Pandoc User's Guide

John MacFarlane

January 27, 2012

Pandoc's markdown

Pandoc understands an extended and slightly revised version of John Gruber's markdown syntax. This document explains the syntax, noting differences from standard markdown.

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.

Paragraphs

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

Headers

An header 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 header level:

```
## A level-two header

### A level-three header ###
```

As with setext-style headers, the header text can contain formatting:

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

Extension: `blank_before_header`

Standard markdown syntax does not require a blank line before a header. 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.
```

Header identifiers in HTML, LaTeX, and ConTeXt

Extension

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

```
- Remove all formatting, links, etc.
- Remove all punctuation, 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,

Header	Identifier
Header identifiers in HTML	`header-identifiers-in-html`
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 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, LaTeX, and ConTeXt formats.

If the `--section-divs` option is specified, then each section will be wrapped in a `div` (or a `section`, if `--html5` was specified), and the identifier will be attached to the enclosing <`div`> (or <`section`>) tag rather than the header itself. This allows entire sections to be manipulated using javascript or treated differently in CSS.

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 headers), with each line preceded by a > character and a space.

> This is a block quote. This paragraph does indeed have more than one line.
>
> 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.

Extension: `blank_line_before_blockquote`

Standard markdown syntax does not require a blank line before a block quote. Pandoc does require this (except, of course, at the beginning of the document). The reason for the requirement is that it is all too easy for a > to end up at the beginning of a line by accident (perhaps through line wrapping). So, unless `--strict` is used, the following does not produce a nested block quote in pandoc:

> This is a block quote. > Nested.

Verbatim (code) blocks

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

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

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

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

Delimited code blocks **Extension:** `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 (∼) or backticks (‘) and end with a row of tildes or backticks 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, delimited 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
~~~~~~~~~~
~~~~~~~~~~~~~~~~
```

Optionally, you may attach attributes to the code block using this syntax:

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

Here `mycode` is an identifier, `haskell` and `numberLines` are classes, and `startFrom` is an attribute with value `100`. Some output formats can use this information to do syntax highlighting. Currently, the only output formats that uses this information are HTML and LaTeX. 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, do `pandoc --version`.) Otherwise, the code block above will appear as follows:

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

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

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

This is equivalent to:

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

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 four-space rule

A list item may contain multiple paragraphs and other block-level content. However, subsequent paragraphs must be preceded by a blank line and indented four spaces or a tab. The list will look better if the first paragraph is aligned with the rest:

- First paragraph.

Continued.

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

  ```
  { code }
  ```

List items may include other lists. In this case the preceding blank line is optional. The nested list must be indented one tab:

- fruits

  – apples
    - ∗ macintosh
    - ∗ red delicious

    - pears
    - peaches

- vegetables
    - brocolli
    - chard

The markdown syntax guide is not explicit whether the four-space rule applies to *all* block-level content in a list item; it only mentions paragraphs and code blocks. But it implies that the rule applies to all block-level content (including nested lists), and pandoc interprets it that way.

Ordered lists

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

In standard markdown, enumerators are decimal numbers followed by a period and a space. The numbers themselves are ignored, so there is no difference between this list:

1. one
2. two
3. three

and this one:

5. one
6. two
7. three

Extension: `fancy_lists`

Unlike standard markdown, Pandoc allows ordered list items to be marked with uppercase and lowercase letters and roman numerals, in addition to arabic numerals. List markers may be enclosed in parentheses or followed by a single right-parentheses or period. They must be separated from the text that follows by at least one space, and, if the list marker is a capital letter with a period, by at least two spaces.[1]

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
3) Three

1. Four

- Five

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

1. one
2. two
3. three

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

```
B. Russell was an English philosopher.
```

do not get treated as list items.

This rule will not prevent

```
(C) 2007 Joe Smith
```

from being interpreted as a list item. In this case, a backslash escape can be used:

```
(C\) 2007 Joe Smith
```

Definition lists

Extension: `definition_lists`

Pandoc supports definition lists, using a syntax inspired by PHP Markdown Extra and reStructuredText:[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. The body of the definition (including the first line, aside from the colon or tilde) should be indented four 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
```

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:

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

```
(@)  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.

Compact and loose lists

Pandoc behaves differently from `Markdown.pl` on some “edge cases” involving lists. Consider this source:

```
+   First
+   Second:
    -   Fee
    -   Fie
    -   Foe

+   Third
```

Pandoc transforms this into a “compact list” (with no `<p>` tags around “First”, “Second”, or “Third”), while markdown puts `<p>` tags around “Second” and “Third” (but not “First”), because of the blank space around “Third”. Pandoc follows a simple rule: if the text is followed by a blank line, it is treated as a paragraph. Since “Second” is followed by a list, and not a blank line, it isn’t treated as a paragraph. The fact that the list is followed by a blank line is irrelevant.

Ending a list

What if you want to put an indented code block after a list?

```
-   item one
-   item two

    { my code block }
```

Trouble! Here pandoc (like other markdown implementations) will treat `{ my code block }` as the second paragraph of item two, and not as a code block.

To "cut off" the list after item two, you can insert some non-indented content, like an HTML comment, which won't produce visible output in any format:

```
-   item one
-   item two

<!-- end of list -->

    { my code block }
```

You can use the same trick if you want two consecutive lists instead of one big list:

```
1.  one
2.  two
3.  three

<!-- -->

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

Horizontal rules

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

Tables

Extension: `simple_tables`, `multiline_tables`, `grid_tables`, `pipe_tables`, `table_captions`

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

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 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:[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. A caption may optionally be provided (as illustrated in the example above). A caption is a paragraph beginning with the string `Table:` (or just `:`), which will be stripped off. It may appear either before or after the table.

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.

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

Multiline tables

Multiline tables allow headers 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 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.
-------------------------------------------------------------

: 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.

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.). Alignments are not supported, nor are cells that span multiple columns or rows. Grid tables can be created easily using Emacs table mode.

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 simple table syntax.
```

The syntax is the same as in PHP markdown extra. The beginning and ending pipe characters are optional, but pipes are required between all columns. The colons indicate column alignment as shown. The header can be omitted, but the horizontal line must still be included, as it defines column alignments.

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.

Title block

Extension: `pandoc_title_blocks`

If the file begins with a title block

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

it will be parsed as bibliographic information, not regular text. (It will be used, for example, in the title of standalone LaTeX or HTML output.) The block may contain just a title, a title and an author, or all three elements. If you want to include an author but no title, or a title and a date but no author, you need a blank line:

```
%
% Author

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

The title may occupy multiple lines, but continuation lines must begin with leading space, thus:

```
% My title
  on multiple lines
```

If a document has multiple authors, the authors may be put on separate lines with leading space, or separated by semicolons, or both. So, all of the following are equivalent:

```
% Author One
  Author Two

% Author One; Author Two

% Author One;
  Author Two
```

The date must fit on one line.

All three metadata fields may contain standard inline formatting (italics, links, footnotes, etc.).

Backslash escapes

Extension: `all_symbols_escapable`

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

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

one will get

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

instead of

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

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

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

A backslash-escaped space is parsed as a nonbreaking space. It will appear in TeX output as ~ and in HTML and XML as `\&#160;` or `\&nbsp;`.

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

Backslash escapes do not work in verbatim contexts.

Smart punctuation

Pandoc will produce typographically correct output, converting straight quotes to curly quotes, `---` to em-dashes, `--` to en-dashes, and `...` to ellipses. Non-breaking spaces are inserted after certain abbreviations, such as "Mr."

Note: if your LaTeX template uses the `csquotes` package, pandoc will detect automatically this and use `\enquote{...}` for quoted text.

Inline formatting

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\*.
```

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

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

Superscripts and subscripts 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∼`, not `P∼`.

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 delimited code blocks:

```
‘<$>‘{.haskell}
```

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. How it is rendered depends on the output format:

- HTML: with (mathjax)[http://www.mathjax.org/]
- Latex: with Latex

Raw HTML

Markdown allows you to insert raw HTML (or DocBook) anywhere in a document (except verbatim contexts, where <, >, and `&` are interpreted literally).

Standard markdown allows you to include HTML "blocks": blocks of HTML between balanced tags that are separated from the surrounding text with blank lines, and start and end at the left margin.

Within these blocks, everything is interpreted as HTML, not markdown; so (for example), `*` does not signify emphasis.

Pandoc interprets material between HTML block tags as markdown. Thus, for example, Pandoc will turn

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

into

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

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

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.

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

LaTeX macros

For output formats other than LaTeX, pandoc will parse LaTeX `\newcommand` and `\renewcommand` definitions and apply the resulting macros to all LaTeX math. So, for example, the following will work in all output formats, not just LaTeX:

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

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

In LaTeX output, the `\newcommand` definition will simply be passed unchanged to the output.

Links

Markdown allows links to be specified in several ways.

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

```
<http://google.com>
<sam@green.eggs.ham>
```

Pandoc will render autolinked URLs and email addresses as inline code.

Inline links An inline link consists of the link text in square brackets, followed by the URL in parentheses. (Optionally, the URL can be followed by a link title, in quotes.)

```
This is an [inline link](/url), and here’s [one with
a title](http://fsf.org "click here for a good time!").
```

There can be no space between the bracketed part and the parenthesized part. The link text can contain formatting (such as emphasis), but the title cannot.

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 can be space between the two.) The link definition must begin at the left margin or indented no more than three spaces. It 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.

Here are some examples:

```
[my label 1]: /foo/bar.html  "My title, optional"
[my label 2]: /foo
[my label 3]: http://fsf.org (The free software foundation)
[my label 4]: /bar#special  ’A title in single quotes’
```

The URL may optionally be surrounded by angle brackets:

```
[my label 5]: <http://foo.bar.baz>
```

The title may go on the next line:

```
[my label 3]: http://fsf.org
  "The free software foundation"
```

Note that link labels are not case sensitive. So, this will work:

```
Here is [my link][FOO]

[Foo]: /bar/baz
```

In an *implicit* reference link, the second pair of brackets is empty, or omitted entirely:

```
See [my website][], or [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 Header identifiers in HTML, LaTeX, and ConTeXt, below). For example:

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

or

```
See the [Introduction].

[Introduction]: #introduction
```

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

Images

A link immediately preceded by a `!` will be treated as an image. The link text will be used as the image's alt text:

```
![la lune](lalune.jpg "Voyage to the moon")

![movie reel]

[movie reel]: movie.gif
```

Pictures with captions

Extension

An image occurring by itself in a paragraph will be rendered as a figure with a caption.[4] (In LaTeX, a figure environment will be used; in HTML, the image will be placed in a `div` with class `figure`, together with a caption in a `p` with class `caption`.) The image's alt text will be used as the caption.

[4]This feature is not yet implemented for RTF, OpenDocument, or ODT. In those formats, you'll just get an image in a paragraph by itself, with no caption.

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

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

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

Footnotes

Extension: `footnotes`

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

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

[^1]: Here is the footnote.

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

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

        { some.code }

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

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

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

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

Extension: `inline_notes`

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

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

Inline and regular footnotes may be mixed freely.

Citations

Extension: `citations`

Pandoc can automatically generate citations and a bibliography in a number of styles (using Andrea Rossato's `hs-citeproc`). In order to use this feature, you will need a bibliographic database in one of the following formats:

Format	File extension
MODS	.mods
BibTeX/BibLaTeX	.bib
RIS	.ris
EndNote	.enl
EndNote XML	.xml
ISI	.wos
MEDLINE	.medline
Copac	.copac
JSON citeproc	.json

You will need to specify the bibliography file using the `--bibliography` command-line option (which may be repeated if you have several bibliographies).

By default, pandoc will use a Chicago author-date format for citations and references. To use another style, you will need to use the `--csl` option to specify a [CSL] 1.0 style file. A primer on creating and modifying CSL styles can be found at `http://citationstyles.org/downloads/primer.html`. A repository of CSL styles can be found at `https://github.com/citation-style-language/styles`. See also `http://zotero.org/styles` for easy browsing.

Citations go inside square brackets and are separated by semicolons. Each citation must have a key, composed of '@' + the citation identifier from the database, and may optionally have a prefix, a locator, and a suffix. Here are some examples:

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

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

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

A minus sign (`-`) before the `@` will suppress mention of the author in the citation. This can be useful when the author is already mentioned in the text:

```
Smith says blah [-@smith04].
```

You can also write an in-text citation, as follows:

```
@smith04 says blah.

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

If the style calls for a list of works cited, it will be placed at the end of the document. Normally, you will want to end your document with an appropriate header:

```
last paragraph...

# References
```

The bibliography will be inserted after this header.

Producing slide shows with Pandoc

You can use Pandoc to produce an HTML + javascript slide presentation that can be viewed via a web browser. There are four ways to do this, using S5, DZSlides, Slidy, or Slideous. You can also produce a PDF slide show using LaTeX beamer.

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 the slide show, simply type

```
pandoc -t s5 -s habits.txt -o habits.html
```

for S5,

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

for Slidy,

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

for Slideous,

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

for DZSlides, or

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

for beamer.

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.

Structuring the slide show

By default, the *slide level* is the highest header level in the hierarchy that is followed immediately by content, and not another header, somewhere in the document. In the example above, level 1 headers are always followed by level 2 headers, which are followed by content, so 2 is the slide level. 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 header at the slide level always starts a new slide.
- Headers *below* the slide level in the hierarchy create headers *within* a slide.
- Headers *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.
- A title page is constructed automatically from the document's title block, if present. (In the case of beamer, this can be disabled by commenting out some lines in the default template.)

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

For Slidy, Slideous and S5, 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 `s5/default` (for S5) or `slideous` (for Slideous), or at the Slidy website at `w3.org` (for Slidy). (These paths can be changed by setting the `slidy-url`, `slideous-url` or `s5-url` variables; see `--variable`, above.) For DZSlides, the (relatively short) javascript and css are included in the file by default.

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.

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.

To style beamer slides, you can specify a beamer "theme" or "colortheme" using the `-V` option:

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

Literate Haskell support

If you append `+lhs` to an appropriate input or output format (`markdown`, `rst`, or `latex` for input or output; `beamer`, `html` or `html5` 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 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, 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.

Authors

© 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 Oktaş, 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, Jérémy Bobbio, Justin Bogner, qerub, Christopher Sawicki, Kelsey Hightower, Masayoshi Takahashi, Antoine Latter, Ralf Stephan, Eric Seidel, B. Scott Michel, Gavin Beatty, Sergey Astanin.