The Woodnotes Guide to Vim for Writers

Randall Wood (www.therandymon.com)

July 18, 2009

Contents

1 Why Vim for Writers? 2

2 Gvim vs. Vim 3

3 Line Wrapping and Text Width 3

4 Files (Opening, Saving, etc.) 4

5 General Editing of Text 4

6 Getting Around with the Cursor 5

7 Scrolling 7

8 Bookmarks (“Marks”) 7

9 Selecting Text, Cutting and Pasting 8

10 Searching and Replacing 9

11 Using Ranges 9

12 Multiple Windows, Buffers, and Tabs 10

13 Inserting Special Characters 10

14 Dealing with DOS, Unix conversion problems 11

15 Spell Checking **11**

16 Macros **12**

17 Learning more about Vim **12**

18 Acknowledgments, License, and Version History **13**

List of Figures

1 Files (Opening, Saving, etc.) 4
2 General Editing Commands 5
3 Basic Movement 6
4 Advanced Movement 6
5 Simple Search Commands 7
6 Scrolling Relative to Document 7
7 Scrolling Relative to Screen 7
8 Visual Mode Selection Commands 8
9 Cutting and Pasting 9
10 Ranges 10
11 Some Common Digraphs 11

1 Why Vim for Writers?

Vim is well used and well liked by computer programmers and computer scientists, but many of its features make it a productive tool for writers and authors. Namely, you can navigate and work efficiently using succinct, keystroke commands, it is highly customizable, makes efficient use of screen real estate, is fast and extremely efficient. Furthermore, many of us have taken the time to get to know Vim for other purposes, and are so impressed we'd like to make use of it for our writing projects.

However, this is not a manual on how to use or learn Vim. Many of these exist, and all of them are better than what I could produce. Rather, this manual is written for the writer who wants to use Vim as his primary text editor for the purpose of producing prose, novels, fiction, a thesis, a treatise, or similar. If you already know how to use Vim, this Woodnotes manual will help you use Vim efficiently as a writer or author. It is a

companion volume to the Woodnotes Guide to Emacs for Writers[1], and covers the same information.

Your first task then, if you haven't already done so, is to get to know Vim. I recommend the use of a good reference card to learn and remember the commands. There are many available; my personal favorite is the creation of Laurent Grégoire.[2] This guide uses the same notation as Laurent's reference card, so it would help to have it in front of you as you continue.

The graphical vi-vim cheat sheet and tutorial[3] is an excellent way to learn Vim and serves as a cheat sheet afterwards (they also have a Dvorak version!). And of course, Vim's own tutorial (just type `:vimtutor` is effective, and should probably be the first place you begin learning this powerful software.

2 Gvim vs. Vim

Vim has traditionally been a console application, but there is a graphical version called Gvim as well[4], which some prefer. With the exception of spell checking and the ability to choose among several color schemes, I'm not aware of any great difference between the two. You can also use console Vim in a terminal window on your graphical desktop, which, in my opinion is the best of all worlds, since you can take advantage of the better font rendering and resolution while still making maximum use of your screen space (run the terminal window in full screen mode for the best results).

3 Line Wrapping and Text Width

Unless you choose otherwise, each line of text will extend off the screen with no wrapping. For authors, that's not convenient. So issue the command `:set linebreak` to wrap the text when it hits the screen edge. This is what other software calls "soft wrapping" in that it conveniently manages the text on screen but doesn't insert newlines or carriage returns in the file. For long text works, this is probably what you want.

If however, you want for example, 80 character hardwrapped text (text formatted with a new line after 80 characters) like in the good old days, issue `:set textwidth=80`. If you are writing, for example, a LaTeX document in which your document is a source file, this latter method may be more useful over all. Using the former method, what Vim considers to be a "line" is actually what we would call a paragraph, and moving up by one line at a time (using `j` and `k`) actually moves the cursor up by one paragraph at a time. Using the latter method, moving up one line actually moves you up one line on the screen, which is convenient. Otherwise, use `gj` and `gk` to advance the cursor one screen line at a time.

[1] `http://therandymon.com/content/view/16/98/`
[2] `http://tnerual.eriogerg.free.fr`
[3] `http://www.viemu.com/a\_vi\_vim\_graphical\_cheat\_sheet\_tutorial.html`
[4] "G" because it is built with the GTK+ toolkit"

4 Files (Opening, Saving, etc.)

Use `:e` to begin editing a file and `:w` to write it to the disk. Watch out if you change the name of the file when saving: this is not "save as" but rather the command to save a file by the other name while you continue editing the first. That is to say, if you are editing file “Introduction”" and issue a `:w Temp` a file called “Temp” will be saved to the disk which for an instant will be identical to “Introduction.” But as soon as you continue typing, your new edits will continue to be applied to the file “Introduction.” That makes this a good way to save temporary (working) files as you go. Use `:saveas` to save the working file under a new name and continue editing that new file.

One neat Vim trick of perhaps limited use to authors is the ability to specify a list of files to edit when you call the program. If, from the command line, you enter `vim ch1 ch2 ch3` Vim will allow you to edit each of the three files in succession. In that case, `:wn` will write the file, close it,and then open the next (`:wN` opens the previous.). You can use `:rw f` to write the range r to file f (for example, `:*w dialogue` will take the entire visually selected range and write it out to a file called “dialogue.” Working in the other direction, `:r dialogue` will take the contents of the file “dialogue” and insert it at the cursor.

Finally, an easy way to go through the current file and select bits and pieces of it for use elsewhere is to write out, appending to the other file. For example, once you’ve selected something, `":w>>goodies` will append that selected text to the file “goodies.” You can then open up the file “goodies” to clean it up.

Command	Key
Edit (Open)	:e
Write (Save)	:w
Save as	:saveas
Append to file	:w>>

Figure 1: Files (Opening, Saving, etc.)

5 General Editing of Text

Here, too, an adequate mastery of a few common commands makes basic text entry a lot more efficient and allows you to make edits on the fly without stopping to take your hands off the keyboard. They all require the control key, because these commands are expected to be entered while you are still in insert mode.

The most important is `control-w`, which deletes the most recent word and permits you to keep typing. Similarly, `control-r x` inserts the contents of register x at the cursor.[5]

In command mode, use the following to speed up editing and in particular, to take advantage of commands that allow you to, with one keystroke, position the cursor, erase

Function	Key
Insert before, after cursor	i, a
Insert at beginning, end of line	I, A
Change text of motion m	c m
Change to end of line	C
Delete text of motion m	d m
Delete to end of line	D
Delete letter under, before cursor	x,X
Change letter under cursor and keep typing	s
Replace letter under cursor	r
Open a new line below, above cursor	o,O
Join this line to the next	j

Figure 2: General Editing Commands

text, and begin editing.

A few examples will make the benefit clear. As an author I frequently begin a sentence and then decide halfway through that I am not pleased with what I have begun to write. A simple `c(` instructs Vim to change (erase and prepare for replacement of) everything to the beginning of that sentence. I can just as easily type `c7b` to change the last 7 words. If I've edited something mid-paragraph and am ready to now place the cursor at the end of the paragraph and keep writing, it's as simple as typing a capital `A`. It's also quick to select some text using visual mode, press `d` to delete it, and continue.

Note that where delete and insert commands are concerned, pressing the `.`key (the period) instructs Vim to repeat the last command. If you type `dw` (delete word) and then `3.` the second command will delete 3 more words. Be careful!

One last word about editing. Vim remembers up to 100 of your last commands[6] This allows you a lot of flexibility with the undo command. Press `u` in command mode to undo your last command, and keep pressing it until you are satisfied. `Control-r` works in the opposite direction, redoing what you've just undone.

6 Getting Around with the Cursor

6.1 Basic Movement

Vim will most impact your proficiency by facilitating rapid movement of the cursor and navigation through your document without taking your hands from the keyboard. Your mastery and application of movement and scrolling commands are thus essential.

[5]The fact that the text is inserted *at the cursor* is important. If you insert text in command mode using the `put` command (`p` or `P`) it is sometimes inserted on a new line, and you must then join the two lines with `j`.

[6]This is configurable and can be much, much higher if you wish.

Direction	Key
Left, Right	h, l
Down, Up	j, k
Screen Line Down, Up	gj, gk
Forward, Backward	Key
word	w,b
sentence	(,)
paragraph	{,}
Beginning, End of Doc	gg, G

Figure 3: Basic Movement

Note that if you are using soft wrapping, what Vim considers to be a "line" is actually what we would call a paragraph since to Vim, a line ends with a "new line" character. As such, moving up by one line at a time (using `j` and `k`) actually moves the cursor up by one paragraph at a time. In that case, use `gj` and `gk` to advance the cursor one screen line at a time.

You can gain some time with prepending a numerical reference, so typing `3(` will take you back 3 sentences, and `5}` will advance the cursor 5 paragraphs.

Vim is aware of what you have visible on the screen, and allows you to quickly jump around on that basis.

Position on Screen	Key
Top Line	H
Middle Line	M
Bottom Line	L
line n from top, bottom of window	nH, nL

Figure 4: Advanced Movement

6.2 Navigation by Search

A very convenient way to get around is by searching for a word located at the position you'd like to go to. Vim offers several mechanisms for searching, which gives you some flexibility in your approach.

The `f` and `t` commands are most useful for finding characters that appear infrequently in your document, like punctuation marks; as such, I don't use them frequently. But searching using the `/` and `?` commands is a fast way to get around your document. You can type `? Dingleberry` to be taken to the most recent use of that word in your document. Continue searching in that direction by pressing `n` (in this case, since we're searching upwards, `n` will continue searching upwards) or `N` to search in the opposite direction (in this case, down). Use a prepended numeral to jump that many search results: `? fleabag` and then `3n` will search for the word fleabag and take you to the 4th occurrence.

Search Mechanism	Key
search forward (“down”)	/
search backward (“up”)	?
next, previous occurrence	n,N
next, previous occurrence of character c	fc, Fc
just before next occurrence of character c	tc, Tc
next, previous occurrence of word under the cursor	#,*

Figure 5: Simple Search Commands

7 Scrolling

Vim will save you time by eliminating the need to take your hands off the keyboard in order to position the cursor by sliding those little scroll bars up and down with the mouse. Scrolling via the keyboard is immensely efficient.[7] At a minimum, get used to the `zz` command, which will position the cursor and the line you’re currently working on, in the center of the screen. Just don’t confuse it with `ZZ` which is a shortcut for “exit.”

Scroll Direction and Amount	Key
Line up, down	control-E, control-Y
Half page up, down	control-u, control-d
Full page up, down	control-b, control-f

Figure 6: Scrolling Relative to Document

Position Cursor on Screen	Key
At the top	zt
In the middle	zz
At the bottom	zb

Figure 7: Scrolling Relative to Screen

8 Bookmarks (“Marks”)

Placing marks through your text is an interesting way of navigating, if you find yourself navigating back to certain positions frequently. But it’s more important a tool as part of cutting and pasting. Type `m` plus a letter from a-z or A-Z to establish a mark, and then `'` (that’s the single quote, not the backtick) plus that letter to return the cursor to the position identified by that mark. By distinguishing between upper and lower case identifiers, Vim provides you 52 individual marks, more than you’ll probably need. If you

[7] Remember “forward/backward” and “up/down”.

forget where they are, typing `:marks` at the Ex prompt will provide you a list of them, plus the first couple of words at each mark to help you identify them.

9 Selecting Text, Cutting and Pasting

9.1 Visual Mode

There are easy ways and hard ways to select text, but they are both useful and worth getting to know. The easiest way is called Visual Mode. Trigger it by pressing `v` or `V`. A lower case v starts highlighting text one letter at a time; this is good when you want to edit part of a sentence. A capital `V` starts highlighting text one line at a time. When you've selected the text, hit `d` to delete it (or "cut" it, since it winds up in a buffer), or `y` to yank ("copy") it. Then navigate to where you want and hit `p` to place ("paste") it. Normally, the cursor remains at the top of what you have just pasted, which gives you a chance to scroll down through the next text. But if you want to paste and go, `gp` and `gP` paste and then reposition the cursor at the bottom of the next text.

You can also use t and f to advance the cursor until a point just before or right on the letter you are searching for. I frequently start visual mode with `v` and then advance up to the first comma and delete it. That looks like this: `vf,d`. That is, "start visual mode, advance to the comma, delete." Otherwise, use the `a` key plus `w, s, p` to add a word, sentence, or paragraph. But I just as often use the regular motion commands, and they work fine.

Command	Key
Start visually selecting by character, line	v, V
Exchange cursor position with start of highlighting	o
Start highlighting on previous visual area	gv
Add word, sentence, paragraph	aw, as, ap

Figure 8: Visual Mode Selection Commands

The more complicated way is to use the motion commands alone to select areas. For example, if you want to copy the entire sentence you just wrote, you could issue `y(` (that is, "yank to the beginning of the sentence"). You'll get no visual feedback for this operation, but the text will be yanked and ready to place elsewhere. As an author, though, this is a fast way to keep your hands on the keyboard. If you don't like what you just wrote, a simple `d(` will remove your last sentence, `d5b` will remove the last 5 words, `dTZ` will delete back until the last occurrence of a capital Z, and so on.

9.2 Using Named Registers

By using the multiple named registers, you can hang on to whatever you copy and yank. Select one using the quote (") as follows: `"ad{` means "delete into named register 'a' all

the text back to the beginning of the paragraph." If you forget what you've stored in the registers, the Ex command `:registers` will show them all.

Using both marks and registers in a single command means you mustn't mix up your ' and " characters: the former is for marks and the latter for registers. So, `"ay'b` means "Yank from the cursor's position to mark 'b', and store it in register 'a.'

Finally, you can place the contents of any register with the put command (`p`). For example, `"hp` will put the contents of register 'h' under the cursor, beginning on a new line (a capital `P` will place it on a new line above the cursor).

Command	*Key*
Cut/Delete	d
Yank/Copy	y
Put/Paste below, above cursor	p,P

Figure 9: Cutting and Pasting

10 Searching and Replacing

Let's start with the most common one as an example. Suppose you've written a story and now you want to change one of the character's names from Larry to Moe. It's as simple as `%s/Larry/Moe/g` which says "For the entire document, change all occurrences of Larry to Moe." The percent sign is the range identifier that indicates the search should take place over the entire document. You can't use motion commands here, but you can use line numbers (which is not that useful for authors), so :0,.s/hoo/ha/g will replace hoo with ha from line 0 to the present line. You can also append `c` for "confirm each replacement."

11 Using Ranges

In the Larry/Moe example above, the `%` tells the search command to search the entire document. But searching and replacing is one of the few times a writer might be interested in limiting the effect of a command to a certain range of lines rather than to the entire document. Use visual mode to select a certain part of your document, and then limiting the search and replace to that range with `*` instead of `%`. Figure 10 shows the other options at your disposal.

Here are two examples:

`.,$s/German/French/g` "From the cursor's current position to the end of the document, replace German by French."

`*s/Zune/iPod/gc` "Within the visual area, change all occurrences of Zune with iPod, and confirm each replacement."

Range	Key
Use , or ; to separate two line positions	, ;
Current line	.
Last line in file	$
Entire file	%
Visual area	*
Mark t	't

Figure 10: Ranges

12 Multiple Windows, Buffers, and Tabs

You can split the current screen into two parts where both both buffers show the same document, just by issuing `:split` (`:sp`). The screen will be divided into an upper and lower half; this is useful for looking at multiple parts of a document simultaneously. It's just as easy though to divide the window into left and right halves, with :vsplit (`:vspl`). Change between views with `control-w w` and close every other view but the current one by issuing `control-w o`. Note that if you ever use the help system, the window splits, and you will need to 'quit' the help system by issuing `:q` to get back to your document; of course you can leave the help window open, issue `control-w w` to go to the other view, and return to editing your document.

13 Inserting Special Characters

If you write in a language other than English, you will need to enter characters not necessarily on your keyboard: accented characters and letters in other scripts. Even if you use English, you might find it occasionally necessary to enter special characters. Frankly, the easiest way to do this is by means of your desktop environment (Gnome, KDE on Linux).[8] Vim is able to communicate with the desktop processes and receive whatever characters you send it. However, Vim has another mechanism of its own for producing special characters. In Vim they are called "digraphs," two keystroke combinations that lead to the production of a single character, like 'a for á and DG for the degree symbol (35°C).

When, while editing, you need a special character, press `control-k` and then the two digit code for that letter. This incomplete table shows some of the digraphs you would use for text with diacritical marks common to European languages. Type `:help digraphs` for a complete list.

As an example, if you want a c cedilla (ç), the comma key plus one other character is the way to produce it, so as you're typing in insert mode, type `control-k` then `,c`. If you want a tilde n (ñ), type `control-k` then `?n`.

[8] On KDE, open the control panel and under Regional/Accessibility – Keyboard layout, enable "Enable keyboard layouts." On Gnome, either add the "Special Characters" applet to a panel or enable the keyboard switcher.

Character Name	Char	Meaning
Exclamation mark	!	Grave
Apostrophe	’	Acute accent
Greater-Than sign	>	Circumflex accent
Question mark	?	Tilde
Hyphen-Minus	-	Macron
Left parenthesis	(	Breve
Full stop	.	Dot above
Colon	:	Diaeresis
Comma	,	Cedilla
Underline	_	Underline

Figure 11: Some Common Digraphs

14 Dealing with DOS, Unix conversion problems

You will occasionally have to deal with conversion of a DOS file which appears in Vim with a ^M character at the end of each line. The simplest way to do so is to simply search for and replace them, as follows: `%s/control-m//g`. That is, to search for a ^M you search for control-M. You can just as easily search for carriage returns (\r) like this: `%s/\r//g`. If you’re using Vim on a Windows machine, the opposite may occur, and you’ll find ^J characters at the line ends and beginnings of a Unix file. Treat it the same way, searching and replacing.

Both phenomena are caused by Vim failing to identify a text file as one format or another (usually because both types of line endings are present). The `fileformat` command is how you declare your intention; the next time you save the file the appropriate line endings will be used, and when you reopen it the strange characters should disappear. For example `:set ff=unix`.

15 Spell Checking

Vim took a big step forward with version 7, which introduced spell checking and thus made Vim a much more attractive system for writing long text works. To take advantage of the new functionality, begin by typing `:set spell`, which turns on the spell checker with the default language.

Once the spell checker is running, the keystrokes `[s` and `]s` move the cursor from error to error. With the cursor positioned over a misspelled word, the keystroke `zg` declares that this word is ‘good’ and should henceforth be ignored; `zw` identifies a good word as one that should’ve been identified as an error. Finally, `z=` will request a list of suggestions from the ispell program, which allows you to choose what you want.

For Vim to be able to add these words to a custom dictionary, you must define your “spellfile,” and Vim’s useful approach to doing so is to permit you to define and use as

many spellfiles as you'd like. As such, you can have one spelling list for acronyms you need for one type of file without 'contaminating' another dictionary. For example, `:set spellfile=.spellfile.sailing.add` creates a custom wordlist where you can put all your arcane sailing acronyms, and then choose not to load that dictionary while working on your medical thesis.

This is useful but not perfect – I happen to think emacs has the advantage here – and consequently prefer to exit vim and use the ispell program to spell check text files already written; the advantage is no highlighted text distracts you. However, as an editing tool, in-program spell checking can be a useful and productive feature.

16 Macros

The easiest way to deal with repetitive text is to assign it to a keyboard macro using the `imap` command. If you enter `:imap „br Best regards,`, the effect will be to map the expression "Best regards," to the key combinaton „br. Try a couple of these so you are familiar with the possibilities. Choose something exotic like the double-comma combination I use here to avoid stumbling on those keys in regular text. Note that once you've defined a macro, the cursor will change when typing if you begin to enter those first keys while it decides whether or not what you're typing triggers a macro. When you quit Vim, the macros will vanish.

For a longer term solution, put the macros in a file and source it, as follows. Let's say you create a file called "keystrokes.vim" Stuff it with lines like `imap „br Best regards,`. Then, from Vim, issue `:source keystrokes.vim` and the new functionality will be imported.

17 Learning more about Vim

While running Vim, the `:help` command is a great resource, if you know the name of what you're looking for (for example, `:help wrap`), and not so helpful if you don't. For more general questions, tips, tricks, FAQs, and similar, Vim's web site is particularly useful: `www.vim.org`. In particular, the vim tips wiki is a repository for all sorts of tricks. Once you've mastered the basics, have a look at what the Vim experts have contributed. At `http://vim.wikia.com/wiki/Vim_documentation` you can find several other high quality sources of information.

I highly recommend Steve Oualline's book *Vi Improved – Vim*. Although it was written for version 5.7, it is still mostly up to date and covers a lot more than this Woodnotes Guide does! It is even available as PDF download.

18 Acknowledgments, License, and Version History

As usual, no man is an island. Thanks first of all to Norman Kraft (http://zenwrites.com who read my Emacs for writers guide (http://therandymon.com/content/view/16/98/) and suggested this document. Thanks as well to Laurent Gr'egoire (http://tnerual.eriogerg.free.fr) for his fantastic Vim Reference Card, which helped get me started when I was learning Vim, and has been a lifesaver ever since.

This document is published under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 license.[9] Please send comments, criticisms, and corrections to me at the email address found at my website. Enjoy this guide: I enjoyed creating it.

- July 2009: first draft

[9] http://creativecommons.org/licenses/by-nc-sa/2.5/