Wrap Your Mind Around Vim 8.0

John Filippone

April 18, 2017

0.1 ideas that need to be fit in somewhere

1. This book does not go into detail about how to use vim, just what it does. But because vim is so customizable, the how does not matter very much once you know which features you want to use, if it turns out that the way to do those things in vim takes a lot of keystokes and are therefore not effective, you can remap keys or write scripts to get the same functionality in a totally customized way that works for you 2. section on how to learn and practice effectively. Use the vim help files and the internet. Integrate a few commands at a time take it slow at first, think of the most efficient way before you make an edit.

0.2 Purpose

The object of this book is to help the Vim user efficiently learn the tool by enumerating the vast capabilites of Vim 8.0 concisly and simply.

0.3 This Book's Unique Approach

This is not a 'HOW to use Vim' book. It is more of a 'this is WHAT Vim can do; now that you know what it does and what the features are technically called, you can easily learn how to use only the features of Vim that you actually care about on your own' book. The approach of this book is based on the premise that learning HOW to use a particular feature in Vim is easy; the hard part is figureing out the WHAT feature you want to learn. Once you know what the thing you want to do is and what it is called, all it takes some internet searching and some practice. Most people get stuck in Vim because they do not understand the capabilities of the tool or the technical terminology associated with it. Either the user does not know that Vim is capable of doing the thing they want or the user knows what they want and suspects that Vim can do it but has no idea what the feature is called. This book provides a straight forward, consice, and easy-to-understand way of enumerating WHAT Vim does so that you, the user, can get on with the easy part of learning HOW to use only the features you want.

0.4 Important Historical Perspective

When Vim was created, it was made by Bram for Bram, meaning: the engineer who developed Vim initially, developed it for his personal use. He developed the features that he personally wanted to use. As years went on and Bram continued developing, the community asked for other features that Bram did not necessarily care about. He implemented them anyway as long as he felt enough people would enjoy having them. After decades of development, Vim has become a vast collection of features. There are too many for any one person to really need or want or use them all. The vast set of features is meant to accomodate a vast set of users where each user only needs and/or wants a small subset of all features. It is for this reason that this book focues on efficient learning of Vim rather than complete learning.

0.5 How Most People Learn Vim

Most people are thrust into Vi and/or Vim and are forced to frantically learn the basic editing commands. Chances are that if this is the case we are talking about an engineer or scientist; let's call him Steve. Steve realizes after learning hjkl cursor navigation that the impact of editor efficiency on his workflow is significant so Steve sets out to learn some more useful commands on the internet or perhaps a book. Unfortunately, most books for Vim are recipe books. They provide tutorial on specific features or describe great ways to solve common problems with Vim. These are great for picking up tricks but you will brush through many of these before learning the key set of Vim features that work best for you and the task you are applying them to. Steve gets tired of that pretty quickly. The internet can be very helpful if you know what you are looking for as long as you don't fall into a wormhole of the extremely mystifying power-user language. The weird dialog about buffers and Ex-mode etc. is easy enough for you to sift through but the real trouble for Steve is the former. When you start using Vim you have no idea what you want to learn because you have no idea what Vim can do! Even though Steve has a vuage idea of what functionality he wants, he does not know the proper Vim terminology for the thing he wants to do. That makes searching for a how-to very hard. Naturally, this is where Steve stops learning and decides he is relatively content with getting by on his fair amount of editing efficiency. Every once in a while Steve sees his coworker doing something

slick in the editor and says, "hey...how did you do that?" Steve might also learn something new after getting fed up with doing some repetative task and frantically searching the internet for a solution but Steve has the same key issue. Steve is smart and can learn to do just about anything, but when it comes to Vim he has no idea what it is that he needs to learn.

0.6 How You Ought To Learn Vim

- Understand what Vim has to offer.
- Pick a subset of key features you need to learn in order to have an efficient workflow for your specific task.
- Learn each of these features through the Vim User Manual, books, and/or the internet.
- Practice using them enough to commit them to muscle memory for a truly efficient workflow.

This is an efficient method of learning Vim or any tool. Only learn the features you actually care to use. Know what the tool can do so that later on down the road when you need some other functionality, you know what to look for.

0.7 How to Use This Book

You can read it straight through or you can skip to the parts you really care about. It will be valuable to at least skim all of the material so that you come a way with an understanding of the full scope of Vim capabilites. If you are mystified by the terminology in this book, see the Vim Terminology section in the back. That section will also give you a general scope and sense of how Vim is structured.

0.8 Vim Terminology

Buffers, word, WORD, motions, modes (normal, command, ex, insert, visual), commands, yank v copy, put v paste, registers, text objects, windows,

tabs, scripts, plugins, vimrc, dotfiles, NERDTree, Pathogen, Vundle, mapping, abbreviations, macros, autocommands, options, text object text object is like the w in cw command, one edit, movement command, pattern in context of searching, bottom-line command (includes searches and anything on bottom line that is not formally a command), the ex editor, toggle options (note the set option—nooption—option? syntax), set, map, ab, concept of word vs WORD, ctags, tag stacking or tag stacks, power user, transparent editing, session, instance of Vim, buffers (active, inactive, hidden, unlisted, modifiable, read-only, read errors), special buffers (quickfix, help, directory, scratch), alternate file (saved in the hashtag register) or alternate buffer, VIMRUNTIME dir,

0.9 Inner workings

General form of (page 21 O'Reilly Learning the Vi and Vim editors) (command)(number)(text object) or (number)(command)(text object) for change, delete, yank, and put what Vim puts in the registers (sometimes called buffers?) switching between modes concept of a word concept of an edit buffers tags priority and kind gvim menu items priority structure syntax groups and subgroups for coloring (are they used for anything else?) map overrides normal vim functionality order of initialization files like vimrc, exrc, etc. viminfo and retaining session state

0.10 Vim Capabilities

Writing text basic insert/append completion whole line current file keywords dictionary option keywords initially dictionary option is empty; must add a dictionary file thesaurus option keywords initially thesaurus option is empty; must add a thesaurus file can search multiple thesauruses possible to use a programming thesaurus i.e. retrieve getchar getcwd get direntries getenv getgrent can get info in the pop up menu about which thesaurus each suggestion comes from current and included file keywords tags (ctags) filenames searches only current directory macros vim command completion (useful for developing scripts) user-defined through completefunc option omni-completion available for c, css, html, javascript, php, python, ruby, sql, xml, maybe more by now spelling suggestions generic complete with ctrl-N this does potentially all

the other complete methods all in one the exact complete methods used are defined by the complete option start inserting at nifty places end of line start of line beggining of new line above or below smart indenting :set autoindent (next line starts on same indent level on enter) indent in insert mode ctrl-t and ctrl-d automatically insert leading comment characters when writing a comment that exceeds length of line defined by wrapmargin or textwidth audoindent and smartindent cindent probably best for most programmers customization cinkeys: keys that trigger a reevaluation of indenting cinoptions: customizes a lot of the syntax related indenting behavior; look more into this cinwords: words that trigger a reevaluation of indenting (case sensitive regardless of ignorecase option) indentexpr defines an expression that is evaluated each time a new line is created in a file the expression evaluates to a number of indents for that line scripts exist in VIMRUNTIME for most languages already; use 'filetype plugin indent on' to use the appropriate one based on filetype regular indenting use tab in insert mode tab over syntax or otherwise defined blocks through various methods inserting tabs vs spaces expandtab makes the TAB key insert spaces override this with CTRL-V before writing a tab (useful for Makefiles) insert x amount of something at once i.e. 3iabcESC writes abcabcabc put from register auto text wrapping (instead of hitting enter) read command !sort filter (:r !sort file-that-needs-sorting) alphanumeric sort lines registers hold anything, even edit commands registers hold read only metadata like filename create custom abbreviations for writing text with :ab auto generate matching "" " [] () HTML-tags etc. digraphs there are built in shortcuts for typing; just have to learn them easily generate box outlines for box comments with abreviations abreviations Cursor Navigation hjkl search and find as a motion moving vertically through wrapped line searching by word, sentence, paragraph, and section (can set different macros for identifying paragraphs, and sections) beggining/end of line jump/goto to line by number (G or goto command) to column number with num prefix and — goto top, middle, or bottom line on screen (H, M, L) num prefix to go x lines away from top or bottom jump to matching or closest ([and more with to the nth percent of a file to the nth byte scrolling automatic scroll as you type passed bottom scroll by screen or half screen scroll by line with cursor in place scroll relative to cursor (zENTER, z., z-) scroll relative to any line (same as z with numeric prefix) return to previous cursor position save and goto invisible bookmark called mark see all marks with :marks special marks exist bookmarks are not stored in file; stored in session Editing text substitutions (regex support) global by line(s) confirma-

tion context aware substitue global deletes, moves, and copies with :g and ex commands deletion command and all variations yank and put replace change and all of its variations: cw, cb, c2b, c0, C, etc type over existing text toggle upper/lower case copy/paste within a single file yank or delete into reg then put from reg copy/paste between files use ex commands or vi commands to save content to register (or just regular yank) then put (lose undo history when switch between files; use tabs instead) read command will copy a full file into the file you are editing copy/paste with basic point right lick method when smart indenting is on this can result in the progressivly cascading indent problem; solve with set paste? look into this transpose characters (xp) join lines move blocks of text with :g command and pattern matching (helpful if applies to tons of instances in same file hense :g) ex commands usually only useful when need to do something specific everywhere; otherwise there are easier ways to do things still useful to at least be aware of the way ex works and can be accessed in vim vim allows for extreme flexibility with pattern matching and basic move, copy, delete commands; one must open thier mind to these possiblities when faced with a huge repetative task it may be solved with one well-formed, probably long ex command nice examples on p84 of OREILLY filter text in file through Unix commands select text with Visual mode to perform an edit on it select with any vim cursor navigation technique select word, sentence, or paragraph that the cursor is in; ex. 4aw selects 4 words when in Visual mode select lines slect blocks (can't do this in your average gui editor) useful for tables insert, append, replace, or change text to multiple lines on the same column indent blocks write right to left black hole register edit binary files -b delete up until predefined mark conifgure backspace key with backspace option Searching search forward/backward with wrapping or not (regex support) fuzzy find automatically search word under the cursor highlighting of search terms quick scrolling through search results combine search as motion command with c d p y find case sensitive option incremental search :set incsearch search history with arrow keys search for def of variable (local or global scope) jump to macro def display macro definition of macro under the cursor search for a word under the cursor in the current file an dany brought in by include directives: [CTRL-I,]CTRL-I Workflow Environment windows (splits) from initialization or during session add/remove/switch windows displaying two instances of same file multi window ex commands move cursor from window to window resize windows size is in lines and columns; change to n value or +/- lines/columns options when you switch to a window it will auto resize

to your winheight and winwidth option values equalalways, cmdheight, winminwith, winminheight move windows around display swap windows with retaining size or retaining window layout rotate windows of a column or row tabs tabnew, tabclose, tabonly, tabprev, tabnext, CTRL page up/down cycle from last tab straight to first and vis versa status lines for each window is a toggle :[n]split [++opt] [+cmd] filename equalsalways option to always make new windows equal size look into :new and the autocommands it executes :sview and :sfind look into conditional split commands p179 OREILLY execute command on all windows with :windo open tag or file (on path defined in option variable, path) in new window or tab close windows quit, close, hide, and close all others hidden option customize vim stuff with env vars edit multiple files vim f1.txt f2.txt ... see files with :args use next and previous to shuffle through or shortcut to alternate file move around to different files with buffer switching commands save current session settings to vimrc file with :mkvimrc filename home dir dot file executed first then the one in cwd; possiblity for different settings in different envirionments many ways to get vim to display current vim state (options, abreviaitons, registers, etc.) ctags comatability :tag command (look further into this) :tags shows list of tags that you have traversed through :tagselect shows instances of the same tag tag stacking tags option defines where vim looks for tags compatible with etags copy to system clipboard from Vim in gvim this is default I think? verify! edit-compile-edit cycle with quick fix find out if this can be done in many languages, not just c compile from vim with :make filename define program used to make with makeprg option jump to location of errors easily move between locations of all the errors rename refactor with vimgrep and quick fix p 283 O'REILLY automatically jump to compile error lines File management creating files opening/closing files opening files read only open file with cursor at position on line number on first occurance of pattern on last line recovering edited buffers after crash write to open file or any file by name or create new file and write to it write or append part of a file to a new file open a bunch of files in one session and switch between them on one screen open file reguardless of file type edit binary files; set the binary option for safer editing; this turns on/off some vim settings that make editing binaries safer but editing binaries is still not recommended file browsing delete and rename files and directories search thorugh dir like a normal vim file open file in split window can you do a quiet open? where file goes into a new tab? directly edit compressed files and directories (transparent editing) configure vim to produce backup files teach vim new filetypes based on unfamiliar file

extensions (ex. let vim know that a .inc extension is a C file) command tab completion :e :h etc. vim diff Undo/redo multi level undo/redo (aka infinite undo?) by default undolevels is 1000 undo by line repeat an edit (this is not redo; it is repeat edit) special tricks for repeating substitutions p80 of OREILLY undo/redo branching; see usr32.txt Folding nested folds fold types syntax based use set foldmethod=syntax and set foldenable indent based use set foldmethod=indent use set foldlevel=n to define how much you see and toggle foldlevel with zm and zr regex defined manually defined use any motion commands with a z command marker defined diff differences are folded operate folds recursively or one at a time (open, close, delete, toggle) (is toggle same as open/close?) look into foldenable an dfoldlevel foldcolumns visual que on the left if you're into that execute command on a folded line will execute that command on every line in the fold keep manually defined fold with :mkview and :loadview edit files remotely scp, ftp, sftp, http, dav, rcp backups automatically backup files specify location for backups backup, writebackup, backupcopy, backupdir, backupnext options save session state viminfo file managed by the viminfo option parameters save lines for each register up to n lines number of search pattern items to save number of command line commands to save max files vim will maintain info on contains command line history search string history input line history registers file marks last search and substitute patterns buffer list global variables mksession :mksession filename will save just about everything about the session into a file that can be sourced in a vim session later managed by sessionoptions option what you can save empty windows windows and tabs hidden and unloaded buffers current dir manually created folds, state of folds, local fold options global vars help window local options i.e. options set locally to a particular window or buffer options window sizes dir in which session file is located unix end of line format window position on the screen quick command history reference with ctrl-f in commandline GUI wrap or no wrap for horizontal lines vim might shift lines left and right to make them fit? check this out slidescroll option limits such shifting sidescrolloff option may also be useful listchars option defines characters that act as visual cues for lines that have more content to left or right of the screen left/right scrolling? folding Color syntax highlighting (filetype detection beyond extension name) syntax defining files are in VIMRUNTIME/syntax column coloring colorschemes I recommend base16-3024 customize with colorscheme, hightlight, and background options redefine coloring for syntax groups with highlight command :highlight will show you all coloring for all syntax groups customize syntax group coloring for specific file

types by including it in an after script ex. for xml put a file called xml.vim in the /.vim/after/syntax dir that contains your highlight commands also put that dir in runtime path like this :set runtimepath+= /.vim/after/syntax create your own syntax file possibly for a custom file extension; in this case you can get vim to automatically detect your extension and use your syntax color test highlighting specific column(s) search terms matching [(etc. line numbering normal numbers relative numbers redraw screen Crl-l mode indicator show white space chars with :list gvim basic point click and scroll you expect gvimrc checkout scrollbars; customizing them; use of scroll bars with multiple windows menus control order of root menus at the top (vim has a menu priority system including default menu priorities for existing menus that might be important to mention) control order of items within menus control spacing between menu items make your own menus and menu items that execute vim commands tear off menues toolbar treated similarly as a one dimentional menu uses bmp images as icons define tooltips for icons Windows gvim self installing executable basic copy/pasting is compatible with System clipboard look into gvim specific options Run Unix commands from vim :!command is this technically filter? mix this with motion commands :sh read output of unix command directly into file (:r !command) convert files to html with :TOhtml command show differences between two files vimdiff non unix vim versions come with a vim version of diff (unix of course uses the built in command) diffexpr option defines replacement expression Customization Vim script .vim scripts in vim runtime directories serve as good examples if/else inline, elseif and else are optional datetime support variables optionally explicitly define variable scope concatinate with . built in functions look into help file usr41.txt(has underscore after usr but tex hates it) for details on over 200 built in functions execute echo filetype detect exists define functions with/without args (can you use args?) call get function to call periodically statusline trick p.202 of O'REILLY better way? arrays dictionaries context conversion for variables (most notably string to int..verify this!) execute function based on filetype (autocommand) plugins basic ex scripting called from bash or :so possible to do complicated things if you know what you're doing sorting glosseries updating last mod time of the file changing tabwidth for different file types can use comments use spaces in ex commands to make them more readable execute script based on time of day everything you put in vimrc is technically vim script Autocommands execute arbitrary script on event happens decide which files will be subject to the autocmd script groups delete commands when they are no longer needed allows

for dynamic changing of autocommands and which files they execute on can delete groups but make sure to delete all commands in group first evaluating expressions (not only int math) Remapping keys remap command mode keys :map remap insert mode keys :map! remap function keys that are mapped by the terminal (p.110 OREILLY) (look into this more) looks deeper into which exact keys can be remapped command combinations like ddp Macros (aka @-Functions) pre/post edit routines VIMINIT env var .vimrc for initialization state (and gvimrc; executed after vimrc if using gvim) .viminfo for session context can define scripts that run when exiting buffers pre and post scripts run automatically when switching buffers in the same session Regex support character classes [: :] collating symbols [. .] equivalence classes [= =] for substitution, regex works in search portion, some other constructs exist for the replacement portion matching beggining and end of words (not a general regex thing? only vim thing?) look into isident, iskeyword, isfname, magic, and isprint options Applied Vim to specific languages HTML Python Mouse compatability Support Vim help files with tags vimtutor use K to open man page for word under cursor K takes a prefix number i.e. 2K to specify section number in the man page keywordprog option defines progrm that is executed by K i.e. man in unix keywordprog can be modified for example to nothing, signaling vim to use :help the iskeyword option defines what K defines as a word Backwards compatible with vi Vim configurations when compiling Vim