Popular Vim Commands - Comprehensive Vim Cheat Sheet

By Cody Arsenault   Updated on July 13, 2020

[Vim](#) was made available in 1991 and is a free, open source software. Available both as a command line interface and as a standalone program with a GUI, Vim is a text editor that is a modal version of the [vi editor](#) created for Unix in the 1970s; Vim stands for vi improved. While it was designed with [Unix](#) in mind, versions of it are available for most operating systems and Vim is also available for Android and iOS smartphones.

While you may be familiar with the concept of a text editor, the modal part may throw you. A modal editor is one that allows you to **edit text in different modes**, and in the case of Vim, the mode determines what the alphanumeric keys on your keyboard do and how Vim editor commands work.

For example, in insert mode, your keyboard behaves normally, so what you type in is what you see, just **like with a standard text editor**. However, if you switch to command mode, the letters on your keyboard will allow you use Vim commands to **move within the text**. If you play video games that use the left-hand keys on the keyboard to move your character around, this concept is probably familiar to you.

To open a file using Vim you can use the following command (simply replace filename.css with your actual file name).

```
vim filename.css
```

The idea behind having a modal text editor is that it allows you to write and edit text, including code, without requiring your hands leave the keyboard. Vim isn't for everyone, and it requires you to learn a variety of Vim editor commands to get the most out of it. That said, many people who have started using it and become comfortable with it won't even consider another editor. The image below shows an exampe of what a css file looks like when viewed using Vim.

This text editor is particularly well-suited for people who are programmers, coders, system administrators or individuals looking for a streamlined way to edit text. The editor allows you to edit text in multiple windows, which can be helpful to programmers and editors alike. If you're interested in giving Vim a shot, the following is a basic explanation of Vim modes and a list of frequently used Vim commands, along with a definition of what they do.

Vim modes

There are some arguments as to how many modes that Vim has, but the modes you're most likely to use are command mode and insert mode. These modes will allow you to do just about anything you need, including creating your document, saving your document and doing advanced editing, including taking advantage of search and replace functions.

Command mode

This is the default mode that you'll be in once you open Vim. If you're in a different mode and want to go back to command mode, just hit the Escape key. This mode allows you to use Vim commands and move through your document. From command mode, you can also use last-line commands, which generally start with the use of a colon. For example, `:w` saves your file and `:q` allows you to exit Vim.

Insert mode

This mode allows you to enter text into your document. You can enter insert mode by pressing the `i` key. Keep in mind that to save your document, you'll need to go **back to command mode** since only text input is allowed in this mode.

Installing Vim

There are a few ways to install Vim and the one you end up using will sometimes depend on which system you're using.

Install Vim using Git:

According to Vim themselves, install Vim via Git is the simplest and most efficient method. Simply use the following commands:

```
git clone https://github.com/vim/vim.git
cd vim/src
make
```

If you already have Vim installed but want to update to the latest version, you may need to use additional Git commands which can be found [here](#).

Install Vim on Ubuntu/Debian:

If you're using Ubuntu or Debian use `apt-get` to install Vim, like so:

```
sudo apt-get install vim
```

Install Vim on CentOS/Fedora:

If you're using CentOS or Fedora, use yum to install Vim:

```
sudo yum install vim
```

If you want a more advanced set of features on CentOS/Fedora, you'll need to install `vim-enhanced`, to do this, run the following command instead:

```
sudo yum install -y vim-enhanced
```

Vim commands

The following is a list of frequently used commands and what they do. Many of the commands can be made to repeat by adding a number to the command. This is not an exhaustive list because more advanced commands, such as how to use multiple buffers, are not included. However, just about all of the basic commands for opening, editing and saving documents are included as well as commands that enable you to find and replace text and work with multiple documents.

1. Basic Vim commands

The most simple commands allow you to open and close documents as well as saving them. As with most other text editors, there are protections in place to help you avoid exiting the editor without having saved what you're working on.

`:help [keyword]` - Performs a search of help documentation for whatever keyword you enter

`:e [file]` - Opens a file, where [file] is the name of the file you want opened

`:w` - Saves the file you are working on

`:w [filename]` - Allows you to save your file with the name you've defined

`:wq` - Save your file and close Vim

`:q!` - Quit without first saving the file you were working on

2. Vim commands for movement

When using movement commands, you can put a number in front of them to make Vim complete a command multiple times. For example, `5h` will move your cursor five spaces to the left, and `90j` will put your cursor at the beginning of the 90th line down from where your cursor currently is.

`h` - Moves the cursor to the left

`l` - Moves the cursor to the right

`j` - Move

`k` - Move

`H` - Puts the cursor at the top of the screen

`M` - Puts the cursor in the middle of the screen

`L` - Puts the cursor at the bottom of the screen

`w` - Puts the cursor at the start of the next word

`b` - Puts the cursor at the start of the previous word

`e` - Puts the cursor at the end of a word

`0` - Places the cursor at the beginning of a line

`$` - Places the cursor at the end of a line

`)` - Takes you to the start of the next sentence

`(` - Takes you to the start of the previous sentence

`}` - Takes you to the start of the next paragraph or block of text

`{` - Takes you to the start of the previous paragraph or block of text

`Ctrl + f` - Takes you one page forward

`Ctrl + b` - Takes you one page back

`gg` - Places the cursor at the start of the file

`G` - Places the cursor at the end of the file

`#` - Where # is the number of a line, this command takes you to the line specified

3. Vim commands for editing

Those who use Vim tend to use the term "yank" where most people would use the terms copy and paste. Therefore, the command for copying a word is `yw`, which stands for yank word, and the command for pasting whatever has been copied is `p`, meaning put. If you look up additional commands in the future, it can be confusing if you don't know what yank and put mean when using Vim.

You also have two options for how to select text. You can either use commands like `dd`, which deletes a single line, and `yy`, which copies a single line, or you can highlight text and then copy it to the unnamed register. The paste commands work the same whether you've highlighted text or used a command to automatically copy it.

As with movement commands, putting a number in front of the command can increase the number of times a task is completed. For instance, putting a number in front of `yy` will increase the number of lines copied, so `5yy` will copy five lines.

`yy` - Copies a line

`yw` - Copies a word

`y$` - Copies from where your cursor is to the end of a line

`v` - Highlight one character at a time using arrow buttons or the h, k, j, l buttons

`V` - Highlights one line, and movement keys can allow you to highlight additional lines

`p` - Paste whatever has been copied to the unnamed register

`d` - Deletes highlighted text

`dd` - Deletes a line of text

`dw` - Deletes a word

`D` - Deletes everything from where your cursor is to the end of the line

`d0` - Deletes everything from where your cursor is to the beginning of the line

`dgg` - De

KeyCDN uses cookies to make its website easier to use. Learn more about cookies.

`dG` - Dele

`x` - Deletes a single character

`u` - Undo the last operation; u# allows you to undo multiple actions

`Ctrl + r` - Redo the last undo

`.` - Repeats the last action

4. Vim commands for searching text

Like many other text editors, Vim allows you to search your text and find and replace text within your document. If you opt to replace multiple instances of the same keyword or phrase, you can set Vim up to require or not require you to confirm each replacement depending on how you put in the command.

`/[keyword]` - Searches for text in the document where keyword is whatever keyword, phrase or string of characters you're looking for

`?[keyword]` - Searches previous text for your keyword, phrase or character string

`n` - Searches your text again in whatever direction your last search was

`N` - Searches your text again in the opposite direction

`:%s/[pattern]/[replacement]/g` - This replaces all occurrences of a pattern without confirming each one

`:%s/[pattern]/[replacement]/gc` - Replaces all occurrences of a pattern and confirms each one

5. Vim commands for working with multiple files

You can also edit more than one text file at a time. Vim gives you the ability to either split your screen to show more than one file at a time or you can switch back and forth between documents. As with other functions, commands make going between documents or buffers, as they're referred to with Vim, as simple as a few keystrokes.

`:bn` - Switch to next buffer

`:bp` - Switch to previous buffer

`:bd` - Close a buffer

`:sp [filename]` - Opens a new file and splits your screen horizontally to show more than one buffer

`:vsp [filename]` - Opens a new file and splits your screen vertically to show more than one buffer

`:ls` - Lists all open buffers

`Ctrl + ws` - Split windows horizontally

`Ctrl + wv` - Split windows vertically

`Ctrl + ww` - Switch between windows

`Ctrl + wq` - Quit a window

`Ctrl + wh` - Moves your cursor to the window to the left

`Ctrl + wl` - Moves your cursor to the window to the right

`Ctrl + wj` - Moves your cursor to the window below the one you're in

`Ctrl + wk` - Moves your cursor to the window above the one you're in

6. Marking text (visual mode)

Visual mode allows you to select a block of text in Vim. Once a block of text is selected you can use visual commands to perform actions on the selected text such as deleting it, copying it, etc.

`v` - starts visual mode, you can then select a range of text, and run a command

`V` - starts

`Ctrl + v` - starts visual block mode (selects columns)

`ab` - a block with `()`

`aB` - a block with `{}`

`ib` - inner block with `()`

`iB` - inner block with `{}`

`aw` - mark a word

`Esc` - exit visual mode

Once you've selected a particular range of text, you can then run a command on that text such as the following:

`d` - delete marked text

`y` - yank (copy) marked text

`>` - shift text right

`<` - shift text left

`~` - swap case (upper or lower)

7. Tab pages

Just like any browser, you can also use tabs within Vim. This makes it incredibly easy to switch between multiple files while you're making some code changes instead of working in one single file, closing it, and opening a new one. Below are some useful Vim commands for using tab pages:

`:tabedit file` - opens a new tab and will take you to edit "file"

`gt` - move to the next tab

`gT` - move to the previous tab

`#gt` - move to a specific tab number (e.g. 2gt takes you to the second tab)

`:tabs` - list all open tabs

`:tabclose` - close a single tab

Simple Vim workflow example

If you haven't had a chance to play around with Vim much yet, you might be wondering what a simple workflow looks like when using it. It's relatively simple:

1.   Open a new or existing file with `vim filename`.

2.   Type `i` to switch into insert mode so that you can start editing the file.

3.   Enter or modify the text with your file.

4.   Once you're done, press the escape key `Esc` to get out of insert mode and back to command mode.

5.   Type `:wq` to save and exit your file.

Of course, there is so much more you can do with Vim, however as a beginner, the above steps are what a simple Vim workflow looks like.

Summary

don't have to worry too much about accidentally deleting large swathes of text without being able to recover them since you can use the undo command multiple times.

While not for everyone, functionality like being able to work on more than one document at a time in windowed screens and the ability to do **major editing without a mouse** is what makes Vim so popular. You can [download](#) the editor for free, and there are a variety of plugins and extensions that can improve its functionality and add additional Vim commands.

TAGS   `tools`   SHARE

A Collection of the Best Text Editors in 2018

All Windows and Mac machines have basic text editors, but TextEdit and Notepad aren't sufficient for...

100+ Awesome Web Development Tools and Resources

The best and worst thing about being a web developer is that the web is constantly changing. While...

SUPERCHARGE YOUR CONTENT DELIVERY 🚀

Try KeyCDN with a free 14 day trial, no credit card required.

Get started

Comments

Bill   4 years ago   Reply

Nice summary to get started with Vim, but I have a few corrections for you (so we don't confuse people trying to use vim for the first time):

Start Vim using `vim filename` instead of `vi` . Usually, `vi` will run vim with vi compatibility mode, which is less user friendly.

Open a file using `:e filename` instead of `:o filename` .

`90j` would move the cursor down 90 lines, not 9. For that, you would use `9j` .

Yanking text does not necessarily go to the clipboard. When you yank normally with `yw` or similar, it goes into the "unnamed register", ready to be put with the next `p` command. To copy to the clipboard (so that you can paste it in another program), specify the `+` register as in `+yy` and paste from the clipboard with `+p` (See `:help` registers).

Deleted text also goes into the unnamed register by default (if you don't specify a register for it). So you ca...

R...   ...urrences of a pattern without confirming each one (`g` for all occurrences in a line):

```
:%s/pattern/replacement/g
```

Replace all occurrences of a pattern but confirm each one:

```
:%s/pattern/replacement/gc
```

Cody Arsenault 🔵 4 years ago

Hey Bill, thanks for catching that. We have modified the article to reflect those changes.

Henry Schaffer   1 year ago

"and `yw` , which copies a single line" a single word?

Corey 🔵 1 year ago

Thanks for letting us know about this mistake. We've updated the article accordingly.

Comment policy: Comments are welcomed and encouraged. However, all comments are manually moderated and those deemed to be spam or solely promotional in nature will be deleted.

Comment

bold `code` ```block```

Name

Email

Submit

KeyCDN uses cookies to make its website easier to use. Learn more about cookies.   ✕

PRODUCT   COMPANY   SUPPORT

© 2021 proinity LLC
Made in Switzerland

Terms Privacy GDPR

PRODUCT

Features

Network

Pricing

API

COMPANY

About

Blog

Contact

Referrals

Careers

SUPPORT

Knowledge Base

Network Status

Open Source

FAQ

Tools