NAME

pandoc - general markup converter

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. It can read markdown and (subsets of) Textile, reStructuredText, HTML, and LaTeX; and it can write plain text, markdown, reStructuredText, HTML, LaTeX, ConTeXt, RTF, DocBook XML, OpenDocument XML, ODT, GNU Texinfo, MediaWiki markup, EPUB, Textile, groff man pages, Emacs Org-Mode, and Slidy or 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, citations, and markdown inside HTML block elements. (These enhancements, described below under Pandoc's markdown, can be disabled using the `--strict` option.)

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.

Using Pandoc

If no *input-file* is specified, input is read from *stdin*. Otherwise, the *input-files* are concatenated (with a blank line between each) and used as input. Output goes to *stdout* by default (though output to *stdout* is disabled for the `odt` and `epub` output formats). For output to a file, use the `-o` option:

```
pandoc -o output.html input.txt
```

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

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

If multiple input files are given, `pandoc` will concatenate them all (with blank lines between them) before parsing.

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 are listed below under the `-t/--to` option. Supported input formats are listed below under the `-f/--from` option. Note that the `rst`, `textile`, `latex`, and `html` readers are not complete; there are some constructs that they do not parse.

If the input or output format is not specified explicitly, `pandoc` will attempt to guess it 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.

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
```

OPTIONS

`-f` *FORMAT*, `-r` *FORMAT*,
: `--from=`*FORMAT*, `--read=`*FORMAT* Specify input format. *FORMAT* can be `native` (native Haskell), `json` (JSON version of native AST), `markdown` (markdown), `textile` (Textile), `rst` (reStructuredText), `html` (HTML), or `latex` (LaTeX). If `+lhs` is appended to `markdown`, `rst`, or `latex`, the input will be treated as literate Haskell source: see Literate Haskell support, below.

`-t` *FORMAT*, `-w` *FORMAT*,
: `--to=`*FORMAT*, `--write=`*FORMAT* Specify output format. *FORMAT* can be `native` (native Haskell), `json` (JSON version of native AST), `plain` (plain text), `markdown` (markdown), `rst` (reStructuredText), `html` (HTML), `latex` (LaTeX), `context` (ConTeXt), `man` (groff man), `mediawiki` (MediaWiki markup), `textile` (Textile), `org` (Emacs Org-Mode), `texinfo` (GNU Texinfo), `docbook` (DocBook XML), `opendocument` (OpenDocument XML), `odt` (OpenOffice text document), `epub` (EPUB book), `slidy` (Slidy HTML and javascript slide show), `s5` (S5 HTML and javascript slide show), or `rtf` (rich text format). Note that `odt` and `epub` output will not be directed to *stdout*; an output filename must be specified using the `-o/--output` option. If `+lhs` is appended to `markdown`, `rst`, `latex`, or `html`, the output will be rendered as literate Haskell source: see Literate Haskell support, below.

`-s`, `--standalone`
: Produce output with an appropriate header and footer (e.g. a standalone HTML, LaTeX, or RTF file, not a fragment).

`-o` *FILE*, `--output=`*FILE*
: Write output to *FILE* instead of *stdout*. If *FILE* is `-`, output will go to *stdout*. (Exception: if the output format is `odt` or `epub`, output to stdout is disabled.)

`-p`, `--preserve-tabs`
: Preserve tabs instead of converting them to spaces (the default).

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

`--strict`
: Use strict markdown syntax, with no pandoc extensions or variants. When the input format is HTML, this means that constructs that have no equivalents in standard markdown (e.g. definition lists or strikeout text) will be parsed as raw HTML.

`--normalize`
: Normalize the document after reading: merge adjacent `Str` or `Emph` elements, for example, and remove repeated `Space`s.

`--reference-links`
: Use reference-style links, rather than inline links, in writing markdown or reStructuredText. By default inline links are used.

`-R`, `--parse-raw`
: Parse untranslatable HTML codes and LaTeX environments as raw HTML or LaTeX, instead of ignoring them. Affects only HTML and LaTeX input. Raw HTML can be printed in markdown, reStructuredText, HTML, Slidy, 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.)

`-S`, `--smart`
: Produce typographically correct output, converting straight quotes to curly quotes, `---` and `--` to dashes, ande `...` to ellipses. Nonbreaking spaces are inserted after certain abbreviations, such as "Mr." (Note: This option is significant only when the input format is `markdown` or `textile`. It is selected automatically when the input format is `textile` or the output format is `latex` or `context`.)

`-5`, `--html5`
: Produce HTML5 instead of HTML4. This option has no effect for writers other than `html`.

`-m` [*URL*], `--latexmathml`[`=`*URL*]
: Use the LaTeXMathML script to display embedded TeX math in HTML output. To insert a link to a local copy of the `LaTeXMathML.js` script, provide a *URL*. If no *URL* is provided, the contents of the script will be inserted directly into the HTML header, preserving 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 the script, so it can be cached.

`--mathml`[`=`*URL*]
: Convert TeX math to MathML. In standalone mode, a small javascript (or a link to such a script if a *URL* is supplied) will be inserted that allows the MathML to be viewed on some browsers.

`--jsmath`[`=`*URL*]
: Use jsMath to display embedded TeX math in HTML output. The *URL* should point to the jsMath load script (e.g. `jsMath/easy/load.js`); if provided, it will be linked to in the header of standalone HTML documents. If a *URL* is not provided, no link to the jsMath load script will be inserted; it is then up to the author to provide such a link in the HTML template.

`--mathjax=`*URL*
: Use MathJax to display embedded TeX math in HTML output. The *URL* should point to the `MathJax.js` load script.

`--gladtex`
: Enclose TeX math in `<eq>` tags in HTML output. These can then be processed by gladTeX to produce links to images of the typeset formulas.

`--mimetex`[`=`*URL*]
: Render TeX math using the mimeTeX CGI script. If *URL* is not specified, it is assumed that the script is at `/cgi-bin/mimetex.cgi`.

`--webtex`[`=`*URL*]
: Render TeX formulas using an external script that converts TeX formulas to images. The formula will be concatenated with the URL provided. If *URL* is not specified, the Google Chart API will be used.

`-i`, `--incremental`
: Make list items in Slidy or S5 display incrementally (one by one). The default is for lists to be displayed all at once.

`--offline`
: Include all the CSS and javascript needed for a Slidy or S5 slide show in the output, so that the slide show will work even when no internet connection is available.

`--chapters`
: Treat top-level headers as chapters in LaTeX, ConTeXt, and DocBook output.

`-N`, `--number-sections`
: Number section headings in LaTeX, ConTeXt, or HTML output. By default, sections are not numbered.

`--listings`
: Use listings package for LaTeX code blocks

`--section-divs`
: Wrap sections in `<div>` tags (or `<section>` tags in HTML5), and attach identifiers to the enclosing `<div>` (or `<section>`) rather than the header itself. See Section identifiers, below.

`--no-wrap`
: Disable text wrapping in output. By default, text is wrapped appropriately for the output format.

`--columns=`*NUMBER*
: Specify length of lines in characters (for text wrapping).

`--ascii`
: Use only ascii characters in output. Currently supported only for HTML output (which uses numerical entities instead of UTF-8 when this option is selected).

`--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. If `--strict` is specified, *references* is used regardless of the presence of this option.

`--id-prefix=`*STRING*
: Specify 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*
: Specify classes to use for indented code blocks--for example, `perl,numberLines` or `haskell`. Multiple classes may be separated by spaces or commas.

`--toc, --table-of-contents`
: Include 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 on `man`, `docbook`, `slidy`, or `s5` output.

`--base-header-level=`*NUMBER*
: Specify the base level for headers (defaults to 1).

`--template=`*FILE*
: Use *FILE* as a custom template for the generated document. Implies `--standalone`. 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*
: Set 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` *URL*, `--css=`*URL*
: Link to a CSS style sheet.

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

`-A` *FILE*,
: `--include-after-body=`*FILE* 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 be used repeatedly to include multiple files. They will be included in the order specified. Implies `--standalone`.

`--reference-odt=`*FILE*
: Use 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 the user data directory (see `--data-dir`). If this is not found either, sensible defaults will be used.

`--epub-stylesheet=`*FILE*
: Use the specified CSS file to style the EPUB. If no stylesheet is specified, pandoc will look for a file `epub.css` in the user data directory (see `--data-dir`, below). If it is not found there, sensible defaults will be used.

`--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.

`--epub-metadata=`*FILE*
: Look in the specified XML file for metadata for the EPUB. The file should contain a series of Dublin Core elements, as documented at `http://dublincore.org/documents/dces/`. 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:language>` (from the locale), and `<dc:identifier id="BookId">` (a randomly generated UUID). Any of these may be overridden by elements in the metadata file.

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

`-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`.

`--bibliography=`*FILE*
: Specify bibliography database to be used in resolving citations. The database type will be determined from the extension of *FILE*, which may be `.mods` (MODS format), `.bib` (BibTeX format), `.bbx` (BibLaTeX format), `.ris` (RIS format), `.enl` (EndNote format), `.xml` (EndNote XML format), `.wos` (ISI format), `.medline` (MEDLINE format), `.copac` (Copac format), or `.json` (citeproc JSON). If you want to use multiple bibliographies, just use this option repeatedly.

`--csl=`*FILE*
: Specify CSL style to be used in formatting citations and the bibliography. If *FILE* is not found, pandoc will look for it in

```
$HOME/.csl
```

in unix and

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

in Windows. If the `--csl` option is not specified, pandoc will use a default style: either `default.csl` in the user data directory (see `--data-dir`), or, if that is not present, the Chicago author-date style.

`--natbib`
: Use natbib for citations in LaTeX output.

`--biblatex`
: Use biblatex for citations in LaTeX output.

`--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:

```
$HOME/.pandoc
```

in unix and

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

in Windows. A `reference.odt`, `epub.css`, `templates` directory, or `s5` directory placed in this directory will override pandoc's normal defaults.

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

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

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

is equivalent to

```
pandoc -o foo.html -s
```

`-v, --version`
: Print version.

`-h, --help`
: Show usage message.

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/FORMAT.template` in the user data directory (see `--data-dir`, above).

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

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

will be replaced by the document title.

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

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

`header-includes`	contents specified by `-H/--include-in-header` (may have multiple values)
`toc`	non-null value if `--toc/--table-of-contents` was specified
`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)
`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
`lang`	language code for HTML documents

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$, $endfor$
```

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.

PRODUCING HTML SLIDE SHOWS WITH PANDOC

You can use Pandoc to produce an HTML + javascript slide presentation that can be viewed via a web browser. There are two ways to do this, using S5 or Slidy.

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

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

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

To produce the slide show, simply type

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

for S5, or

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

for Slidy.

A title page is constructed automatically from the document's title block. Each level-one header and horizontal rule begins a new slide.

The file produced by pandoc with the `-s/--standalone` option embeds a link to javascripts and CSS files, which are assumed to be available at the relative path `ui/default` (for S5) or at the Slidy website at `w3.org` (for Slidy). If the `--offline` option is specified, the scripts and CSS will be included directly in the generated file, so that it may be used offline.

You can change the style of the slides by putting customized CSS files in `$DATADIR/s5/default` (for S5) or `$DATADIR/slidy` (for Slidy), 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.

Incremental lists

By default, these writers 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.

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 `literate-haskell` 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.

AUTHORS

Â© 2006-2011 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.)
Other contributors include Recai OktaÅ, Paulo Tanimoto, Peter Wang, Andrea Rossato, Eric Kow, infinity0x, Luke Plant, shreevatsa.public, Puneeth Chaganti, Paul Rivier, rodja.trappe, Bradley Kuhn, thsutton, Nathan Gass, Jonathan Daugherty, JÃ©rÃ©my Bobbio, Justin Bogner, qerub, Christopher Sawicki, Kelsey Hightower.

PANDOC'S MARKDOWN

For a complete description of pandoc's extensions to standard markdown, see `pandoc_markdown` (5).

SEE ALSO

`markdown2pdf` (1), `pandoc_markdown` (5).

The Pandoc source code and all documentation may be downloaded from <http://johnmacfarlane.net/pandoc/>.