple values)

date date of document, as specified in title block

Variables may be set at the command line using the `-V/--variable` option. This allows users to include custom variables in their templates.

Templates may contain conditionals. The syntax is as follows:

```
$if(variable)$  
X  
$else$  
Y  
$endif$
```

This will include `X` in the template if `variable` has a non-null value; otherwise it will include `Y`. `X` and `Y` are placeholders for any valid template text, and may include interpolated variables or other conditionals. The `$else$` section may be omitted.

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

```
$for(author)$  
<meta name="author" content="$author$" />  
$endfor$
```

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

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

Pandoc's markdown vs. standard markdown

In parsing markdown, Pandoc departs from and extends standard markdown in a few respects. (To run Pandoc on the official markdown test suite, type `make test-markdown`.) Except where noted, these differences can be suppressed by specifying the `--strict` command-line option or by using the `hsmarkdown` wrapper.

Backslash escapes

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

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

one will get

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

instead of

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

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

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

A backslash-escaped space is parsed as a nonbreaking space. It will appear in TeX output as `'~'` and in HTML and XML as `'\ '` or `'\ '`.

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 `'
'`. This is a nice alternative to markdown's "invisible" way of indicating hard line breaks using two trailing spaces on a line.

Subscripts and superscripts

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

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

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

Strikeout

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

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

Nested Lists

Pandoc behaves differently from standard markdown on some “edge cases” involving lists. Consider this source:

1. First
2. Second:
 - Fee
 - Fie
 - Foe
3. Third

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

Ordered Lists

Unlike standard markdown, Pandoc allows ordered list items to be marked with uppercase and lowercase letters and roman numerals, in addition to arabic numerals. (This behavior can be turned off using the `--strict` option.) List markers may be enclosed in parentheses or followed by a single right-parentheses or period. They must be separated from the text that follows by at least one space, and, if the list marker is a capital letter with a period, by at least two spaces.²

²The point of this rule is to ensure that normal paragraphs starting with people’s initials, like B. Russell was an English philosopher. do not get treated as list items.

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

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

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

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

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

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

Definition lists

Pandoc supports definition lists, using a syntax inspired by PHP Markdown Extra and reStructuredText:³

Term 1

: Definition 1

Term 2 with *inline markup*

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

³I have also been influenced by the suggestions of David Wheeler.

```
: Definition 2

    { some code, part of Definition 2 }

Third paragraph of definition 2.
```

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

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

```
Term 1
  ~ Definition 1
Term 2
  ~ Definition 2a
  ~ Definition 2b
```

Reference links

Pandoc allows implicit reference links with just a single set of brackets. So, the following links are equivalent:

1. Here's my [link]
2. Here's my [link] []

```
[link]: linky.com
```

(Note: Pandoc works this way even if `--strict` is specified, because `Markdown.pl 1.0.2b7` allows single-bracket links.)

Footnotes

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

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

`[^1]`: Here is the footnote.

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

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

```
{ some.code }
```

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

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

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

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

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

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

Inline and regular footnotes may be mixed freely.

Tables

Two kinds of tables may be used. Both kinds presuppose the use of a fixed-width font, such as Courier.

Simple tables look like this:

Right	Left	Center	Default
-----	-----	-----	-----
12	12	12	12
123	123	123	123

```
1      1      1      1
```

Table: Demonstration of simple table syntax.

The headers and table rows must each fit on one line. Column alignments are determined by the position of the header text relative to the dashed line below it:⁴

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

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

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

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

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

Multiline tables allow headers and table rows to span multiple lines of text. Here is an example:

```
-----
Centered  Default      Right Left
Header   Aligned        Aligned Aligned
-----
```

⁴This scheme is due to Michel Fortin, who proposed it on the Markdown discussion list.

```

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

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

```

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

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

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

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

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

```

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

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

```

Table: Here's a multiline table without headers.

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

Delimited Code blocks

In addition to standard indented code blocks, Pandoc supports *delimited* code blocks. These begin with a row of three or more tildes (~) and end with a row

of tildes that must be at least as long as the starting row. Everything between the tilde-lines is treated as code. No indentation is necessary:

```
~~~~~  
{code here}  
~~~~~
```

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

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

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

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

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

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

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

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

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

Title blocks

If the file begins with a title block

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

it will be parsed as bibliographic information, not regular text. (It will be used, for example, in the title of standalone LaTeX or HTML output.) The block may contain just a title, a title and an author, or all three lines. Each must begin with a % and fit on one line. The title may contain standard inline formatting. If you want to include an author but no title, or a title and a date but no author, you need a blank line:

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

Titles will be written 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.

Markdown in HTML blocks

While standard markdown leaves HTML blocks exactly as they are, Pandoc treats text between HTML tags as markdown. Thus, for example, Pandoc will turn

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

into

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

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

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

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

Header identifiers in HTML

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

- Remove all formatting, links, etc.
- Remove all punctuation, except underscores, hyphens, periods, and tildes.
- Replace all spaces and newlines with hyphens.
- Convert all alphabetic characters to lowercase.

- Remove everything up to the first letter (identifiers may not begin with a number or punctuation mark).
- If nothing is left after this, use the identifier `section`.

Thus, for example,

Header	Identifier
Header identifiers in HTML	<code>header-identifiers-in-html</code>
<i>Dogs?</i> —in <i>my</i> house?	<code>dogs--in-my-house</code>
HTML, S5, or RTF?	<code>html-s5-or-rtf</code>
3. Applications	<code>applications</code>
33	<code>section</code>

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

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

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

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

Blank lines before headers and blockquotes

Standard markdown syntax does not require a blank line before a header or blockquote. 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 `>` or `#` 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.
```

Math

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

TeX math will be printed in all output formats. In Markdown, reStructuredText, LaTeX, and ConTeXt output, it will appear verbatim between `$` characters.

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

In Texinfo output, it will be rendered inside a `@math` command.

In groff man output, it will be rendered verbatim without `'`s.

In MediaWiki output, it will be rendered inside `<math>` tags.

In RTF, Docbook, and OpenDocument output, it will be rendered, as far as possible, using unicode characters, and will otherwise appear verbatim. Unknown commands and symbols, and commands that cannot be dealt with this way (like `\frac`), will be rendered verbatim. So the results may be a mix of raw TeX code and properly rendered unicode math.

In HTML and S5 output, the way math is rendered will depend on the command-line options selected:

1. The default is to render TeX math as far as possible using unicode characters, as with RTF, Docbook, and OpenDocument output. Formulas are put inside a `span` with `class="math"`, so that they may be styled differently from the surrounding text if needed.
2. If the `--latexmathml` option is used, TeX math will be displayed between `$` or `$$` characters and put in `` tags with `class LaTeX`. The LaTeXMathML script will be used to render it as formulas. (This trick does not work in all browsers, but it works in Firefox. In browsers that do not support LaTeXMathML, TeX math will appear verbatim between `$` characters.)
3. If the `--jsmath` option is used, TeX math will be put inside `` tags (for inline math) or `<div>` tags (for display math) with `class math`. The jsMath script will be used to render it.
4. If the `--mimetex` option is used, the mimeTeX CGI script will be called to generate images for each TeX formula. This should work in all browsers. The `--mimetex` option takes an optional URL as argument. If no URL is specified, it will be assumed that the mimeTeX CGI script is at `/cgi-bin/mimetex.cgi`.

5. If the `--gladtex` option is used, TeX formulas will be enclosed in `<eq>` tags in the HTML output. The resulting `htex` file may then be processed by `gladTeX`, which will produce image files for each formula and an `html` file with links to these images. So, the procedure is:

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

Inline TeX

Inline TeX commands will be preserved and passed unchanged to the LaTeX and ConTeXt writers. Thus, for example, you can use LaTeX to include BibTeX citations:

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

Note that in LaTeX environments, like

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

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

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

Custom headers

When run with the “standalone” option (`-s`), `pandoc` creates a standalone file, complete with an appropriate header. To see the default headers used for `html` and `latex`, use the following commands:

```
pandoc -D html
```

```
pandoc -D latex
```

If you want to use a different header, just create a file containing it and specify it on the command line as follows:

```
pandoc --custom-header=MyHeaderFile
```

Producing S5 with Pandoc

Producing an S5 web-based slide show with Pandoc is easy. A title page is constructed automatically from the document's title block (see above). Each section (with a level-one header) produces a single slide. (Note that if the section is too big, the slide will not fit on the page; S5 is not smart enough to produce multiple pages.)

Here's the markdown source for a simple slide show, `eating.txt`:

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

# In the morning

- Eat eggs
- Drink coffee

# In the evening

- Eat spaghetti
- Drink wine
```

To produce the slide show, simply type

```
pandoc -w s5 -s eating.txt > eating.html
```

and open up `eating.html` in a browser.

Note that by default, the S5 writer produces 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 (that is, to display incrementally without the `-i` option and all at once with the `-i` option), put it in a block quote:

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

In this way incremental and nonincremental lists can be mixed in a single document.

Note: the S5 file produced by pandoc with the `-s/--standalone` option embeds the javascript and CSS required to show the slides. Thus it does not depend on any additional files: you can send the HTML file to others, and they will be able to view the slide show just by opening it. However, if you intend to produce several S5 slide shows, and you are displaying them on your own website, it is better to keep the S5 javascript and CSS files separate from the slide shows themselves, so that they may be cached. The best approach in this case is to use pandoc without the `-s` option to produce the body of the S5 document, which can then be inserted into an HTML template that links to the javascript and CSS files required by S5. (See the instructions on the S5 website.) Alternatively, you may use `-s` together with the `--template` option to specify a custom template.

You can change the style of the slides by putting customized CSS files in

```
$HOME/.pandoc/s5/default
```

(on unix) or

```
C:\Documents And Settings\USERNAME\Application Data\pandoc\reference.odt
```

(on Windows). 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's pandoc data directory.

Literate Haskell support

If you append `+lhs` to an appropriate input or output format (`markdown`, `rst`, or `latex` for input or output; `html` for output only), 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.
- In markdown output, code blocks with class `haskell` 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, headers 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.