A Writer's Guide to Vim (v1.2)

Prologue

I've used many word-processing programs, starting back with WordPerfect in the 1990s. From there I drifted into AbiWord and OpenOffice. I stuck with them after I made the jump from Microsoft to Linux in the mid-2000s, but had already started using text editors to write. I liked the cleanness of the interface and the fact that files were saved in .txt format—something that's both portable and (for the most part) cross platform. I used the gedit text editor that came with the Gnome version of Debian, and was happy with the results. I'd occasionally try opening vim, but I always ended up frustrated, banging on the keyboard in a futile attempt to exit the program.

As I became more experienced with Linux, I got more comfortable working on the command line and bought a book on shell-scripting, which contained a chapter on vim and emacs. I tried some vim commands while holding the book in my lap and this time, was able to actually save a file and exit the program without frustration. In fact, I rather liked the experience. I made the switch from gedit to vim and haven't looked back.

Many of the online discussions of vim's features center—naturally enough—around programmers' and coding. But programmers use features that a regular prose writer generally doesn't, so I began documenting vim tips and tricks I found useful. A few weeks ago, as I learned yet another neat thing, I realized it might be good to write up my collection to share with other writers. This is the result.

Vim Basics

IF YOU AREN'T ALREADY FAMILIAR WITH VIM, I HIGHLY RECOMMEND YOU STOP HERE AND TAKE THE TIME TO LEARN THE BASICS USING THE "VIMTUTOR" PROGRAM. Many who write with vim got their start here. Simply type "vimtutor" at the command line and follow the instructions.

If you haven't tried vimtutor and nevertheless have opened vim for the first time and are lost, you can get out of vim by hitting the ESC key a couple of times, then typing ":q!". That dumps you out of the program without saving anything. Strictly speaking, you only need to hit the ESC key once to drop you from insert mode (which is what you use to type) to normal mode (which is what you use to edit/move around the document); however, if you've already tried typing a few commands, two or three stiff taps on ESC will ensure that you're back in normal mode and can exit the program.

If you *do* want to save something before exiting, instead of ":q" type ":w [filename]". This saves the on-screen text into a file. (Personally, I recommend saving your text often. No one is more annoyed than a writer who's just lost an hour's work because they forgot to save it. The voice of sad experience….)

If you've already saved to a filename and want to save your latest changes then exit, type ":wq"

You can also open a new file when you open vim. Just type "vim [filename]". If the file exists, vim will open it; if it doesn't, it'll be created and any subsequent ":w" commands you give will save to that file.

Once you've started your vim journey, you'll have many questions. The best way to get an answer is to use the vim help documents. They're huge—everything I mention in this guide is less than 1% of what the help documents contain. You access them by typing ":h(elp) [what you're looking for]". For example, ":h buffers" to learn about vim's use of buffers.

You can also search for key commands. Not sure what "ZZ does?" Type ":h ZZ" and find out. And there's even a :help help to tell you how the help pages work.

Vim Modes

Unlike some other text editors, vim has modes. Some people love them, others (especially people new to vim) hate them, but they provide a lot of power and flexibility to the program.

The modes are:

Normal mode: This is used to move around and/or modify text. Want to delete a word or a whole line or a paragraph? Use normal mode. Want to jump down ten lines and then five words over from the cursor's current position? Yep. Normal mode.

You get into normal mode by hitting your ESC key.

NOTE: The ESC key is bit of a pain to use, so many long-term vim users remap the ESC key to something else so that they don't have to constantly reach all the way to the corner of the keyboard. There's a bit of debate among vim users which key(s) should be used in place of ESC. Personally, I use a quick series of taps on the "jk" keys, though many people also use "ii" (since typing "i" is what gets you into insert mode). HOW to remap your ESC key will be shown later, in the .vimrc section.

Insert mode: As a writer, you'll spend a lot of time in this mode. It’s used to type new words, lines and paragraphs. While in insert mode, you can navigate around the screen using your arrow keys, a line (or letter) at a time or with PgUp, PgDn, Home and End. But moving around the text is much quicker in normal mode. It seems counter-intuitive to change modes to move just a few lines or words through the text but trust me, once you get the hang of it, you'll rarely notice.

You get into insert mode by typing "i" (for insert at the current cursor position), "a" (for append—insert starting at the next position to the right of the cursor), "o" (for append to the line below) or “O” (for append to the line above).

When in insert mode, you'll see "-- INSERT --" in the lower left corner of your screen.

Visual Mode: This mode should be familiar to most word processor and text-editor users. In visual mode, text selected (by the mouse or using the movement keys) gets highlighted on the screen. Any editing command then given (delete, copy, reformat) acts upon the entire block of highlighted text.

You get into (and out of) visual mode by typing "v" from normal mode.

Command mode: Command mode accesses some of the more cryptic-yet-powerful features in vim. Command mode allows you to see cut-and-paste buffers, the help menu, and to search/replace throughout your text.

You get command mode by typing a colon ":" then a command. For example, ":w" and ":q" are part of command mode.

Vim Motions

Now we come to the real secret power of vim: moving around the text. If you've done the vimtutor program, you know that "h" moves you one character back; "j" moves you one line down; "k" moves you one line up; and "l" moves you one character over. The reason for these keys is historical: on old computer keyboards, there were no separate arrow keys—the arrows were marked on four of the regular keys. (You can guess which ones they were....) To further the historical lesson, the ESC key once sat where the caps lock key now does, which is why it was chosen to move you out of insert and into normal mode.

Using one-line-at-a-time (or one-character-at-a-time) motions are tiresome. So here are some other useful vim movement commands—all are used in normal mode.

Type "0" to move to the beginning of a line
Type "^" to move to the first non-blank character of a line
Type "$" to move to the end of a line
Type "w" to move forward one word
Type "b" to move back one word
Type "e" to move to the end of a word, or end-of-word to end-of-word
Type "(" or ")" to move back/forward one sentence
Type "{" or "}" to move back/forward one paragraph

Vim considers everything within a white-spaced paragraph to be a single "line," no matter how long it is. The "g" commands allow you to move around the text as appears on the screen.

Type "g0" to move to the beginning of the line the cursor is currently on
Type "g$" to move to the end of the line the cursor is currently on
Type "gj" to move down one line on the screen
Type "gk" to move up one line on the screen
Type "gi" to go to the last place where you inserted text and return to insert mode.

Type Ctl+f to move forward one screen
Type Ctl+b to move back one screen
H, M, L moves you to the top/middle/bottom of the current screen. (Think High, Middle, Low.)

You can also move through a text by searching for a word or phrase:

Type "t" and then a letter to take you to the position just *before* that letter in current line
Type "f" and then a letter to take you to the next occurrence of that letter in the current line
Type “*” to search through the document for the word the cursor is currently under
Type "/" then a search term. Hit enter and you'll be taken to the first match.
Type "n" to take you to the next match or Shift+n to the previous match.
Type ":noh" (no highlighting) to remove all the highlights from a set of search results and reset your search

A nice benefit of searching/highlighting is that it can show you if you’ve overused a particular word—just put your cursor under the word and type “*”. The word will be highlighted throughout your entire text. You can also use search to look for adverbs by typing “/ly”.

Editing Actions

As with the movement commands, all editing commands are called from normal mode:

Type "dw" to delete an entire word
Type “dd” to delete an entire line
Type "x" to delete the character under the cursor
Type "p" to paste any yanked (copied) text starting at the current cursor position.
Type "r" and then a letter to replace the character under the cursor with the new letter
Type "cw" to change a word. This deletes the word under the cursor and puts you into insert mode
Type "u" to undo the change you just made. Vim holds a number of previous changes in memory, so you can under a number of changes simply by repeatedly typing "u".
Type "J" to join the line below your current line to the current line.

Where vim begins to shine is when you combine editing actions with motions. Do a quick online search and read the post “*Your problem with Vim is that you don't grok Vi”* on the Stack Overflow website. It's a lot to digest, especially for new vim users, but the powerful idea behind it is that vim is a language with verbs and nouns, and that movements can be your nouns. So, to delete an entire paragraph, type "{" to get yourself to the beginning of the

paragraph, then "d}" to delete everything down to the next paragraph. Want to delete to the end of a line? Type "d$". Delete the next sentence? Type “d)” Vim editing even has a couple of "adjectives:" the letters "i" and "a". If you're in a sentence and want to delete it, type "dis" (delete in sentence). You can also delete words/sentences/paragraphs with "a" in your command. For example, "dap" or "dip" deletes a paragraph. Sometimes there’s more than one way to make an action happen: "{d}" versus "dip" for example. You'll eventually settle into your own favorite key combinations. However, it doesn't hurt to learn other ways of achieving the same effect, as it may turn out to be more efficient than the one you were using before.

Word Count and Spell Checking

To set up spelling in vim, turn it on by typing ":set spell" (You can also set up spelling for different regions/languages. Type ":help spell" to see the full help file on spelling.) Once spell is set, misspelled and improperly capitalized words will be highlighted in your text. You can correct the misspelling either in normal mode or while still in insert mode:

While in normal mode, put the cursor inside the misspelled word and type "z=" to get a list of alternates.

While in insert mode, put your cursor inside the misspelled word and type "Ctl+x s" to pop up a small list of alternate words. "Ctl+n" takes you down in the list; "Ctl+p" takes you up. Hitting the space bar selects the word while keeping you in insert mode, so you can just keep typing.

You can add and remove words to the list of correctly spelled words (good for adding your character names or words in your alien language). Simply put your cursor under the word and type "zg". To remove the word from the list, put the cursor under the word and type "zug".

Counting words is just as easy as spell checking. Put your cursor at the end of the document by typing "G" followed by “$” to make sure you’re at the end of the last line. Then type "g Ctl+g". This will give you the location of the cursor, including the number of words before the cursor's current position. You can count words in a chunk of text is by putting your cursor wherever you want (say where you started writing that morning) and typing "g Ctl+g". That will give you not only the number of words before the cursor, but how many words are in the overall document. Just subtract the two to get your number of added words. Another way is to use visual mode ("v") to highlight a block of text before running "g Ctl+g". This will give you the number of words in the highlighted selection.

Auto-complete/Thesaurus/Grammar Checking/Abbreviations

Good news and bad news. First, the good news: vim has auto-completion. In insert mode, start typing a word, then hit "Ctl+n" to get a list of possible completions. You navigate up and down the list using the same keys as for spell-check: "Ctl+n" and "Ctl+p". The bad news: that's the only way auto-completion

works in vim without plugins. (But see below about abbreviations.) As for the plugins themselves, I don’t use them and so can’t comment.

More good and bad news: You *can* install a thesaurus on vim. Bad news: it's incredibly involved and cumbersome. I don't use a thesaurus; I follow John McPhee's advice in his essay "*Draft No. 4*" that it's better to use a dictionary than a thesaurus. I *did* try setting up a thesaurus just as an experiment. It required either downloading a file from Project Gutenberg, modifying it, then using a special function in your .vimrc configuration file or it required a plugin. Too much work for me.

Vim doesn't have native grammar checking. That also requires a plugin.

I *do* use abbreviations a lot. Vim lets you create a file with a list of abbreviations and their matching full text, which then automatically expands to the full word when you type in the shorter version. For example, abbreviate the name of your character "Evil Count Rattail" as "ECR," then when you type "ECR" and hit the space bar, "Evil Count Rattail" appears in your document. Useful for making sure character/location/cool tech (or magical) item names—especially long ones—remain consistent. Read the help page for abbreviations (":h abbreviations"), but in a nutshell, create a text document with all your abbreviations, one abbreviation per line, in this format:

```
:ia Xan Xanthocrystometaboline
:ia ECR Evil Count Rattail
:ia VE Verdant Empire
```

Then save the file in your .vim directory. Activate your personal abbreviation list by typing ":source ~/.vim/[filename.txt]". It's also possible to activate the list automatically whenever vim starts up through your .vimrc file. (Which will be discussed soon.)

While automatic sourcing of an abbreviation file is nice, depending on how many projects you have going simultaneously, it might actually work better to have one abbreviation file per project, and activate them manually. This would allow you to re-use abbreviations in different projects, so that "ECR" could also stand for "Everyday Common Reader" or "Elephantine Commercial Railway."

Abbreviations *can* also provide a sort of on-the-fly auto-completion and auto-correction. If you create abbreviations for common misspellings you’ll end up with the correct word even if your fingers type the letters out of order.

```
:ia hte the
:ia htat that
:ia ot to
:ia fo of
```

Digraphs

If you use foreign words in your writing, you may need digraphs. Digraphs are easy to make in vim: simply type "Ctl-k" and then the two parts of the character. For example, "ñ" comes from pressing from "Ctl-k n ~". If you want to see a complete chart of digraphs, type ":dig!". If you pull up the chart, you'll notice some of the digraphs don't have two-letter combinations, only a number. Those can't be entered using "Ctl-k." However, they *can* be entered by typing "Ctl-v" and then entering the numeric code. You can now sprinkle your writing with foreign words, Greek, Hebrew, or whatever you can find on the chart. But please don't name one of the aliens in your Science Fiction novel something like "ÞädЖ‡∩²" just because you can....

Snippets

Snippets are bits of text that can be inserted into your document, perfect for things you'd normally type over and over again. I have several snippets set up: one to create a standard manuscript header with name, contact info and word count; one to open a groff template if I decide I want to use that program to create a .pdf out of a document; and a snippet to add a signature block at the end of my emails. To make a snippet, you first have to create a file that contains the text you want to insert. Then when you call the snippet, vim will copy the text verbatim into the current document at the current cursor location.

Snippet files can be stored anywhere, but I keep mine in my .vim directory.

You can insert a snippet manually by typing ":r [filename including path]". But if your snippet is something you use often, it makes sense to assign that action to an easier set of keystrokes. In my case, I used ",header" ",template" and ",sig". For example, the following line sets up my signature block for emails:

nnoremap ,sig :-1read $HOME/.vim/sig.txt<CR>

Though it looks like gibberish, it really means "Map the following keys--,sig--to read the contents of the text file located at $HOME/.vim/sig.txt, at the current position, then add a blank line at the end." These key mappings (and re-mappings) are the sort of thing you put in your .vimrc file so you don't have to type them out each and every time.

Files/Buffers/Windows/Tabs

Like many text editors, vim can open multiple files at once, and you can switch between them as needed. Vim's default method is to use "buffers," loading a file into memory and only displaying it when specifically asked to. Windows allow you to see two (or more) buffers on the same screen at the same time, while tabs let you "page" back and forth between sets of windows.

I tend toward using tabs and lately, buffers. But I don't often have more then a couple of files open at any one time—a main document, say, and perhaps an outline for reference.

First, buffers. While in normal mode, simply type ":e [filename]" or “:find [filename]”. Your new file will open, replacing the file you were looking at a moment ago. However, the previous screen/file isn't gone, just hidden. (You can also open multiple files/buffers by typing "vim [file1] [file2] [file3]".)

Type ":bn" to go to the next open buffer, ":bN" to go back a buffer. (If you only have two files open at one time, ":bn" or "Ctl-^" (or on US keyboards, even "Ctl-6") cycles you back and forth.) You can also type ":bf" to take you to the first open buffer or jump to a buffer by typing “:b [any unique substring of the filename]”. For example, jump to “notes.txt” by typing “:b not” and hitting Enter.

If you've got multiple files open, you can type ":ls" to list the numbers and contents of all the current buffers. You can then go to the buffer you want by typing ":b [Number]". But don't be surprised if the numbers identifying the buffers are far apart—vim keeps track of a number that aren’t normally displayed to users.

Where things can get thorny is in closing buffers. If you're not careful, you can accidentally dump yourself entirely out of vim and back to the command line —with no changes saved in *any* document.

The easiest way to close the current buffer is ":bd". If you've made changes to that buffer, you'll be warned there are unsaved changes and that adding "!" to the end of the ":bd" command will exit without saving.

Now, here's a quirky thing: if you have multiple buffers open, and all the buffers have been saved, typing ":q" (as you would if you were leaving a single document), you'll exit vim. If you've got autosave turned on, your changes *should* have been saved first. But in any case, getting dumped out of vim can be annoying, so try to remember to use ":bd" when closing buffers. (If you want to save all changes in *all* buffers and then exit, type ":xa".)

Windows split your computer screen into two (or any other number of) parts. From normal mode, type ":sp". That splits the screen in half horizontally (and loads the file you're currently working on into it). To open a window and load one of the buffers you *already* have open, type ":sp #[buffer number]". The pound sign is very important--the buffer won't load without it.

To open a vertical (side-by-side instead of top-and-bottom) split, type ":vs". You load another buffer into the vertical split the exact same way as with a horizontal one: ":vs #[buffer number]."

To navigate between windows, type "Ctl+w w" or "Ctl+w Ctl+w."

Typing ":q" will close you out of whichever window you are currently in.

Tabs are collections of any number of windows that can be "flipped through" just like buffers. To create a new tab, type ":tabe" or ":tabnew". To open a tab and load a file into it, type ":tabe [filename]". The command ":tabn" or "gt" will take you to the next tab. To close a tab, type ":tabc".

Do You Need Plugins?

One of the ways vim can be expanded is by using plugins. They are sets of pre-packaged vim commands. You let vim know where the plugin resides and activate it in your .vimrc file. Many programmers have plugins to highlight typing errors in their computer code based on which programming language they're currently using. Other common vim plugins deal with tasks such as auto-completion, file management or ensuring that a program's source-code file will compile.

There *are* a few prose writer-specific plugins, though, such as "vim-pencil" and "goyo," as well as several grammar checkers. I don't use any of them, but feel free to look them up online see if they'd be helpful to you in your writing. A few years back, I stumbled across a video on YouTube called "How to Do 90% of What Plugins Do (With Just Vim)." I highly recommend watching it before diving into plugins, because you may discover that you can get by without them.

And as the presenter of that YouTube video said, I'm not knocking people who decide to use plugins. In my case, it's simply that I don't really feel I need them to do prose writing. Nor am I switching back and forth from writing code (in which line numbering and color-coding of various functions are important) to writing prose (in which they aren't). But if you find a vim plugin that makes your writing easier and more enjoyable, by all means download and install it.

Your Personal .vimrc file

In many of the above examples, I've said something along the lines of "type x to activate function y." The problem is, that activation only lasts until you close vim. However, vim *does* have a configuration file which allows you to make sure those functions are active every time you start the program. It's called a ".vimrc" file.

To create one, open a text editor (I recommend vim) and start typing the functions you want, one per line. When you're done, save the file in your home directory as ".vimrc" (that's "period-vimrc"). From then on, all your tweaks and modifications will be there without any further work on your part.

There's a long and noble history of vim users putting their .vimrc files online so others can cut-and-paste from them. Feel free to borrow the best bits from other users.

Here's my own personal .vimrc, with comments added (any bit of text with a quotation mark turns into a comment that's ignored):

```
"""""""
"Overall Settings
"""""""
set syntax=off                  "I find syntax highlighting annoying
highlight SpellBad ctermbg=red      "highlights misspelled words in red
highlight Pmenu ctermbg=blue        "sets the background of the spellcheck menu to blue
set nocompatible                  "keeps the vim program from behaving like its predecessor "vi"
set linebreak                   "keeps typed text from going all the way to screen edge
set autowriteall                  "makes sure everything is saved before I exit
set viminfo=""                    "nothing is saved in the viminfo file
```

```
set mouse=a                    "set mouse support to on – though I rarely use a mouse in vim
source ~/.vim/abbrev.txt       "where I keep my abbreviations
set spell                      "activate spell-checking


"""""""
"Searching
"""""""
set ignorecase                 "ignore upper/lowercase when searching
set smartcase                  "set smart case in searching
set incsearch                  "set incremental search that highlights as search progresses
set hlsearch                   "highlight all search matches



"""""""
"Tabs
"""""""
set tabstop=5                  "set tab to five spaces
set softtabstop=5              "set backspace tab to five spaces
set expandtab                  "set tab expansion (five spaces)


"""""""
"File Handling/Searching
"""""""
set path+=**                   "allow searching of subfolders
set wildmenu                   "set menu autocomplete


"""""""
"Snippets -- Paste text from pre-set files
"""""""
nnoremap ,header :-1read $HOME/.vim/storyheader.txt<CR>     "puts a manuscript header
into a document
nnoremap ,template :-1read $HOME/.vim/template.txt<CR>      "pastes groff formatting
needed to make a .pdf
nnoremap ,sig :-1read $HOME/.vim/sig.txt<CR>                "inserts my signature block for
emails

"""""""
"Remap jk/kj to <Escape> with no cursor jump (`^).
"""""""
inoremap jk <Esc>`^
inoremap kj <Esc>`^
```

Document Preparation

So now that you're done writing your *Magnum Opus,* how do you get your well-vimmed manuscript ready for an editor or for printing? I'm sorry to say you'll need to use a program other than vim to do that. Some people take the text document and run it through a set of macros called pandoc to convert it into other

formats. I haven't looked at pandoc for some time, but seem to remember that if you write your text document in markdown, pandoc will handle bolding and italicizing words for you. But if you're like me and use your own idiosyncratic way of marking text, pandoc might not work for you.

Others just cut and paste the entire text into an empty Microsoft Word or LibreOffice document and then go through and edit it to add their bold and italics. Since many magazines want submissions as .doc or .docx files, this is the technique I currently use. It's not the fastest, but it works well enough.

Epilogue

Hopefully, you've found some useful tips and tricks. I intend to make this a living document that changes and grows, so please feel free to email me at contact@andrewgudgel.com with comments and/or suggestions for future versions.

Andrew Gudgel
December 2022