Wrap Your Mind Around Vim 8.0

John Filippone

May 27, 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 new and veteran Vim users efficiently learn the tool by providing a full and concise enumeration of the vast capabilites of Vim 8.0.

0.3 Important Historical Perspective

When Vim was created, it was made by Bram for Bram; meaning, the engineer who developed Vim did so 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 expansive collection of features is meant to accomodate a diverse set of users where each user only needs and/or wants a small subset of all features to achieve their ideal workflow. It is for this reason that this book focues on efficient learning of Vim rather than complete learning. Getting the maximum utility from Vim on your particular workflow does not require you to know how to use all the features but rather only 5-10 percent of them.

0.4 This Book's Unique Approach

This is not a 'HOW to use Vim' book. Instead this book focuses on teaching WHAT Vim can do. 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. Because Vim is such an extensively customizable tool, the HOW of a feature is not particularly important when it comes to achieving efficient workflow. If the default invocation of a feature does not work for you, there is almost definitely a way to change the invocation to something that does work. Any how-to elements of this book will be minimal for the sake of brevity. This book will instead into detail on the WHAT is possible in Vim and WHAT the proper terminology is. 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. This is the most efficient approach because it allows you to avoid leaning HOW to use features you don't need or want. After reading this book you will have effectively wrapped your mind around Vim. You will have a strong understanding of all the moving parts and which features exist. Armed with that understanding you can quickly learn the features you need for your particular workflow. More importantly, the next time your workflow changes, your knowledge of Vim will allow you to quickly assess wether or not Vim is the tool for the job and you will know what to look for.

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. Even very smart users get stuck in Vim because they do not understand the capabilities of the tool or the technical colloquialisms associated with it. When struggling with a problem, the new user has two main issues. They do not know that Vim is capable of doing the thing they want or they know what they want and suspect that Vim can do it but have no idea what the feature is called. 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 hundreds of pages before learning the key set of Vim features that work best for you and the task you are applying them to. Most will try these 'HOW to' approaches, then quickly give up

on them and default to being satified with a cumbersome workflow. 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 mystifying power-user language. The weird dialog about buffers and Ex-mode etc. may be easy enough for you to sift through but the real trouble is the former. When you start using Vim you have no idea what you want to learn! Most have a vuage idea of what functionality they want, but do not know the proper Vim terminology. That makes internet searching for a how-to very hard. Most users will pick up a trick here and there over the years from their coworkers or after getting fed up with a difficult workflow, but never fully wrap their mind around Vim.

0.6 How You Ought To Learn Vim

- Understand what Vim has to offer. (Wrap Your Mind Around Vim)
- 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.
- When you change your workflow, use your understanding from step one and revert to step 2.

This is an efficient method of learning Vim. 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. This book aids you with the most important part of this process: Step one.

0.7 How to Use This Book

You can read this 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. Key terms will be displayed in bold text throughout the text in order to dispell confusion between words that have a unique meaning when used in a Vim context.

0.8 Vim Terminology

Buffers, word, WORD, motions, modes (normal, command, ex, insert, visual, select, operator pending, insert normal mode, replace mode), 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, readonly, read errors), special buffers (quickfix, help, directory, scratch), alternate file (saved in the hashtag register) or alternate buffer, VIMRUNTIME dir, events, swp or swap files, status line, argument list, block and non block text objects, jump and the distinction between and jump and a motion (if it goes into the jump list it is a jump; motions are always within the file; if it goes to another buffer it is a jump; some motions within file are classified as jumps and go in the jump list)

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?) special registers, black hole register, append to reg, expression reg, yank reg, 2 clipboard registers switching between modes concept of a word and WORD p 184 of Oualline concept of how vim interprets filenames, comments, identifies, definitions, and printing characters is related concept of an edit using arrow keys during an insert affects the edit concept of operator motion grammar 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 ctrl-Q and ctrl-S should be avoided in mappings because they are commands for the terminal to stop and start showing otuput; look into this +cmd argument; command line argument for vim that starts file at a line number or first instance of a pattern like this: vim +97 file.txt; also works for some vim commands concept of setting options and maybe an organized overview; sufixes ? and (ampersan) and prefix no; options are boolean, numeric, string-related, +=, -=, and carrot= for string option values can reset all to default with :set all(ampersan) set multiple options in one line 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 going to have too look up most update vim 8.0 info on regex support and learn about regex in general a lot so that i can speak the regex language in the book magic and very magic Basic ways to use features gui command line args options ex commands mouse modelines; comments that define options automatially p268 Oualline

0.10 Vim Capabilities

Editing text basic edits change delete insert substitute append yank put move join join lines joinspaces option will put two spaces in between sentences when joining global edits deletes, moves, and copies with :g and ex commands move blocks of text with :g command and pattern matching (helpful if applies to tons of instances in same file hense :g) basic edit combos transpose characters (xp) transpose lines (ddp) ex commands (maybe don't need this section) 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 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 repeat a visual selection with gv move cursor between beggining and end of selection with o write a sleection to a file pipe selection through unix program like sort: select text then use !sort in normal mode (performs operation on lines whether or not you are using visual line mode select mode behaves more like what 'normal' editors do; backspace to del or type to del and replace gh, gH, gctrl-h use mouse to do select mode with selectmode option set to mouse? can toggle between visual and select mode combine edits with motions Insertions basic insert/append start inserting at nifty places end of line start of line; this can be first non blank char or first column regardless if it is blank 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 use command with number arg use comparator chars use ctrl-g and ctrl-t in insert mode insert the char right above or below the cursor with ctrl-y and ctrl-e in insert mode insert x amount of something at once i.e. 3iabcESC writes abcabcabc completion whole line current file keywords dictionary option keywords initially dictionary option is empty; must add a dictionary file or multiple thesaurus option keywords initially thesaurus option is empty; must add a thesaurus file or multiple 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 sleep command can be useful for pausing macro execution 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 where vim searhes for words is managed by complete option path option defines where vim looks for files abreviations create custom abbreviations for writing text with :ab abbreviations can be mode specific like maps can list abbreviations use ctrl-c instead of esc key if you want to exit insert mode without completing abbreviations noreabbreviation to avoid chains auto generate matching "" " [] () HTML-tags etc. command line scroll thorugh commands with tab, shift-tab, up and down special characters digraphs there are built in shortcuts for typing; just have to learn them ctrl-v in insert mode followed by char code insert char literally with ctrl-v insert mode followed by a non digit char like tab comments easily generate box outlines for box comments with abreviations maybe us external program, boxes, to make comment boxes auto wrap comments this means when you are writing a comment, the editor will automatically input the comment leader on the next line does this work for languages other than c? managed by formatoptions option comment and uncomment blocks easily with block visual mode comment and un comment with ex command like this :normal i// write in foreign languages including right to left languages, farsi and hebrew also other languages; maps keyboard to other language characters; need to look into how this is done more write right to left inserting evaluated expressions with expression register invoked with ctrl-r= inserting tabs vs spaces expandtab makes the TAB key insert spaces override this with CTRL-V before writing a tab (useful for Makefiles) combination of tabs and spaces is possible with softtabstop and smarttab and shiftwidth control size of tabs expandtab does not affect existing tabs but :retab command will convert tabs to whatever vim is currently configured to shiftround will make vim always tab to a column that is a multiple of shiftwidth substitutions (regex support) global by line(s) by visual selection with the use of ex command confirmation context aware substitue change and all of its variations: cw, cb, c2b, c0, C, etc put word under cursor directly into sub command with ctrl-r ctrl-w replace type over existing text toggle between insert and replace mode with insert key Virtual replace with gR and gr; difference is it will act as if tabs are spaces always copy/paste put from register registers hold anything, even edit commands registers hold read only metadata like filename from insert mode with ctrl-r and ctrl-r ctrl-p read command !sort filter (:r

!sort file-that-needs-sorting) 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; paste is a shorthand for setting a bunch of other options if you use this you can use a pastetoggle key to switch in and out of paste mode quickly or use plus register; this will work reguardless of how paste is set? toggle upper/lower case editing with unix commands filter text in file through Unix commands 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) alphanumeric sort lines edit binary files -b formatting justify text so that it looks nice; i.e. put line breaks in nice places automatically VIRUNTIME/macros/justify.vim gq does it gqmotion also works i.e. useful with paragraph motion gqip will do it to the paragraph from inside the paragraph instead of having to be on the first line or first char of paragraph align text; left right center :1,5 right 30 30 is the width so text will align right to column 30 use unix program fmt for formating set formatprg=fmt edit a bunch of files the same way batch file i.e. a vim script containing a set of commands that you can run on a file from the unix command line use bufdo to run a source command on all buffers; the source command can source a batch file encryption rot13 encrypt with g?? editing matching structures text object motions inside and around Surround.vim plugin by Tim Pope allows surrounding visual selection with matching chars matchit.vim plugin allows advanced matching beyond the basic chars Cursor Navigation motions hjkl moving vertically through wrapped line search and find as a motion by word, sentence, paragraph, and section (can set different macros for identifying paragraphs, and sections) beggining/end of line moving to first non black char 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 showmatch will make it so that when you close a matching pair cursor will do a quick jump to matching pair and jump back; kind of annoying if you ask me can also jump to next or previous unmatched pair or conditional (if, else); kind of shady, test this out; vim can detect unbalanced matching but not which element is unmatched?

to the nth percent of a file to the nth byte to start or end of method to next or prev or to begining and end of comments jump stack change stack one change stack for every buffer tag stack jump to last insertion with gi one jump stack for every window 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) scroll specified amount specified by scroll option control amount of scroll when cursor moving toward the edge of the screen 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 global marks transend files within a vim session; what happens when combinging other commands with a motion to a global mark? search as a motion jump to file under cursor with gf suffixesadd can help with filenames that lack extension 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 find case sensitive option use backslash c and backslash C to override ignorecase option smartcase incremental search :set incsearch scroll through 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 and any brought in by include directives: [CTRL-I,]CTRL-I search offset; change locaiton of cursor when finding a search result; i.e. /keyword/b2 will put the cursor on the y of keyword when that keyword is found 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 auto text wrapping (instead of hitting enter) left/right scrolling? 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 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 make menues dependent on mode special menu names "ToolBar" and "PopUp" noremap menu mappings 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 title bar string make title bar say the name of the file being edited also can manually set title bar adjust max length of title bar string can also manaage string in minimized window bar customize mouse behavior turn mouse on and of by mode mouse mappings double click time turn off mouse pointer when typing make select mode default over visual mode or choose toggle method invoke certain dialog boxes with commands instead of menu items confirmation dialog boxes when action will delete data browse options; badass; must write about this; maybe display the browse options menu in the book; allows you to browse an organized list of options, see defs and toggle; :browse set set font with guifont; set guifont=* will bring up a menu of fonts to choose from change style of the cursor convert files to html with :TOhtml command vimdiff gui execute vimdiff from commandline non unix vim versions come with a vim version of diff (unix of course uses the built in command) diffexpr option defines replacement expression showcmd vim shows messages on the command line after certain commands like write, you can make these

short with shortmess option errorbells and visualbells format status line customize reporting changes with report list mode customize chars list mode uses with listchars Workflow history scroll through history on commandline for ex commands for searches command-line window q: for commands q/ for searches edit commands in the command line window even with ex commands exit command-line window with :q just like any other window ga will show you ascii number or char under cursor Backwards compatible with vi Mouse compatability behave option can help make mouse behavior more like the way you are use to 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 move window to a new tab tabs can vim open silent tabs? tabnew, tabclose, tabonly, tabprev, tabnext, CTRL page up/down cycle from last tab straight to first and vis versa tabs can contain windows reorder tabs split search; same as * command except does it in a new split window; ctrl-w ctrl-i 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; and generally add s to a lot of commands to get them to do the same thing in a split window 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 only (close all others) hidden option windows can have their own status line or not bind the scrolling of two windows vertical and/or horizontal with scrollbind look into syncbind as well preview window 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 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 related to jump stack ctrl-o and ctrl-i for back and forward check for scope of tag stack and jump stack accross windows and

tabs and multi buffer window contexts tags option defines where vim looks for tags compatible with etags copy to system clipboard from Vim system clipboard register is "* ? verify this 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 look into redir command 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 swap files vim will try not to override swp files; it will name the next one swo then swn and so on control when swp file gets written; 4 seconds or 200 chars is default control where swp files are written; can be multiple files preserve command will write to swp 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 make directories if you open a buffer that does not exist in a dir or chain of dirs that does not exist, create the missing dirs with :!mkdir -p 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? edit file that is under cursor with gf which is same as :find filename path option defines where vim looks for filenames wildcard menu for comandline filename completion siffixes option defines extensions that are given low priority in wildcard menu look into wildmode Explore, Sexplore, Vexplore commands that are shorthand for open file browser for current active buffer dir in current window, new horizontal split window or new vertical split window directly edit compressed files and directories (transparent editing) configure vim to produce backup files automatically backup files specify location for backups with backupdir backup, writebackup, backupcopy, backupdir, backupnext options vim will add a extension to backup files; manage the extension with backupext patchmode? difference between backup and write backup? if you are in a write protected file and you don't want to loose changes just write to another file from vim with :write filename teach vim new filetypes based on unfamiliar file extensions (ex. let vim know that a .inc extension is a C

file) fileformats this has to do with file formats with regards to end of line chars; unix is line feed, apple is return and dos is both fileformats option manages this vim auto detects unix,apple,dos file formats regarding the end of line chars and is able to read all formats save a file in a different file format control wether or not files end with and EOL char with endofline option encryption key option holds your encription key encrypt an existing file or new file vim encryption is weak when a file is changed by another program while you are editing and then you write, vim will give you a warning as ask if you want to continue remove white space at ends of lines with substitude command; suggestion is to rig an auto command to do this on write cd and pwd command mode commands lcd for changing context of specific window 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) repeat an ex command with @: repeat a substitution with ampersan repeat an insert with ctrl-A special tricks for repeating substitutions p80 of OREILLY undo/redo branching; see usr32.txt ex commands are not always straight forward to undo edit files remotely scp, ftp, sftp, http, dav, rcp edit files that contian a given word vim ‘grep -l ’special-term’ *‘ 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; full history with :history command; can set amount of remembered commands configure vim to execute python, perl, tct, and probably more inside of vim can make insert mode the default Customization creating custom operators create an operator to comment code help file is map-operator creating custom motions help file is omap-info home dir dot file executed first then the one in cwd; possiblity for different settings in different envirionments if it is a thing, you can probably customize it conifgure backspace key with backspace option

Vim configurations when compiling Vim Vim script most common commands have one or two letter shortcuts; when using these in script makes them hard to read; instead you can use explicit commands like :copy and :delete for just about anything .vim scripts in vim runtime directories serve as good examples if/else inline, elseif and else are optional while datetime support variables optionally explicitly define variable scope handle strings and ints literal string with single quote or simple string with double; simple expands escape characters, literal string does not special variables for env vars, registers, etc built in vim variables ints can be decimal, octal, or hex, (or binary?) can delete variables comparisons can do basic boolean comparisons on ints and strings can check if string matches regex with comparison; string = regex or the compliment: string ! regex ignorecase string comparision to ignore add ? (i.e. ==?); must match case: add hashtag default string compare honors ignorecase option enter filenames dynamically file name under cursor, curretn file, alt file, lots more modify that filename by adding/removing abs path, get extension etc. to experiment: put cursor over the name of a file :echo expand("¡cfile¿:p") this will echo the abs path of the file under the cursor this could be pretty powerful..look more into applications of this 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 can also echo in color filetype detect exists confirm; dialog boxes wow cool such gui define functions with/without args call return if you define a function that already exists you will get an error but you can override it with function! FuncNameThatAlreadyExists range functions? function FuncName() abort will define a func that aborts on fist error function FuncName(start, ...) means it reqires one arg and can have up to 20 more stored in a:1, a:2, etc; a:0 is the count of extra args list user def functions with :function can delete functions defind command mode commands kind of like mapping for command mode; must start with capital letter list user def commands with :command can clear user def commands can take arguments predetermined number 0 or 1 any number one or more range; look in to what it means to have a range argument??? 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 autocommands are by default not triggered by other autocommands but you can specify that they can be if you want can choose to ignore events evaluating expressions (not only int math?) add, mult, sub, int div, mod, negation increment and decrement integers even in octal and hex form nrformats manages which number formats are recognized Remapping keys map commands affect keys in one or more modes; operator pending mode comes in to play here with omap 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; look into the ability to map alt key, p344 Oualline says it can be done on gui chains of maps are possible but also preventable with noremap can also set noremap option to make noremap the default behavior; might break some scripts! unmap; takes only one arg and sets that arg to default functionality remove all mappings with mapclear list mappings with :map command combinations like ddp Macros (aka @-Functions) feature: macros get killed when a motion command fails technique: write a macro for one line and visually select lines and use normal command to execute macro on each line technique: use argdo or bufdo to run macro on multiple files use let command to save a var and increment it and = reg to put it in the text to create number lists edit macros by putting them in the doc and yanking them back into the register after editing 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 customize how vim defines constructs sentence, paragraph, word, WORD, comment, filename, identifiers, keywords, matchingpairs, definitions, includes, errorformat customize how indent events are triggered cinwords, cinkeys, cinoptions customize :make command choose binary or linier search for grep verbose option so you can see what vim is doing under the hood there are clearly defined levels for this Oualline 402 Applied Vim to specific languages HTML Python 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 using vim to browse man pages is lame unless you run it through ul -i like this: man date — ul -i — vim; verify this alternatively use vim substitute to fix; maybe rig it up in a autocmd; Oualline p 167

0.11 Outline

0.12 More Resources to Research

Vim resources I need to check out and maybe provide vi Lover's Home Page by Thomer M. Gil at http://www.thomer.com/vi/vi.html Vi Pages by Sven Guckes at http://www.vi-editor.org appendix A of O'REILLY contians important info about command syntax structure contains documentation of ex commands appendix B of O'REILLY contians documentation of some "most important" options tutorial from unix world magazine by walter zintz old url: http://www.networkcomputing.com/unixworld/tutorial/009/009.html