# Why I love Vim: It’s the lesser-known features that make it so amazing

# by Amit Kulkarni

# Since I started using Vim in 2016, I’ve discovered several lesser-known features that Vim offers out of the box without any plugins.

# Can you cover some basics before you start rambling about these new things?

# Oh sure! Before I copy paste a few commands from a cheatsheet, I am going to make a bold assumption: you wouldn’t be reading this if you wanted a cheatsheet and you already knew Vim basics.

# You might have just heard that Linux distributions ship with a default command-line text editor called Vim, and may want to just give it a try.

# So, let’s assume you are completely new to this whole game and start from just what we need as basics (without history/boring theory).

# NOTE: If you know the basics, click here to scroll past them

# What’s your deal here compared to tons of other articles on Vim?

# Most of intro articles on Vim begin with modes of Vim, inserting, saving and exit. If you’re really in the mood of theoretical perfection learning mode, feel free to read whatever helps in wikibooks.

# There are also some great books and articles that tell you that there is a philosophy behind the way Vim works and that the commands in VI/Vim are meant to be combined. Absolutely true and I’m sure you will appreciate it once you get used to the editor and the power it provides.

# I have heard funny stories and seen funny images about learning curve of Vim. Is that true? Is it actually that bad?

# Well, haters gonna hate ? However, according to me, the image that somewhat gives a proper representation of Vim is:

# The majority of the articles on Vim refer to the learning \*curve\* as a learning \*wall\*, but hey, there’s some positivity: look at the other side of the wall!

# For beginners, it’s literally a wall since they have never done anything like this before to use an editor on command line. The thing that appealed most to me when I started as a beginner was the ubiquity of Vim.

# Log in to any (non-windows) machine from any terminal and you can literally get an editor by typing \*vi\* with your eyes closed. The editor will appear in front of you!

# Another thing that appealed to me is the ability to work without a mouse and without wasting any productive time on touchpad or getting a mouse for laptop.

# I know, I know, I can hear some of you yelling “Emacs! Emacs!” I get it. But once I was hooked to Vim, I just never really had any interest in emacs (may be because of the installation required). So, yeah emacs is also great I guess. Feel free to jump ship before you start sailing on this beautiful journey with VI(m).

# I just opened my terminal and typed vi and hit return key. All i see is a welcome screen. I can’t type and I don’t know how to get out of it. Are you sure it’s a powerful editor with capabilities?

# 100% sure. The behavior you just witnessed is the \*wall\* we saw earlier. Trust me VI(m) can do a lot of other things. It just has its own ways of using it. You can edit files, open tabs, split screen horizontally or vertically, browse file system, run linux commands without leaving your file, trigger make builds from your source code without exiting the file, bookmark directories, even better: bookmark lines of a file, find and replace words, of course copy-paste and a lot more.

# Yeah! Like that’s a big deal for an editor to support those. Meh! Everyone does that. What’s the big deal?

# There’s no big deal, the only deal I see is the ability to focus on your file/code without leaving keyboard. Really, if you don’t mind using a mouse, then go ahead and open your MS word/GUI Editor and do all the editing you wish to do.

# Fair enough. But, seriously why not an IDE for some work?

# Okay, so you are a developer and have had some liking/love for an IDE. No, VI(m) is not a replacement for your shiny IDE. VI(m) does not have the out of the box awesome capabilities of your IDE. VI(m) is just small in size (package and installation) compared to the bulky IDEs and is available to use without any configuration and installations. Seriously, VI(m) is no match for some great things your IDE provides.

# Enough talk, show me the basics?

# Sure, before you begin, just keep in mind that any Vim user has to basically deal with command mode and insert mode. There’s no escape (literally, not the Esc key).

# Let’s say you are using some editor and you want to delete a long function in C language. The simple steps you do are: Position your cursor at the beginning of the line, then press Shift + Down arrow till end or use mouse. This action that you had to do to select those lines required you to \*stop\* typing and press keys. Isn’t it? Don’t tell me you were typing something and simultaneously pressed keys to magically select the body of your function.

# Be reasonable. You paused typing and did the selection work to tell your editor that you want to do something with this text (copy/cut/Bold/italics/anything).

# This pause that you took is equivalent to being in command mode in VI(m). This is the time when you tell VI(m) that you want to do some actions on some lines/word/anything and you are not going to type. Now, VI(m) throws you out of insert mode and you are locked out of typing text in your file. Obviously, the other mode in which you can actually type in your file is the insert mode.

# BTW, if you were wondering how are you selecting body of function without selecting text or using mouse, I accomplish that by placing the cursor on the opening braces and using keystrokes: d%

# Yes, that deletes the contents of your function body. No, that’s not some creepy combination of keys to remember! indicates you want to delete something. is going to move the cursor to the end of the matching brace.d%

# Now, that we have established the basic modes, let’s dive into basic VI(m).

# If you know the name of the file you are writing:

# $ vi myfile.c

# If you are not sure of file name and want to start typing:

# $ vi

# As soon as you open vi, you will be in the command mode. To enter insert mode, press . Type whatever you wish. Press to return to command mode. Now you have a few options to exit depending on how you opened vi.iEsc

# If you gave a file name: will write those changes safely to disk. will quit the editor. You can combine these actions with : and key:w:q:wqReturn

# If you did not give a filename: will write the contents to the file and quit the editor. If you are not interested in the text you wrote and wish to exit without saving anything: and you are out! The is required at the end to say : “Yes, I am sure I don’t want to save the contents and I want to get out urgently”:wq filename.cfilename.c:q!!

# [DEMO] Basic vim usage To start the vim editor: Use vim command on shell To start editing a file using vim, use : vim filenameasciinema.org

# There you go! You just created, edited and saved(or may be not) your first vi file. Congratulations ?

# As I mentioned earlier, this is not a beginner’s introduction to VI(m). There are plenty of other articles (I will provide reference at the end of article) to get started. I just inserted that intro so that you’re not disappointed after landing on this page and finding nothing to learn?

# This is the line where beginners say goodbye to intermediate users and head to the reference section for more brilliant intro articles.

# Welcome to the intermediate users. These are some cool capabilities of VI(m) that I wasn’t aware of. But now I use them daily to be more productive.

# For those of you who prefer TL;DR:

# tab-pages

# sessions

# line numbers (+ marks) and copy/paste

# folds

# indention with =

# insert-completion

# netrw

# splits/windows

# :! and a little bit about :make

# Vim tab-pages

# Did you mention tabs in Vim? I didn’t know that existed!

# I know, right! A tab page is a page with one or more windows with a label (aka tab) at the top.

# If you are interested in knowing more about windows, buffers, tab pages: technical details

# Have a look:

# Vim tabs in action

# Steps:

# Open Vim with any file or just Vim: $ vim file1

# Type the contents of file and get into command mode (Press Esc )

# :tabedit file2 , will open a new tab and take you to edit file2

# :tabedit file3 , will open a new tab and take you to edit file3

# To navigate between these tabs, you can be in normal mode and type : or to go to next tab or previous tab respectively. You can also navigate to a particular index tab (indexed from 1) using where, i is the index of your tab. Example: takes you to 2nd tabgtgT{i}gt2gt

# To directly move to first tab or last tab, you can enter the following in command mode: or for first or last tab respectively. To move back and forth : for next tab and for previous tab:tabfirst:tablast:tabn:tabp

# You can list all the open tabs using : :tabs

# To open multiple files in tabs: $ vim -p source.c source.h

# To close a single tab: and to close all other tabs except the current one: . Use the suffix to override changes of unsaved files:tabclose:tabonly!

# [DEMO] Tabs in VIM VIM supports tabs to open multiple files and work with themasciinema.org

# I think this feature enables us to effectively save time by sharing the buffer between tabs and enabling us to copy paste between tabs and keep multiple sessions of different tab set for category of work. Example: You can have a terminal tab with all Vim tabs of source code C files only and you can have another terminal tab with all Vim tabs of header files (.h).

# Tabs provide so much convenience to keep all my files open and access them when I want. However, isn’t it a pain to open all tabs every time I reboot or close and open the terminal?

# Right! We all like to have our own sessions of work in which we work with a set of files and would like Vim to restore that session of tabs the way we left it. Vim allows us to save and restore those tab sessions! ✋

# Steps:

# Open any number of tabs you wish to work with

# From any tab, press and enter the command modeEsc

# Type and hit enter:mksession header-files-work.vim

# Your current session of open tabs will be stored in a file header-files-work.vim

# To see restore in action, close all tabs and Vim

# Either start vim with your session using : or open vim with any other file and enter command mode to type: and BOOM! All your tabs are opened for you just the way you saved it!$ vim -S header-files-work.vim:source header-files-work.vim

# If you change any session tabs (close/open new), you can save that back using : while you are in the session:mks!

# [DEMO] Sessions in VIM VIM allows users to store their work sessions separately based on the projects they are working on. Users can easily…asciinema.org

# Can I copy/cut paste without having to know line numbers?

# Oh Yes! Earlier I used to see the line numbers ( ) of the functions I wanted to copy/cut. Let’s say I want to copy/cut lines 34 to 65. I used (Copy/Yank) or (Cut/Delete).:set nu:34,65y:34,65d

# Of course counting the lines and using or (where is number of lines) is not an option for hundreds of lines ?{n}yy{n}ddn

# There can be some functions that span multiple pages and you don’t want to go down only to forget what was the first line number. There’s a simple way to achieve this without worrying anything about line numbers!

# Steps:

# Enter normal mode, go to the start line

# Type (Mark point with alphabet ‘k’ or use any other alphabet)mk

# Move down (page down or whatever) and move to the end line

# y'k will yank/copy all the lines from start to end

# d'k will cut/delete all the lines from start to end

# I have some annoying long functions at the top of my file and I don’t want to waste my time scrolling or jumping to lines. This may be a lot to ask because this is not an IDE but, by any chance can we fold the code blocks?

# Absolutely! Let’s say you want to skip remembering those line numbers and walk around with your new found love \*the markers\*. Go to the beginning of the function body and type . Now, just go to the end of the function body using (brace matching) or any other convenient technique and press and you’re done!mb%zf'b

# Before and after:

# Before-After

# If you are comfortable using the line numbers, the command is even easier to remember: (fo stands for code fold). Once you have folded your code, it’s easy to toggle between open and closed views using (Open the code fold) and (Close the code fold). Don’t stress it so much. Just use to toggle between open and closed folds ?:5,16fozozcza

# Let’s say you spent considerable time folding your functions in a large file, you would obviously want to retain those folds every time you open that file right? (If not, why did you waste your energy folding them!?), so there’s a solution right in your . Insert the following lines in and your code folds are saved and restored:~/.vimrc~/.vimrc

# autocmd BufWinLeave \*.\* mkview autocmd BufWinEnter \*.\* silent loadview

# I’m usually careful with my indentation but sometimes, I have to edit some other idiot’s source code and it bugs me to edit his/her code without indentation. Are there any magical keystrokes to make that happen?

# Sure! It’s as simple as: . Really that’s all ! ( i is inner object=i{)

# [DEMO] Indentation in VIM VIM allows blocks of code to be indented with a few keystrokes. All you have to do is place the cursor in a block of…asciinema.org

# Before-after:

# Before-After

# All you have to do is place the cursor anywhere within a block you want to indent, press to enter normal mode and then: . Boom! Your entire function body (including inner blocks) is indented.Esc=i{

# NOTE: Don’t expect indentation of your python files ?. It only works when Vim can identify the start and end using opening and closing parenthesis)

# You can also increase/decrease the indentation within a block using : ;i{ to increase a<i{ to decrease in normal mode.>nd

# I may be dreaming but (\*quavering voice\*), I mean I just want to give it a try, Uhmm, I may be pushing it really far with this one but (\*5 second pause\*)..never mind, lets move on to my next question

# Vim is quite open-minded to take criticism or face the fact that it’s not an IDE, go ahead, let’s see what you’ve got.

# Uhmmm, sorry but by any chance (\*panting\*) with any plugin or something, does vim have autocomplete like an IDE?

# ? You may be surprised but yes it does! ? and guess what… \* drum rolls \* \* drum rolls \* \* drum rolls \* \* drum rolls \*

# Without a plugin!

# You heard me right! The only condition for Vim to show you options is “Vim should know what you’re talking about.” It could be through an included source file or defined functions or variables.

# All you have to do is start typing and then press in insert mode.Ctrl+n

# Examples in C, Python and Java

# Just imagine the uses! Specially if you’re writing C code and you cannot recollect the exact OpenSSL library call, all you have to do is include the header!

# [DEMO] Autocomplete feature in VIM VIM has autocomplete suggestions for keywords, function names if the appropriate header files are included or if the…asciinema.org

# Vim auto-complete helping with OpenSSL functions

# Let me remind again: No plugins required ?

# NOTE: The header files may be at some other locations on Mac and Vim may not be able to find them. I just use a Mac to login to a linux machine. So, if you’re using Mac, sorry about that.

# I understand Vim is just a text editor but if you want me to work without losing focus and without exiting Vim every now and then, what options do I have if I can’t remember all the file names?

# Simple, use the file explorer provided by VIM ? Yes, Vim provides a simple file explorer (\*without any plugins\*). Just type : :from any Vim window and you will see an easy to navigate file explorer which can be navigated using ⬆️ and ⬇️ arrow keys. Press Ekey to open a file/directory. Use :to exit the explorer and vim. If you do not wish to quit vim and continue working with an open file, you have 3 options:Explore nter/Return q

# Open the explorer in a horizontal ( ) or vertical ( ) split and exit the explorer using :Sexplore:Vexplore:q

# Open the explorer in another tabpage using and exit using :Texplore:q

# Open file explorer in your current window and then unload the current buffer and delete it from the buffer list using (buffer delete).:bdel

# :Explore from any vim window shows the file explorer

# NOTE: You can also use the short command to open the file explorer:Ex

# Sometimes I have to repeat same steps on some lines to edit something. I am pretty sure Vim will have some feature that enables me to do this. Am I right?

# 100% Right! You are talking about macros and Vim supports macros. Repeating the last executed command is simple and can accomplish simple repetitive tasks. However, if the text processing is made up of several steps to achieve a result, macros come in handy.

# Consider an example C header file :

# void encrypt\_text(char \*text, int bytes) void decrypt\_text(char \*text, int bytes) void process\_text(char \*text, int bytes) void another\_important\_function(int bytes, double precision)

# Oops! You forgot to put a semicolon at the end of each line and also you just realized that all these functions return an integer error code instead of void.

# The steps you need to perform for making change in one line are:

# Place the cursor at the beginning of the word void

# Press in normal mode to delete the word and type cwvoidint

# Press , move to the end of line using to insert EscShift+a;

# Press and press to return to the beginning of the edited lineEsc^

# Resulting in:

# int encrypt\_text(char \*text, int bytes); void decrypt\_text(char \*text, int bytes) void process\_text(char \*text, int bytes) void another\_important\_function(int bytes, double precision)

# You can just record this sequence of steps and replay it on all 4 lines.

# All you have to do is, before you start the sequence, start recording the macro in any alphabet (let’s say ) by pressing in normal mode. Now your steps are being recorded in . Once you are done with all your steps, just press in normal mode. This will end the recording. To replay these steps, just keep the cursor at the same place where it was placed during macro. Press and we’re done! BOOM! Vim will repeat the same steps for you on that line! To repeat it on multiple lines, you can also use after using command onceaqaaq@a@@@a

# I know Vim is nowhere close to an IDE and I may be having some unreasonable hopes but just a quick question: Remote editing of files possible with Vim?

# If you think of it considering the available resources: [1] Vim [2] openssh-client (Comes installed with most Linux flavors)

# You are in luck my friend! Yes, Vim supports remote editing of files ? Vim just utilizes the secure connection established by scp (secure copy) provided by openssh-client. There are times when you are working with files on multiple remote machines and it’s a waste of time to log into a machine just to edit one single file! You can relax in your current machine if you just know your remote machine credentials and path.

# vim scp://remoteuser@remote\_IP\_or\_hostname/relative/path/of/file

# For example: I need to edit a file on 10.0.18.12 stored in and I have login credentials for , I can access the using:/home/dev-john/project/src/main.cdev-johnmain.c

# $ vim scp://dev-john@10.0.18.12/project/src/main.c

# I can use the relative path because, I can start looking for the file from the home directory of dev-john

# TIP: If you access a remote machine frequently, you can create an ssh config file to create a shortcut for the connection. Create a file with~/.ssh/config

# Host remote-dev-machine Hostname 10.0.18.12 User dev-john IdentityFile ~/.ssh/id\_rsa

# Now you can access your file using:

# $ vim scp://remote-dev-machine/project/src/main.c

# If it’s confusing to remember the relative path and not intuitive, you can also specify it with an alternative:

# $ vim scp://remote-dev-machine/~dev-john/project/src/main.c

# Awesome! I’m already thrilled at the out-of-the-box capabilities of Vim. Looks like you’ve a solution to a lot of common editing problems. Let’s see. I have a file with over 2000 lines and the functions of my interest are located at line 9, line 768 and line 1898. I know I can jump to a line using line number but I’m not so good at remembering those numbers. Got anything for me?

# Hell yeah! What you’re looking for is a local bookmark solution in Vim using letters. All you have to do is :

# Place your cursor on any line at any position

# Press to make sure you’re in normal modeEsc

# Press where is any letter from m{lowercaseletter}{lowercaseletter}a-z

# You just created a local bookmark to navigate in your file

# To view all your bookmarks: Press and enter command mode, type and hit . You’ll see a list of your bookmarks. To visit any bookmark at any time, just press and type . Kaboom! You’ll arrive at the exact same location with cursor where you bookmarked. Example:Esc:marksEnter/ReturnEsc`{lowercaseletter}

# Bookmarks in Vim

# I have created a local bookmark to line 21, column 18 using . If I’m editing something on line 1783, I would just press and type :aEsc`a

# To solve your problem, all you’ve to do is create 3 local bookmarks and quickly jump to them by looking at .:marks

# Problem solved ?

# What if I told you that you can create global bookmarks too?! ? Yes, it is possible to create global bookmarks too! These are equivalent to your windows or GUI shortcuts (or linux soft/hardlinks) except you don’t need to create an actual link. You heard me right! You can literally jump from editing a file in /to another file and line in /from your Vim without exiting ! ?dir1 project/src/

# Fret not, it’s not a big new thing to remember. All you have to do is: Use an uppercase letter instead of lower case letter to create a global bookmark. That’s all! Really! You navigate to the global bookmark using the same process. Example: If you’ve created a bookmark using , all you’ve to do is press and type and BAM! You jump to your global bookmark (Vim remembers the path, so you don’t have to type anything about the path)mPEsc`P

# You can access the global bookmarks in the same way as local : :marks

# :marks mark line col file/text P 53 4 ~/project/src/large-file.c A 11 0 ~/project/README.md

# NOTE: If you are not interested in the cursor position and just want to be there at the beginning of you bookmarked line, use instead of (Use a single quote instead of back tick to be positioned at the beginning of the line)'P`P

# I’ve heard that Vim supports window splitting along with tabs! I understand tabs are great and you get to work with multiple open files at once. But, what about splitting? Why would I want that?

# Scenarios:

# You may want to edit a file by looking at another file simultaneously (May be you are defining a C function by looking at it’s declaration in a header file)

# You may want to edit some portion of a file by looking at the top/bottom portion of the same file simultaneously

# Your work may require you to edit a file by looking at different portions of different files simultaneously

# Vim supports splitting of screen both horizontally and vertically. Even better, you can even browse file system to open a file when you split your screen.

# Here are the available options:

# :split filename - split window horizontally and load filename :vsplit file - vertical split and open file ctrl-w up arrow - move cursor up a window ctrl-w ctrl-w - move cursor to another window (cycle) ctrl-w \_ - maximize current window vertically ctrl-w | - maximize current window horizontally ctrl-w = - make all equal size :sview file - same as split, but readonly :close - close current windowNormal window (top left), :split <file> (top right), :vsplit <file> (bottom)

# Maximizing a window for work:

# The pane needs to be maximized vertically and horizontally to occupy the entire window

# Resizing:

# CTRL-W [N] - Decrease current window height by N (default 1) CTRL-W [N] + Increase current window height by N (default 1) CTRL-W [N] < Decrease current window width by N (default 1) CTRL-W [N} > Increase current window width by N (default 1)

# Is there a way to use file explorer while I split panes? (I can’t remember and type the file names always!)

# Of course, all you have to do is type : for horizontal file explorer and for vertical file explorer. You can also use to open the file explorer on right side (instead of default left):Sexplore:Vexplore:Vexplore!

# Again, all this works \*without any extra plugins\* ?

# I am in the middle of editing some code and I quickly need to run a shell command. Should I save my work, exit Vim and run my commands? I bet there is a better way out with Vim

# You bet! Vim just doesn’t want you to leave Vim and wants you to continue focussing on your work. Hence the option to execute shell commands from within your Vim. Don’t worry, all your unsaved work is not discarded, you just execute your command and BAM! you are back in your unsaved/saved file safely!

# Let’s say you are in the middle of a coding session and you quickly need to head out to take a look at man page of file operations because you forgot the signature! You don’t have to save your work, exit Vim and then check man pages or you don’t have to open another tab just for the man page. You can issue the command from right within the Vim editor.

# [DEMO] Unix commands from VIM VIM allows users to execute shell commands from within VIM without exiting. All you have to do is enter the command…asciinema.org

# Left to Right (Execute shell commands from Vim and jump back to editor)

# Guess what! Prepare to be amazed. Vim also supports command from within your file! All you have to do is navigate to a directory with . Open any file (Could be your source code) and make all the changes and save it. Wait, there’s no need to exit to see the compilation result. You can trigger your make build from right within Vim:makeMakefile

# [DEMO] Trigger make builds from vim VIM allows users to trigger make builds without exiting VIM. All we have to do is enter the command mode and type :makeasciinema.org

# Triggering make builds from Vim

# Similarly you can build other targets in your Makefile!

# Example: Build directory clean up

# Cleaning directory using make command from VIM

# I hope these cool features will help you to use Vim more productively.

# Your feedback is always welcome.

# Feel free to comment, criticize or applaud ?

# References:

# http://www.openvim.com/tutorial.html

# https://linuxconfig.org/vim-tutorial

# ftp://ftp.vim.org/pub/vim/doc/book/vimbook-OPL.pdf

# http://vim.wikia.com/wiki/Tutorial

# http://www.viemu.com/a-why-vi-vim.html

# http://robertames.com/files/vim-editing.html

# https://www.youtube.com/watch?v=wlR5gYd6um0