

# Emacs pandoc-mode

Joost Kremers

21 Sep 2013

## 1 Introduction

pandoc-mode is an Emacs mode for interacting with [Pandoc](#). Pandoc is a program (plus libraries) created by John MacFarlane that can convert a text written in one markup language into another markup language. pandoc-mode is implemented as a minor mode that can be activated alongside the major mode for any of Pandoc's supported input formats. It provides facilities to set the various options that Pandoc accepts and to run Pandoc on the input file. It is possible to create different output profiles for a single input file, so that you can, for example, write your text in Markdown and then translate it to HTML for online reading, PDF for offline reading and Texinfo for reading in Emacs, all with just a few keystrokes.

The current version of pandoc-mode is 2.5 and is compatible with Pandoc version 1.12.

## 2 Installation

The easiest way to install pandoc-mode is to use the [Melpa](#) package repository. Alternatively, you can install it manually: download `pandoc-mode.el`, put it into Emacs' loadpath (you can byte-compile it if you wish) and add the following line to `~/.emacs`:

```
(load "pandoc-mode")
```

Either installation method allows Emacs to load pandoc-mode, but they do not activate it. In order to activate pandoc-mode in a buffer, you need to type `M-x pandoc-mode`. To start pandoc-mode automatically when you load e.g., a Markdown file, you can add a hook to your `~/.emacs`:

```
(add-hook 'markdown-mode-hook 'turn-on-pandoc)
```

However, if you do not want to start pandoc-mode every time you work on a Markdown document, you can use a different function in `markdown-mode-hook`: instead of using `turn-on-pandoc`, you can use `conditionally-turn-on-pandoc`. This function checks if a default settings file exists for the file you're opening and only turns on pandoc-mode if it finds one. (For more info on the settings file, see the section '[Settings Files](#)'.)

Additionally, if you want to automatically load the default pandoc-mode settings file for the file you're opening, you can add the following to your `~/.emacs`:

```
(add-hook 'pandoc-mode-hook 'pandoc-load-default-settings)
```

The function `pandoc-load-default-settings` checks if a default settings file exists for the file being loaded and reads its settings if it finds one.

### 3 Usage

When you start `pandoc-mode`, a menu appears through which all of Pandoc’s options can be set and through which you can run Pandoc on your current document. The most important functions provided by `pandoc-mode` can also be accessed through the keyboard, but by default, most options can only be set through the menu. If you wish, however, it is of course possible to bind your own keys to the functions to set options. See the section [‘Using The Keyboard’](#)).

The menu is divided into two parts. The upper half contains items related to running Pandoc, the lower half is the part where the various settings can be set and changed.

#### 3.1 Input and output formats

The most important settings are the input and output formats. The input format is set automatically by Emacs on the basis of the major mode of the input file, but you can change it through the menu if you need to. The output format defaults to “Native Haskell”, so most likely you will want to set it to something else before you run Pandoc. Note that the output format can also be set through the keyboard with `C-c / w` (TAB completion works.) If the input or output format is one of the Markdown formats (Pandoc supports different Markdown dialects), the relevant submenu contains a further submenu with the Markdown extensions that Pandoc supports. This menu allows you to select or deselect individual extensions.

As already stated, you may wish to use different output formats for a single input file. Most likely, the options that you want to pass to Pandoc will be different for each output format. To make this easier, `pandoc-mode` has the ability to save the settings for a specific output format. If you choose “Save File Settings” from the menu (`C-c / s`), Emacs saves the current settings to a hidden file in the same directory as the file you’re editing. The name of this file is derived from the input file, appended with the name of the output format and the string `.pandoc`. (See the section [‘Settings Files’](#) for details.)

A single document can have a settings file for each output format that Pandoc supports. For example, for this manual, which is written in Markdown, I have three settings files, one for HTML output, one for LaTeX output and one for Texinfo output. These can simply be created by setting all options the way you want them for the first output format, save them, then choose another output format, set the required options, save again, etc. Because the name of a settings file contains the output format for which it was created, the different settings files won’t interfere with each other.

On systems that have symbolic links, it is also possible to specify a default output format. By selecting “Set As Default Format” from the Pandoc menu, a symbolic link is created to the settings file of the current output format (a settings file is created if one doesn’t exist yet). This symbolic link has “default” as format in its name. The file it points to is read by the function `pandoc-load-default-settings`, making it possible to automatically load a specific settings file when `pandoc-mode` is invoked, as described above.

When you switch output formats, either through the menu or with the keyboard (with `C-c / w`), Emacs checks if a corresponding settings file exists and loads it if one is found. That is, you can load a different settings file by simply switching output formats.

Note that the current output format is always visible in the mode line: the “lighter” for `pandoc-mode` in the mode line has the form `Pandoc/<format>`, where `<format>` is the current output format.

The major modes for which `pandoc-mode` selects an input format automatically can be customized. You can add major modes or remove those that you don’t use. Similarly, you can customize the output formats and (especially) the file extensions for each output format.

## 3.2 Auxiliary files and options

The settings part of the menu contains three more submenus: “Files”, “Options”, and “Switches”. Under “Files”, you can set various files that may be used by Pandoc, such as a template file, the reference ODT file, the CSS style sheet, files to be included in the header or before/after the body, etc. Most of these submenus have two options: no file, or specify a file. The “Files” submenu also contains options for the output file and output directory. These are treated somewhat differently from the other files, see the section ‘[Setting an output file](#)’ for details.

When Emacs calls `pandoc`, it expands filenames, so that they are absolute and don’t contain any abbreviations (such as `~` for one’s home directory). This means you can have relative filenames in your settings, or indeed `~`, which can be practical if you move settings files to different locations or e.g. between computers with different OSes. (For example, Linux expands `~` to `/home/<user>`, while on OS X it becomes `/Users/<user>`.)

The CSS style sheet is an exception to this: Emacs always cuts off the directory part of the filename you specify as CSS style sheet and doesn’t expand it. The reason for this is that the CSS style sheet will normally be transferred along with the output file(s) to a server, where it will most likely be in a different directory than on the computer you’re generating your HTML files on.

Under “Options”, you can set other options that require a value. The submenu “Switches” contains Pandoc’s binary options, i.e., those that only can be present or absent and do not accept a specific value.

Furthermore, the menu contains an option to get a list of all the settings that you have defined (this function is also available by typing `C-c / S`). This displays all settings in the `*Pandoc output*` buffer in a Lisp-like format. For example, the settings for LaTeX output of this manual look like this:

```
((number-sections . t)
 (standalone . t)
 (read . "markdown")
 (write . "latex")
 (output . "pandoc-mode-manual.tex")
 (output-dir . "~/src/pandoc-mode/website/")
 (variable
  ("mainfont" . "Linux Libertine O")
  ("sansfont" . "Linux Biolinum O")
  ("geometry" . "a4paper"))
 (latex-engine . "xelatex"))
```

### 3.3 Template variables and metadata

pandoc-mode allows you to set or change template variables through the menu (under the Options submenu) or the keyboard, with `C-c / v` and `C-c / m`, respectively. Emacs will ask you for the name of a variable or metadata item and for a value for it. If you provide a name that already exists (TAB completion works), the new value replaces the old one.

Deleting a template variable or metadata item is done with the corresponding menu item or by calling `C-c / v` or `C-c / m` with the negative prefix argument `C-u -` (or `M--`). Emacs will ask you for the variable name (TAB completion works here, too) and removes it from the list.

### 3.4 Running Pandoc

The first item in the menu is “Run Pandoc” (also accessible with `C-c / r`), which, as the name suggests, runs Pandoc on the document, passing all options you have set. By default, Pandoc sends the output to stdout, which is redirected to the buffer `*Pandoc output*`. (Except when the output format is “odt”, “epub” or “docx”, in which case output is always sent to a file.) The output buffer is not normally shown, but you can make it visible through the menu or by typing `C-c / V`. Error messages from Pandoc are also displayed in this buffer.

Note that when you run Pandoc, Pandoc doesn’t read the file on disk. Rather, Emacs feeds it the contents of the buffer through `stdin`. This means that you don’t actually have to save your file before running Pandoc. Whatever is in your buffer, saved or not, is passed to Pandoc.

If you call this command with a prefix argument (`C-u C-c / r`), Emacs asks you for an output format to use. If there is a settings file for the format you specify, the settings in it will be passed to Pandoc instead of the settings in the current buffer. If there is no settings file, Pandoc will be called with just the output format and no other options.

Note that specifying an output format this way does not change the output format or any of the settings in the buffer, it just changes the output profile used for calling Pandoc. This can be useful if you use different output formats but don’t want to keep switching between profiles when creating the different output files.

### 3.5 Setting an output file

If you want to save the output in a file rather than have it appear in a buffer, you can set the output file through the menu. Note that setting an output *file* is not the same thing as setting an output *format* (though normally the output file has a suffix that indicates the format of the file). The “Output File” submenu has three options: the default is to send output to stdout, in which case it is redirected to the buffer `*Pandoc output*`. Alternatively, you can let Emacs create an output filename for you. In this case the output file will have the same base name as the input file but with the proper suffix for the output format. Lastly, you can also specify an output file yourself.

Note that Pandoc does not allow output to be sent to stdout if the output format is an OpenOffice.org Document (ODT), EPUB or MS Word (docx) file. Therefore, Emacs will always create an output filename in those cases, unless of course you’ve explicitly set an output file yourself.

The output file you set is always just the base filename, it does not specify a directory. Which directory the output file is written to depends on the setting “Output Directory” (which is not actually a Pandoc

option). Emacs creates an output destination out of the settings for the output directory and output file. If you don't specify any output directory, the output file will be written to the same directory that the input file is in.

### 3.6 Creating a pdf

The second item in the menu is “Create PDF” (C-c / p). This option calls Pandoc with an output file with the extension .pdf, causing Pandoc to create a pdf file by first converting to .tex and then calling LaTeX on it.

If you choose this option, Emacs checks if your current output format is latex. If it is, Emacs calls Pandoc with the buffer's settings. If the output format is something other than latex, Emacs checks if you have a settings file for LaTeX output and uses those settings. This allows you to create a pdf without having to switch the output format to LaTeX. If your current output format is not LaTeX and no LaTeX settings file is found, Emacs calls Pandoc with only the input and output formats.

## 4 Settings Files

Apart from settings files for individual files (which are called *local settings files*), pandoc-mode supports two other types of settings files: project files and global files. Project files are settings files that apply to all input files in a given directory (except those files for which a local settings file exists). Global settings files, as the name implies, apply globally, to files for which no local or project file is found. Both types of files are specific to a particular output format, just like local settings files. Project files live in the directory they apply to and are called Project.<format>.pandoc. Global files live in the directory specified by the variable pandoc-data-dir, which defaults to ~/.emacs.d/pandoc/, but this can of course be changed in the customisation group pandoc.

Whenever pandoc-mode loads settings for an input file, it first checks if there is a local settings file. If none is found, it looks for a project file, and if that isn't found, it tries to load a global settings file. In this way, local settings override project settings and project settings override global settings. Note, however, that if a local settings file exists, *all* settings are read from this file. Any project file or global file for the relevant output format is ignored.

You can create a project or global settings file through the menu in the submenu “Settings Files”. This simply saves all settings for the current buffer to a project or global settings file. (Any local settings file for the file in the current buffer will be kept. You'll need to delete it manually if you no longer need it.)

## 5 Managing numbered examples

Pandoc provides a method for creating examples that are numbered sequentially throughout the document (see [Numbered example lists](#) in the Pandoc documentation). pandoc-mode makes it easier to manage such lists. First, by going to “Example Lists | Insert New Example” (C-c / c), you can insert a new example list item with a numeric label: the first example you insert will be numbered (@1), the second (@2), and so on. Before inserting the first example item, Emacs will search the document for any existing definitions and number the new items sequentially, so that the numeric label will always be unique.

Pandoc allows you to refer to such labeled example items in the text by writing `(@1)` and `pandoc-mode` provides a facility to make this easier. If you select the menu item “Example Lists | Select And Insert Example Label” (`C-c / C` : note the capital C) Emacs displays a list of all the `(@)`-definitions in your document. You can select one with the up or down keys (you can also use `j` and `k` or `n` and `p`) and then hit `return` to insert the label into your document. If you change your mind, you can leave the selection buffer with `q` without inserting anything into your document.

## 6 Using @@-directives

`pandoc-mode` includes a facility to make specific, automatic changes to the text before sending it to Pandoc. This is done with so-called @@-directives (“double-at directives”), which trigger an Emacs function and are then replaced with the output of that function. A @@-directive takes the form `@@directive`, where `directive` can be any user-defined string. Before Pandoc is called, Emacs searches the text for these directives and replaces them with the output of the functions they call.

So suppose you define (e.g., in `~/ .emacs`) a function `pandoc-current-date`:

```
(defun pandoc-current-date (output-format)
  (format-time-string "%d %b %Y"))
```

Now you can define a directive `@@date` that calls this function. The effect is that every time you write `@@date` in your document, it is replaced with the current date.

Note that the function that the directive calls must have one argument, which is used to pass the output format to the function (as a string). This way you can have your directives do different things depending on the output format.

@@-directives can also take the form `@@directive{...}`. Here, the text between curly braces is an argument, which is passed to the function that the directive calls as its second argument. Note that there should be *no* space between the directive and the left brace. If there is, Emacs won’t see the argument and will treat it as normal text.

It is possible to define a directive that can take an optional argument. This is simply done by defining the argument that the directive’s function takes as optional. Suppose you define `pandoc-current-date` as follows:

```
(defun pandoc-current-date (output-format &optional text)
  (format "%s%s" (if text (concat text ", ") "")
    (format-time-string "%d %b %Y")))
```

This way, you could write `@@date` to get just the date, and `@@date{Cologne}` to get “Cologne, 21 Sep 2013”.

Two directives have been predefined: `@@lisp` and `@@include`. Both of these take an argument. `@@lisp` can be used to include Emacs code in the document which is then executed and replaced by the result (which should be a string). For example, another way to put the current date in your document, without defining a special function for it, is to write the following:

```
@@lisp{(format-time-string "%d %b %Y")}
```

Emacs takes the Elisp code between the curly braces, executes it, and replaces the directive with the result of the code.

`@@include` can be used to include another file into the current document (which must of course have the same input format):

```
@@include{copyright.text}
```

This directive reads the file `copyright.text` and replaces the `@@include` directive with its contents.

Processing `@@`-directives works everywhere in the document, including in code and code blocks, and also in the `%-header` block. So by putting the above `@@lisp` directive in the third line of the `%-header` block, the meta data for your documents will always show the date on which the file was created by Pandoc.

If it should ever happen that you need to write a literal “`@@lisp`” in your document, you can simply put a backslash `\` before the first `@`: `\@@lisp`. Emacs removes the backslash (which is necessary in case the string `\@@lisp` is contained in a code block) and then continues searching for the next directive.

The directives are processed in the order in which they appear in the customization buffer (and hence in the variable `pandoc-directives`). So in the default case, `@@include` directives are processed before `@@lisp` directives, which means that any `@@lisp` directive in a file included by `@@include` gets processed, but if a `@@lisp` directive produces an `@@include`, it does *not* get processed. (If this should ever be a problem, you can always create a directive `@@include2` and have it processed after `@@lisp`.)

After Emacs has processed a directive and inserted the text it produced in the buffer, processing of directives is resumed from the *start* of the inserted text. That means that if an `@@include` directive produces another `@@include` directive, the newly inserted `@@include` directive gets processed as well.

## 6.1 Defining `@@`-directives

Defining `@@`-directives yourself is done in two steps. First, you need to define the function that the directive will call. This function must take at least one argument to pass the output format and may take at most one additional argument. It should return a string, which is inserted into the buffer. The second step is to go to the customization buffer with `M-x customize-group RET pandoc RET`. One of the options there is `pandoc-directives`. This variable contains a list of directives and the functions that they are linked with. You can add a directive by providing a name (without `@@`) and the function to call. Note that directive names may only consist of letters (a-z, A-Z) or numbers (0-9). Other characters are not allowed. Directive names are case sensitive, so `@@Date` is not the same as `@@date`.

Passing more than one argument to an `@@`-directive is not supported. However, if you really want to, you could use `split-string` to split the argument of the `@@`-directive and “fake” multiple arguments that way.

A final note: the function that processes the `@@`-directives is called `pandoc-process-directives` and can be called interactively. This may be useful if a directive is not producing the output that you expect. By running `pandoc-process-directives` interactively, you can see what exactly your directives produce before the resulting text is sent to pandoc. The changes can of course be undone with `M-x undo` (usually bound to `C-/`), or do your test in the `*scratch*` buffer.

## 6.2 Directive hooks

There is another customizable variable related to @@-directives: `pandoc-directives-hook`. This is a list of functions that are executed *before* the directives are processed. These functions are not supposed to change anything in the buffer, they are intended for setting up things that the directive functions might need.

## 7 Using The Keyboard

Although `pandoc-mode` can be controlled through the menu, it is possible to bind all functions to keyboard sequences. `pandoc-mode` uses the prefix `C-c /`, if you bind any functions to keys, it would be best to use this prefix as well. In order to make your life a little easier if you do decide to bind certain option setting functions to key sequences, all of these have been set up so that they can be used with the prefix key `C-u -` (or `M--`) to unset the relevant option, and with any other prefix key to set the default value (if the option has one, of course).

The option setting functions are all called `pandoc-set-<option>` (with the exception of the option `--read`, i.e., the input format, which is determined automatically and rarely needs to be set by the user). `<option>` corresponds to the long name of the relevant Pandoc switch. Functions can be bound in the following manner:

```
(define-key 'pandoc-mode-map "\C-c/o" 'pandoc-set-output)
```

The following table lists the keys defined by default and the functions they are bound to:

<code>C-c/r</code>	Run pandoc on the document ( <code>pandoc-run-pandoc</code> )
<code>C-c/p</code>	Run <code>markdown2pdf</code> on the document ( <code>pandoc-convert-to-pdf</code> )
<code>C-c/s</code>	Save the settings file ( <code>pandoc-save-settings-file</code> )
<code>C-c/w</code>	Set the output format ( <code>pandoc-set-write</code> )
<code>C-c/m</code>	Set a metadata item ( <code>pandoc-set-metadata</code> )
<code>C-c/v</code>	Set a template variable ( <code>pandoc-set-variable</code> )
<code>C-c/V</code>	View the output buffer ( <code>pandoc-view-output</code> )
<code>C-c/S</code>	View the current settings ( <code>pandoc-view-settings</code> )
<code>C-c/c</code>	Insert a new <code>(@)</code> -item ( <code>pandoc-insert-@</code> )
<code>C-c/C</code>	Select and insert a <code>(@)</code> -label ( <code>pandoc-select-@</code> )