James Lemin

## Introduction

Using vi to write longer documents is ideal, but needs a little additional thought up front. [grok-ing](https://stackoverflow.com/questions/1218390/what-is-your-most-productive-shortcut-with-vim) vi not only allows you to move around efficiently, but also effectively edit and restructure text.

*Fluently speaking vi* allows you leverage the huge amount of power in vanilla vi. Instead of installing hundreds of plugins, minimal configuration can provide the same functionality without sacrificing speed or compatibility. For instance a reasonable analogue goyo might be achieved by removing the ruler and line numbers. Does this look as pretty as the goyo results? No, but these two changes can be expressed in two lines, achieve a similar goal of reducing visual cruft, and don’t require a plugin.

That’s not to say that plugins are bad, they’re a personal choice. For instance the combination of vim-pandoc and vim-pandoc-syntax are hard to beat. They stay out of the way until needed, and provide real functionality that’s hard to beat. Natively handling and completing citations (type @cha, then <c-x> <c-p> to get a list of matching citations) is a straightforward way of getting citations in your document. Arguably VSCode does this better with tab completion, but it does so at the cost of visual cruft and additional loading time. The same goes for vim-pandoc’s :Pandoc command. This can be used to render a document to a variety of formats, including pdf, docx, html, and odt from within vi, combining it with the build, silent, and ! commands allows you to build a document with a single keystroke, or even on save.

if empty(glob('./build'))  
 let g:pandoc#command#autoexec\_command = "Pandoc ! pdf"  
else  
 let g:pandoc#command#autoexec\_command = ":silent !./build silent"  
 endif

Another plugin recommendation is vim-lexical this is another plugin that stays out of the way, seamlessly correcting commonly misspelt words such as ‘teh’.

## Why Use a Text Editor Over Word

Word is a great tool. Whether you want to write a letter, do a mail merge, create some kind of word art monstrosity or anything between, word has a wide range of functionality. The majority of this functionality is in your face however. Whether it’s the spelling and grammar checking, or the extensive formatting options, they’re all visible, distracting you from *writing*. In contrast, with vi, you are explicitly either writing or editing text. The visual distractions are minimal, and can be reduced even further through pretty minimal configuration. The editing in vi feels more precise too; cutting and pasting a sentence can be done with (d) then p rather than pfaffing around with a mouse, or Ctrl+Shift key combinations.

## Using Markdown

Using a lightweight markup language such as markdown means formatting is a trivial case of adding the relevant mars such as \_\_ for italics, or ## for a second level heading. There’s a learning curve, but it’s shallow, and once learnt it’s applicable widely, for example in Reddit posts, adding READMES, or maintaining a personal Zettelkasten. Markdown helps further abstract *writing* and *editing* and *formatting*. Formatting is important, but it’s typically something you’d want to consider *after you’ve written* rather than during the writing process. A second benefit to markdown is that it’s convenient to read, edit, and store. There’s no complex file type here, no binary format that can corrupt, it’s purely a text file with a .md extension.

Although Markdown is straightforward to use, some thing such as tables can be slightly more complex to manage. The below table looks like this:

| Header | Header |  
|---------|---------|  
| Content | Content |

in the source file. Pipes and hyphens are used to build the table. There are ways around this, such as plugins, or using scripting tables, but none as powerful as Excel. On the other hand, tables in markdown are *explicit*; there is no wrestling with what Word *thinks* the table should do. as a bonus, using : your text here syntax, it’s a doddle to caption the tables. Using pandoc, this syntax filters through to base LaTeX syntax, meaning it’ll appear if you use the lot tag in YAML.

here’s a figure for the table

| Header | Header |
| --- | --- |
| Content | Content |

Code blocks are something difficult to achieve natively in Word. In basic markdown stylesheets, they look OK, but applying standard filters in Pandoc, they look great; syntax highlighting is good, and the resulting code blocks look, well, like *code* for instance a for loop in bash:

for i in {0..9}  
 do echo $i  
done

The same loop in Python

for i in range(0,9):  
 print($i)

Not a for loop, not a programming language, but still presented correctly…

select sysdate from dual

### Handling Lists in Markdown

Vanilla markdown simplifies list handling. If you’ve ever wrestled with Word second guessing what you are trying to do with lists, and getting it horribly wrong (*Continue numbering* and orphaned styles are personal bugbears) it’s a breath of fresh air to…

+ represent  
+ bullet-points  
 + like  
 + this

which renders as you’d expect:

* represent
* bullet-points
* like
* this

Numbered lists are more complex, but follow the same understandable  
rules around indentation with the bonus of not needing to remember what  
number you are on:

1. numbered  
1. lists  
1. for the win

1. numbered
2. lists
3. for the win

The options are many; too many to cover here, though it’s worth mentioning to pandoc’s @ syntax which looks like this

to   
pandoc's `@` syntax which looks like this  
  
``` text  
@) numbered  
@) lists  
  
split by text  
  
@) work with special syntax

@) numbered @) lists

split by text

@) work with Pandoc’s @ syntax

Again, this works seamlessly ‘out of the box’ though there’s a  
possibility to confuse this syntax with the pandoc-cite syntax for instance:

* how about a test of vim-pandoc’s hard, automatic wrapping option  
  (hA) this should kick in at column 79
* fingers crossed it does..!
* success..!

*some useful tests / syntax…*

* *italic*
* **bold**
* [small caps]{.smallcaps}
* <http://simple_url_render.com>
* [more complex url render](http://example.com)
* [email](mailto:email@address.com)
* reference link [example](https://james-lemin.com)
* Superscript
* Subscript
* math formulae
* ~~strike through~~

DefinitionTerm

Definition



Hello World image

### A Level-three Heading with a [link](/url) and *emphasis*

Veniam veldt okay cat quis culpa ex cupid tat Lorem nulla qui. Officia  
qua nisi esse et sint velit incididunt. Dolore non dolore quis officia.

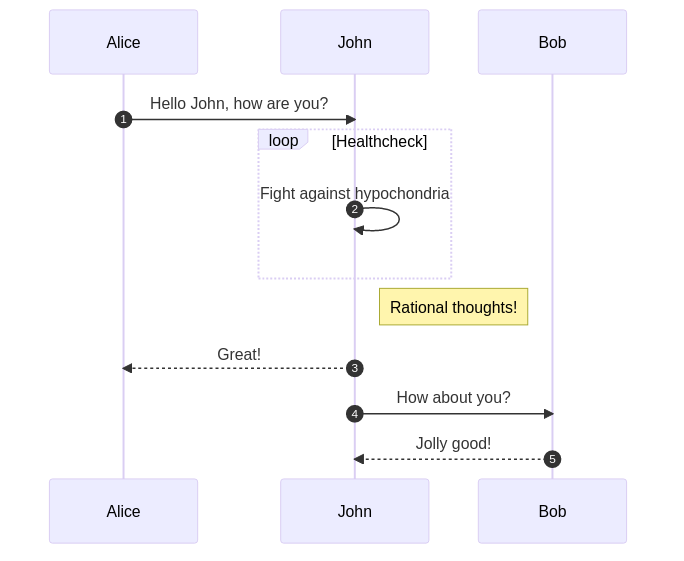
## Using Mermaid Diagrams

Mermaid diagrams can be used to create reasonable-looking diagrams. To do  
this, use the mermaid-cli to pre-process the markdown file, and  
then use pandoc to convert the markdown to the desired format. The command is:

mmdc -i input.md -o output.md

Thre’s a few things to note here:

1. the proprocessor will replace the mermaid code with a link to the diagram image \_in the .tmp directory
2. the preprocessor will add a generic caption text to the image
3. image sizing isn’t considered; use pandoc to resize the image
4. the preprocessor will output svg images by default. These seem to render incorrectly, so we’ll use pngs instead



## Conclusion

Adipisicing magna ad proident Lorem aliqua quis Ali quip Epsom sent. Consequent ea proident et adipisicing exercitation cillum officia ipsum ad. Minim in commodo parader dolour Loren qua auto temper. Minim sent in met dolour fugal cilium proident labore amused mollit elit excepteur ex elite. Dews ease excerpter labium nostrud veniam incident anim aliqua proident ea Ali quip inure ad. Arte consecrator ullamco amet non. Est eiusmod occaecat culpa pariatur eu dolore. Botton (2005). See [here](foo) for more info

## References

<!- References should always go at the end; the filename should make that happen –>

Botton, A. de (2005) *On Seeing and Noticing*. London: Penguin Books Ltd.