Quick Vim Reference

Kurt Schmidt

April 2022

Contents

Foreword

This document is a quick reference, for things you've learned, and a quick survey of some features you might wish to learn.

You must go through the Vim tutorial first

```
$ vimtutor
```

, and keep practising. Get comfortable, and add new skills you find useful.

Crutches

While learning Vim I *really* discourage the use of the mouse, and of the arrow keys. Navigation, selection, will take a little bit of practice, but, this effort is well worth it. There is much more available than the four simple motions, but you'll never get there if you stop at the arrow keys.

The mouse is great for editing photos, playing solitaire, but moving the cursor, selecting text, we have many fine tools. Similarly, the mouse will slow you down.

1 Modes

Vim typically starts in *Normal Mode*. You can navigate around your document, save it, go into insert mode, etc. Usually the `<ESC>` key will bring you back to Normal Mode.

Some of the other modes:

Insert Mode Where you do all your typing. See Insert Mode Section
Visual Mode Use motions to visually select text. See Visual Mode Section
Replace Mode Really just an overstrike mode, as best I can tell.

Use `<ESC>` to get back to *Normal Mode*, same as *Insert*

Command Mode For things that start with a `:`, `/`, or `?`. Shows up at the bottom of the window. See also the Command History Window

1.1 Getting Lost

`<ESC>` will very often cancel the current command, *Normal Mode* key sequence, etc., or, return you to *Normal Mode*.

If you find yourself in `ex` (line editor) mode, try `:vim` .

2 Startup

Some useful features can be invoked at startup, either through a command-line option, or installed alias:

`view` Starts vim in read-only mode. Same as `vi -R`
`vimdiff` Starts you in a diff mode, documents are side-by-side. Make your terminal wide. `vim -d`
`gvim gview` The GUI version (if installed). `vim -g`

2.1 Some Handy Settings

The `set` command is used to see or change many vim settings. There are 3 types:

- number
- string
- toggle

To see summary settings:

```
:set
```

To see all:

```
:set all
```

To view a particular setting:

```
:set filetype?
```

2.1.1 Setting Number/String Options

```
:set ft=python
:set tabstop=3
:set verbosefile=
```

2.1.2 Setting a Toggle Option

Turn it off by preceding the option with "no":

```
:set hlsearch
:set nohlsearch
:set number
:set nonumber
```

Flip the current value by preceding option with "inv":

```
:set invpaste
:set invlist
```

This is handy for mapping a setting to a key.

2.2 Options Beginners Might Consider

Note, you often needn't spell out the entire option. I'll include common shortcuts that I remember. Examples will show settings I recommend.

`[no]compatible cp` Don't behave as the original VI editor

```
set nocompatible
```

`hidden hid` Allows you to switch to another buffer w/out saving current one

```
set hidden
```

`background bg` By default, color schemes assume a light background. If you use a dark terminal, then:

```
set bg=dark
```

`showcmd sc` Toggle. Show partial commands (what you're typing in *Normal Mode*) at bottom

`ruler ru` Toggle. Shows row and column in lower right corner

`hlsearch hls` Toggle. Highlights matches from a search. (Use `:set nohl` to turn it off, or see below for a binding I use)

`incsearch is` Toggle. Show matches as regex is being typed

`wildmenu` Toggle. Better command-line completion (try the `<Tab>` key)

`showmode smd` Toggle. Displays the vim mode, bottom left (I think)

`showmatch sm` Toggle. Shows partner when cursor is placed on a bracket (parenthesis, etc.).

`textwidth tw` Number. Width of buffer. Where VIM will insert characters (on whitespace), as you type (format cmd, etc.)

`tabstop ts` Number. The number of columns a tab displays. (Does *not* affect the buffer, only the appearance.)

```
set tabstop=3
set shiftwidth=3
```

`shiftwidth sw` Number. Number of spaces to use for indent, shift (`<<`, `>>`, etc.). Set to 0 to use `tabstop` value

`number nu` Toggle. Turns on line numbers.

I find I don't really use this. I have the ruler, and `<n>G` will take me to a particular line.

`relativenumber ru` Toggle. Give it a try. I don't use it, but, some might find it handy, especially when `nu` is also set

`wrap` Toggle. Display only. Long lines are wrapped, rather than disappearing off to the right. Doesn't modify the buffer.

`showbreak sb` String. Prefix for wrapped lines.

2.3 `.vimrc` – Vim Config File

You can put your customisations in a config file. Vim will read `~/.vimrc` upon startup. (GVim will instead read `~/.gvimrc`, if present.)

I have created a simple starter `.vimrc` file for newbies to use. Users on the department machines should copy it from my `Public` directory. Or, you can look at it here.

Note, the double-quote, `"`, introduces a line comment in this file

Other than the options discussed above, here are some other settings you might find helpful:

`syntax on` Turns on syntax highlighting, autoindenting, etc. (as appropriate)

`map Y y$` Behave like `y$`, rather than `yy`

`D` and `C` work in this way. The original Vi was inconsistent, here.

`nnoremap <C-L> :nohl<CR><C-L>` Ctrl-L now turns off search highlighting, then redraws the screen

`map <F1> <Esc>`

`imap <F1> <Esc>` I got tired of hitting the `<F1>` (help) key, when reaching for the `<ESC>` key

[Older Lenovos had `<ESC>` *above* the `<F1>` key.]

`set pastetoggle=<F2>` Use `<F2>` to toggle between `paste` and `nopaste`. Helpful if pasting into a buffer from the terminal or desktop clipboard

`map <F3> :set invlist<CR>` Use `<F3>` to toggle between `list`, which shows you newlines, tabs, and other characters, and `nolist`

`map <Leader>m :wall<CR>q:?^make<CR><CR>` Type `\m` to save any dirty buffers, then re-execute the most recent `make` command

3 Administrative Commands (Normal Mode)

`<ESC>`	Cancels current command, or, returns you to *Normal Mode*
`:e fn`	Edit `fn` in a new buffer
`:q`	Close current window (if no changes)
`:q!`	Close current window, dammit!
`:w [fn]`	Write buffer to *fn*, if provided; otherwise, use current buffer name
`:w! [fn]`	Write buffer, possibly overwriting target
`:wq`	Write and quit
`ZZ`	Write and quit
`:x`	Write (only if dirty) and quit

3.1 Background

If you use a terminal with a dark background:

```
:set bg=dark
```

3.2 Redrawing the Window

`ctrl-L`	Redraw (refresh) the screen
`z<CR>`	Redraw, current line to top of window
`z.<CR>`	Redraw, current line to center of window
`z-<CR>`	Redraw, current line to bottom of window

4 Motion

You can move the cursor around with the following keystrokes while in *Command Mode*. Most (all?) of the following can be preceded by a count. Default is 1.

Simple motions

`h`	Left
`j`	Down
`k`	Up
`l`	Right
`0`	Beginning of line
`$`	End of line
`H`	Top of window
`M`	Middle line of window
`L`	Bottom of window
`gg`	1st line of file
n`G`	*n*th (last) line of file
`ctrl-f`	Forward a screenful
`ctrl-b`	Backward a screenful

Please don't use the arrow keys. They're slower than the 4 basic motions, right on the home row. Get good at the basic motions, then add other motions. Don't just dead-end at arrow keys. That's Notepad.

Motions over objects

`w` , `W`	Beginning of next (big) word
`e` , `E`	End of next (big) word
`b` , `B`	Back. Beginning of previous (big) word
`(` , `)`	Beginning (end) of this or previous (next) sentence
`{` , `}`	Beginning (end) of this or previous (next) paragraph

4.1 Line Numbers

- To see your line, column and apparent column in lower right:

  `:set ruler`

- To go to a particular line number ***nn***:

  ***nn*`G`**

- If you really want to see line numbers:

  `:set number`

To turn it off:

```
:set nonumber
```

Also, try this:

```
:set relativenumber
:set nornu
```

4.2 Searching

`/`*re*	Forward incremental regex search
`?`*re*	Backward incremental regex search
`n N`	Go to next (previous) match, same direction
`#`	Match previous string under cursor
`*`	Match next string under cursor
`f`*char*	Advance cursor to next *char* on current line
`F`*char*	Move cursor left to previous *char* on current line
`t`*char*	Advance cursor to 1 column before next *char* on current line
`T`*char*	Move cursor left to 1 column after previous *char* on current line
`;`	Repeat the previous `f`, `F`, `t`, or `T` command
`''`	Return cursor to previous line

Note, `/` and `?` are *Command-line Mode* commands; the cursor will jump to the bottom of the window.

5 Insert Mode

There are various ways to get into *Insert Mode* from *Comand Mode*:

`i`	Insert at (prior to) cursor
`I`	Insert at first (non-white) character in the line
`a`	Append at next column after cursor
`A`	Append at end of current line
`o`	Open new line below cursor
`O`	Open new line above cursor

Each of these can be preceded by a count. You type some text. When you hit `<ESC>`, the edit is repeated.

5.1 Inserting Literal Character `i_ctrl-v`

Use ctrl-v in *Insert Mode* (`i_ctrl-v`), or *Command Mode*, to quote a literal character, like, e.g., Backspace.

Handy if you want the replacement string in a substitute command to have a Newline in it:

```
%s/, /^M/g
```

You can't type `^M` as 2 characters; instead, you hit `ctrl-v` while typing, then the `Enter` key.

5.2 Inserting Special Characters

5.2.1 Character Codes (in *Insert Mode* and *Command Mode*)

Use `ctrl-v` followed by one of the prefixes to enter a character by its codepoint.

Prefix	Mode	Digits	Max Val
(none)	decimal	3	255
o or O	octal	3	377
x or X	hexadecimal	2	ff
u	hexadecimal	4	ffff
U	hexadecimal	8	7fffffff

To insert a vertical tab character (while in *Insert* or *Command* mode) (w/out spaces)

```
ctrl-v x0B
ctrl-v 011
ctrl-v o013
```

To insert a UTF character, say, lower-case phi:

```
ctrl-v u03c6
```

5.2.2 Digraphs

Vim maps many 2-character sequences to special characters.

While typing in *Insert Mode*, use `ctrl-k`, then the 2-key combo.

- To type the degree mark:

  `ctrl-k DG`

- To type o with an accent:

  `ctrl-k o'`

- To type capital lambda

  `ctrl-k L*`

To see a list of digraphs:

```
:help dig
```

, then page down, or search.

6 Edit Operations

6.1 Undo / Redo / Repeat

`u`	Undo
`ctrl-r`	Redo
`.`	Repeat the last edit

Unlike in the original Vi, we have an undo (and a redo) stack, so, you can undo all the way back to the loaded buffer. (See variable `undolevels`.)

6.2 Deleting Text

These commands are available in *Normal Mode*, and can be preceded by a count:

`x`	Delete character under the cursor
`d`*motion*	Delete text from cursor to point of *motion*
`dd`	Delete current (and following, if *count* > 1) lines
`D`	Delete from cursor to end of line

6.3 Pasting Text (Put)

To paste deleted or yanked text into a document from *Normal Mode*

`P`	Paste (before cursor)
`p`	Paste (after cursor)

`xp` is a quick, handy way to transpose two characters

6.3.1 Pasting from Outside Vim

Note about pasting from an external clipboard: Things like autoindent, etc., may cause problems if you paste already formatted text or code into a document. Vim supplies a *paste* setting, which turns off much of the autoformatting:

```
:set paste
```

Remember to cancel this setting for regular editing when done:

```
:set nopaste
```

With this setting in my `.vimrc` I just hit a key to toggle on and off:

```
  " Use <F2> to toggle between 'paste' and 'nopaste'
set pastetoggle=<F2>
```

6.3.2 Read a File, or Command Output, into Buffer

I often find it handy to skip the clipboard, to simply read contents from a file `<fn>` right into the buffer. From *Normal Mode*:

```
:r <file>
```

, will insert contents starting at line beneath the cursor.

To place output from a command into buffer, e.g.:

```
:r! grep Waldo *.tex
```

```
:r! date -R
```

```
:r! head -n12 <file>
```

6.3.3 Open Source in New Buffer

Or, open the file in a new buffer, use Vim to yank text from there to another buffer.

6.4 Copying Text (Yank)

y*motion*	Yank (copy) word, sentence, paragraph, etc.
*cnt*yy	Yank (copy) *cnt* line(s)
Y	Yank (copy) *cnt* lines
y$	Yank (copy) to end of line

6.5 Changing Text

r	Replace single character under the cursor with another. Leaves you in *Normal Mode*
R	Puts you in *Replace Mode*
c	Change. Used much like d, above, but leaves you in input mode
s	Substitute. Delete char under cursor, leaves you in *Insert Mode*
~	Changes case of character under cursor. Can be used w/a count, or highlighed text

6.5.1 Changing Case

Each of these is followed by a motion or text object

gU	Change text to upper-case
gu	Change text to lower-case
g?	Performs Rot13 encoding (decoding)

6.6 Registers

Any of the lower-case letters can be used to store deleted or yanked (copied) text, when preceded by a `"` . Simply precede any delete, yank, or put operation with a register specifier:

```
"f2yy  # copy current line and line below into register f
# move elsewhere
"fp  # paste the 2 lines after the cursor
"fP  # paste the 2 lines before the cursor
```

You can use *Visual Mode* when yanking or deleting text.

The numbered registers contain a history of your yanks. Vim maintains other registers, too.

`:reg` Will show you all the registers and their contents

Vim uses the numbered registers, 0-9, and a few others, to automatically store yanked/deleted text and other things.

```
"%p  # paste filename of current buffer into the buffer
```

7 Formatting

7.1 Indent/Exdent

`>>`	indent line
`<<`	exdent line

- As with many commands, can be preceded by a count
- Or, use *Visual Mode*, then a single `<` or `>` to move whole section
- Or, `<` followed by a *motion* or a text object

7.2 Joining 2 lines

Use `J` to join the current line with the line below.

7.3 Re-formatting

You can ask Vim to reformat, e.g., a paragraph, or a function

- Use *Visual Mode*, then `gp`
- Or, `gp<motion>`
  - `gp}` – reformat to end of paragraph
- Inner-objects
  - `gpip` – reformats entire paragraph cursor is on
  - (See Text Objects, below)

8 Vim Text Objects

These *Normal Mode* keystrokes are sorta like motions, but, they don't generally use the location of the cursor as a mark. E.g., if the cursor is anywhere in a word, we can delete the word, or the sentence, or reformat the paragraph.

Each object can be selected with (`a`) or without (`i`) surrounding whitespace (sorta).

`aw`	A word
`iw`	An inner word
`aW iW`	A (inner) big word
`as is`	A (inner) sentence
`ap ip`	A (inner) paragraph

8.1 Source Code Objects

I don't know that the behavior is dependent on filetype, I've not played, but, these objects make more sense in the context of code, some of them language-dependent.

8.1.1 Strings

`a" i"`	A (inner) double-quoted string
`a' i'`	A (inner) single-quoted string
`a` i``	A (inner) back-quoted string

8.1.2 Other Bracketed Expressions

`a) i)`	A (inner) parenthesized block
`a] i]`	A (inner) bracketed block
`a} i}`	A (inner) brace block
`at it`	A (inner) XML-like markup tag block
`ai ii`	A (inner) indented block

9 Marks, Visual Mode

Marks can be used for navigation, and for commands that take ranges. 2 marks can delimit a selection of text, for a variety of actions, including deleting/yanking, reformatting, write (`:w`), substitute (`:s`) and filter (`:!`) commands.

- To set a mark: `mc` , where `c` is any lower-case letter
- To go to that line: `'c` , where `c` is the mark

9.1 Visual Mode

Visual Mode allows you to graphically select text, sets two marks for you: `` `< `` and `` `> ``. While in *Visual Mode*, *Normal Mode* commands (indent, delete, yank, etc.) will act on this text, and *Command Mode* commands (`:w`,`:s`, etc.) will appear w/the range marks already supplied.

To highligh text, go to one end, then hit:

`v`	visual
`V`	visual, line mode
`ctrl-v`	visual, block mode

, the use motion keys (or a search) to get to the other border.

`<ESC>` will cancel the current *Visual Mode*, return you to *Normal Mode*.

Example:

- Use *Visual Mode* to highlight several lines of text
- Type

  `:`

  - Note the bottom line of your screen. The special marks are already filled in
- Continue to type

  `w ~/tmp/blah.sav`

  - Note, you've save the highlighted text to a new file
  - Remember to delete the file
- Or, hit `<ESC>` to cancel the command

10 Splitting the Window

[I don't quite have the jargon correct, so, don't get too formal.]

When you call for `:help`, e.g., the window splits into 2 panes.

Other ways to split the window / open a new one (from *Normal Mode*) :

`:spl`	Splits window horizontally
`:spl <fn>`	Opens new window horizontally, opens file
`:vspl`	Splits window vertically
`:vspl <fn>`	Opens new window vertically, opens file
`ctrl-w n`	Opens new window horizontally
`ctrl-w v`	Splits window vertically
`:term`	Opens new window above, starts a shell instance
`q:`	Opens the Command History Window

10.1 Managing Windows

Again, many of these can be preceded by a count, which often determines window's size (height or width)

`:q[!]`	Quit current window [dammit!]
`ctrl-w c`	Close current window
`ctrl-w o`	Make current window only (close others)
`ctrl-w +/-`	Increase/Decrease height of window
`ctrl-w </>`	Increase/Decrease width of window
`ctrl-w H/L`	Move window left/right
`ctrl-w J/K`	Move window down/up

10.2 Moving Between Windows

`ctrl-w h`	Go n windows left
`ctrl-w l`	Go n windows right
`ctrl-w j`	Go n windows down
`ctrl-w k`	Go n windows up

10.3 Command History Window

This lets you search back through your command history, perhaps edit, and re-execute commands. `q:` will open a small (7-line) window, which you can navigate in the normal way.

Hit `[ENTER]` to execute the command the cursor is on.

`:q` or `ctrl-C [ENTER]` to cancel.

11 Buffers

11.1 Buffers v tabs

In short, Vim "tabs" should be called "layouts". Using buffers as you might use tabs in a different program is the intention.

This is a discussion of "Tab Madness".

11.2 Using Multiple Buffers

While editing a file you can use `:e <fn>` to open another file into a new buffer.

11.3 Working With Buffers

`:ls`	List (and index) buffers
`:b <n>`	Switch to buffer `<n>`
`:bd`	Delete (close) current buffer (if no changes)
`ctrl-6`	Switch between 2 most recent buffers

I have this handy binding in my `.vimrc`, so I hit `gb`, the buffers are listed, then I just type the new buffer number:

```
nnoremap gb :ls<CR>:b<Space>
```

Careful, `:q` will try to quit all the buffers.

12 Command Mode

Largely, everything can be done here. The original Vi was built on an older line-editor, Ex. You couldn't move the cursor around the document, so, everything was done from this command line. It is here you save the buffer, open new ones, quit, change settings, delete or replace text, delete lines, insert or append, run the buffer against an external filter, etc.

You already know some:

```
:w[!]
:q[!]
:wq
:x
:r
:set
```

12.1 Some Really Helpful Commands

12.1.1 Delete

Without a range, deletes the current line:

```
:d
```

12.1.2 Substitute

Does a REGEX search and replace on first occurrence of current line:

```
:s/white chocolate/not chocolate/
```

To replace *every* occurrence on the line:

```
:s/this/that/g
```

12.1.3 Global

Allows us to apply an Ex command only to lines that match some regular expression.

To delete empty lines:

```
:g/^$/d
```

To delete blank lines (maybe contain whitespace):

```
:g/^\s*$/d
```

12.1.3.1 Inverse Use `v` to apply a command to all lines that *don't* match.

To delete any line which does *not* contain chocolate:

```
:v/[Cc]hocolate/d
```

12.2 Ranges

Many of these commands which act on the buffer can be preceded by a range (the default is simply the current line).

Ranges can be delimited by:

- Line numbers
  - `.` is the current line
  - `$` can be used for the last line
  - `%` is a range, the entire file (short for `1,$`)
- Marks
- Regular Expressions
  - REs are delimited by `/`

Some examples:

- To delete lines 1-12:

  ```
  :1,12 d
  ```

- To swap first occurrence of Merry for Pippin on each line, from current line to end:

  ```
  :.,$s/Merry/Pippin/
  ```

- To swap every occurrence of Merry for Pippin on each line, from current line to end:

  ```
  :.,$s/Merry/Pippin/g
  ```

- To save the HTML header section to another file:

  ```
  :/<head>/,/<\/head>/ w ~/public_html/general_header.html
  ```

  - Note, had to escape that inner `/`

- Make a fold out of the highlighted code (*Visual Mode*):

  ```
  :`<,`>fo
  ```

- Given 2 marks `a` and `b`, `a` precedes `b`, comment out all lines in range (inclusive)

  ```
  :`a,`bs/^/# /
  ```

We can mix range element types.

- Delete lines from beginning of file until the string `<body>` is found (inclusive):

  ```
  :1,/<body>/! d
  ```

12.2.1 Offsets

We can add or subtract lines from range indicators. E.g., to delete starting 2 lines after the cursor up to, but not including, the next occurrence of "Waldo":

```
:.+2,/Waldo/-1 d
```

12.3 External Filters

You can apply any external program which reads `stdin` and writes to `stdout` (filter) to the buffer.

To sort all lines of the file:

```
:%! sort
```

Use awk to put line numbers in the file:

```
:%! awk '{printf( "\%3s \%s\n", NR, $0 )}'
```

- We had to escape the `%` from Vim
  - Vim will replace `%` with the filename of the curent buffer

13 Folds

The easiest way to create a fold is to use visual mode. Highlight section, then:

`:fo`

`zo`	Open fold under cursor
`zc`	Close fold under cursor
`za`	Alternate (open closed fold, close open fold)

Epilogue

I didn't touch a good deal of what Vim has to offer. This document is intended for those fairly new to Vim. It contains some information immediately useful, and some that might provide anchors for further investigation, a preview of things yet to learn.

Become proficient w/the basics, then add a feature to your reportoire, take it for a test drive. If it improves your workflow, great. If not, that's fine. Either way, pick the next feature that looks appealing, take it for a spin. There's some effort involved here, but, on balance, the effort spent will work for you, make you more productive.

Again, there's much to Vim. If you feel as though the editor should do something, it likely does. Or, perhaps, you can record macros or even write a function to do it.

Holding Area

This is a collection of possibly handy tidbits which might get inserted into the document above, at some point. Just a temp area to hold things I run across

13.1 Find Character Code

Place cursor on a character, hit `ga` (in *Normal Mode*), see the code for the charcter under the cursor.

13.2 Searching for Non-ASCII characters

Remember that `/` in normal mode begins a forward search:

```
/[^\x00-\x7F]
```

Note, the `\x` form doesn't work outside the `[]`. See Inserting Special Characters.

13.3 Insert Into Command Line

These work for you in *Command Mode*.

13.3.1 Insert Contents of Register

Use `ctrl-R`, then choose a register:

Register	Description
`"`	Unnamed register, containing the text of the last delete or yank
`%`	Current file name
`#`	Alternate file name
`*`	Clipboard contents (X11: primary selection)
`+`	Clipboard contents
`/`	Last search pattern
`:`	Last command-line
`-`	Last small (less than a line) delete
`.`	Last inserted text

13.3.2 Inserting Object Under Cursor

Again, use `ctrl-R`, followed by:

`CTRL-F`	Filename under the cursor
`CTRL-P`	Filename under the cursor, expanded with 'path' as in |gf|
`CTRL-W`	Word under the cursor
`CTRL-A`	WORD under the cursor
`CTRL-L`	Line under the cursor