

VIM QUICK REFERENCE CARD

Frequently used VIM commands – Version v1.3 June 2015

Vim cheatsheet based on the version by Michael Goerz

Helpsections

`:h <topic>` will open the vimdocs to each *topics* help-page.

`:h /zero-width`

matches with ‘zero-width’ `<@!` patterns

Movements

`:viu` show a summary of all commands

Basic Movements

refer to helpsection `:h motion`

`h l k j` character left, right; line up, down

`b w` word *or* token left, right

`ge e` end of word *or* token left, right

`0 _ $` beginning, first, last character of line

`nG ngg` line *n*, default the last, first

`n|` column *n* of current line

`%` match of next brace, bracket, comment, `#define`

`- +` line up, down on first non-blank character

`B W` space-separated word left, right

`gE E` end of space-separated word left, right

`gO gm` beginning, middle of *screen* line

`g^ g$` first, last character of *screen* line

`fc Fc` next, previous occurrence of character *c*

`tc Tc` before next, previous occurrence of *c*

Jumps

print the jump list with `:jumps`

`{ }` beginning of previous, next paragraph

`()` beginning of previous, next sentence

`[[]]` go to previous, next function *or* \section in \TeX

`{ }` beginning, end of current block

`[z]z` move to start, end of current open fold

`n^O n^I` go to *nth* older, newer position in jump list

`ng; ng,` go to *n* older, newer position in change list

`’.` jump back on last edited line

`’’` toggle back, forward to previous, next position

`’0..9` go to last exit position

`‘ ‘ ”` go to position before jump, at last edit

`‘ [’]` go to start, end of previously operated text

`] ^T` jump to the tag under cursor, return from tag (eg. inside of vimdocs)

Advanced Scrolling

`n^Y n^E` scroll window *n* lines up, downwards

`^D ^U` scroll half a page up, down

`^F ^B` scroll page up, down

`zt zz zb` current line to top, center, bottom of win.

`zh zl` scroll one character to the right, left

`zh zL` scroll half a screen to the right, left

`H M L` jump to *high, middle, low* position in screen

Insert, Command & Visual Mode

Insertion, Replace → insert mode

`cm` change text of movement command *m*

`cc or S` change current line

`C` change to the end of line

`i a` insert before, after cursor

`o O` open a new line below, above the current line

`I A` insert at beginning, end of line

`gi gI` insert text on last edited line, first column

`rc grc` replace character under cursor, without affecting layout

`R gR` replace characters starting at the cursor, without affecting layout

`sc` substitute character *c* under cursor → insert mode

Command Mode CTRL-Keys

`^Vc ^Vn` insert char *c* literally, decimal value *n*

`^A ^@` insert previously inserted text, stop insert → command mode

`^N ^P` text completion before, after cursor

`^W ^U` delete word before cursor, to start of line

`^D ^T` shift left, right one shift width

`^Oc` execute *c* in temporary command mode

`<esc> or ^]` abandon edition → command mode

CTRL-R

`^R^A` content under cursor to command mode

`^R = 5*5` insert 25 into text

`^Rx ^R^Rx` ... insert content of register *x*, literally

CTRL-X (Command Mode Completions)

`^X^L` whole lines

`^X^N ^X^I` keywords in current file, plus included files

`^X^K ^X^N` keywords in dictionary, thesaurus

`^X^]` tags

`^X^F` file names

`^X^D` definitions or macros

`^X^V` vim command line

`^X^U` user defined completion

`^X^O` omni completion

Visual Mode

refer to `:h object-select`

v V ^Vstart *or* stop highlighting characters, lines, block
o exchange cursor position with start of highlighting
gv start highlighting on previous visual area
aw as ap select a word, a sentence, a paragraph
ab aB select a block (), a block { }
g^G Count words, character lines and bytes of selection

Delete, Copy to Registers

Deletion
refer to :h und-tree

x Xdelete character under, before cursor
dm delete text of movement command *m*
dd Ddelete current line, to the end of line
u U undo last command, restore last changed line
⇒ To revert the current buffer to the state before the very first change remembered by Vim in the undo list, use the command: :u1|u
. ^R repeat last *n* changes, redo last undo

Copying

"x use register *x* for next delete, yank, put
ym yank the text of movement command *m*
yy or Y yank current line into register
p P put register after, before cursor position
]p [p like **p, P** with indent adjusted
gp gP like **p, P** leaving cursor after new text

Registers & Macros
show content of all registers: **:reg**

:@x execute register *x* as an *Ex* command

:register x show content of single register *x*
qx qX q record, append, stop recording typed characters as macro into register *x*
@x execute macro of register *x*
⇒ execute *x* on each file of buffer list: :bufdo exe "%normal @x"
@@ repeat previous recorded macro
qxq empty register *x*
⇒ delete lines with pattern *p* into Register *x* and copy to clipboard afterwards: :g/p/d x | let @+ = @x
q: @: list all, repeat macro

Search & Substitute
substitutions work like :s/p/q/flag, you may limit your search to an area between ranges (Ex ranges).

Search, Substitute Flags	
c	confirm each substitution
e	do not issue error messages and continue as if no error occurred
g	replace all occurrences in the line
i I	ignore, mind case for the pattern (overwrites 'ignorecase' and 'smartcase' options)
p # l	print the line containing the last substitute, like :list , prepend line number afterwards
&	must be the first one: keep flags from the previous substitute
n	report the number of matches, do not actually substitute. (the [c] flag is ignored.)

Forward & Backward Searches
refer to :h search-commands

/s↔ ?s↔ search forward, backward for *s*
/s/o↔ ?s?o↔ search fwd, bwd for *s* with offset *o*

Offsets

The offset gives the cursor position relative to the found match:

n, -n	lines down-, upwards, in col 1
e+n, e-n	characters to the right, left of the end of the match
s+n, e-n	characters to the right, left of the start of the match
b+n, e-n	identical to s+,-n above (mnemonic: begin)
;p	perform another search

Quick Search Commands

n or /↔ repeat forward last search
N or ?↔ repeat backward last search
* search backward, forward for word under cursor
g# g* same, but also find partial matches
gd gD local, global definition of symbol under cursor

Substitutions
refer to :h :sub

:rs/p/q/g ... substitute all *p* by *q* in range *r*
:rs q repeat substitution with new *r* & *q*
:rg/x/e :rv/x/e execute *e* on range *r* where *x* matches, not matches
⇒ join any line containing the string *x* with previous line, if it lies between the *a* and *b* marks: :'*a*,'bg/x/-1j
:rg/x/s/p/q/g for every line in *r* containing *x*, substitute *p* with *q*
:r&& :r& repeat last search or substitution on range *r* with, without flags

Ex Ranges
refer to :h cmdline-ranges

,	cursor position interpreted from current line
;	the cursor position will be set to line of last search or substitution

<i>n</i>	an absolute line number <i>n</i>
<code>. \$</code>	the current, last line in file
<code>% *</code>	entire file, visual area
<code>'t</code>	position of mark <i>t</i>
<code>/p/ ?p?</code>	the next, previous line where <i>p</i> matches
<code>-n +n</code>	preceding, appending line <i>n</i>

Patterns (differences to Perl)

refer to `:h pattern` and `:h /zero-width`

<code>< ></code>	start, end of word
<code>\i \k \I \K</code>	an identifier, keyword; excludigits
<code>\f \p \F \P</code>	a file name, printable char.; excludigits
<code>\e \t \r \b</code>	<code><esc></code> , <code><tab></code> , <code><↵></code> , <code><↵></code>
<code>= * +</code>	match 0..1, 0..∞, 1..∞ of preceding atoms
<code>{n,m}</code>	match <i>n</i> to <i>m</i> occurrences
<code>{-}</code>	non-greedy match
<code> </code>	separate two branches (\equiv <i>or</i>)
<code>(\)</code>	group patterns into an atom
<code>& \1</code>	the whole matched pattern, 1 st () group
<code>&</code>	a <i>branch</i> : matches last concat, but only if all preceding concats also match at the same position
\Rightarrow	the following pattern finds all lines that contain both "red" and "blue", in any order: <code>/*red&.*blue</code>
<code>\u \l</code>	upper, lowercase character
<code>\U \L</code>	id., whole pattern
<code>\c \C</code>	ignore, match case on pattern
<code>\@= \@!</code>	<code>char(=?pattern)</code> <code>char(?!pattern)</code>
\Rightarrow	matches pattern, only when line is not ending in 'foo': <code>pattern\(foo\) \@!\$</code>
<code>\@<= \@<!</code>	<code>(=?pattern)char (?!pattern)char</code>
\Rightarrow	everything before the comment '#' is excluded from pattern: <code>/\(#.*\) \@<=pattern</code>

<code>\@></code>	<code>(?>pattern)</code>
<code>_ _ \$</code>	start-of-line, end-of-line, anywhere in pattern
<code>_.</code>	any single char, including end-of-line
\Rightarrow	find any pattern <i>foo bar</i> , even when divided by linebreak: <code>foo_ \s*bar</code>
<code>\zs \ze</code>	set start, end of pattern
<code>\%^ \%\$</code>	match start, end of file
<code>\%n1 \%nc \%nv</code>	matches specific line, column, virtual column <i>n</i>
<code>\%x</code>	match hex character
<code>\%V</code>	match inside visual area
<code>\'m</code>	match with position of mark <i>m</i>
<code>\%(\)</code>	unnamed grouping
<code>\[]</code>	collection with end-of-line included
<code>\%[]</code>	sequence of optionally matched atoms
<code>\v</code>	very magic: patterns almost like perl

Advanced Operations

Special Text Operations

<code>cgn dgn</code> change, delete the next search pattern match (repeat change, deletion with <code><.></code>)
<code>J gJ</code> join current line with next, without space
<code>~ g~m</code> switch case and advance cursor, on movement <i>m</i>
<code>gum gUm</code>	... switch case, lc, uc on movement <i>m</i>
<code>guu gUU</code> lower-/uppercase line

Marks and Tags

<code>:tags</code>	print tag list, <code>:marks</code> print the active marks list
<code>mc</code> mark current position with mark <i>c</i> $\in [a..Z]$
<code>'c 'C</code> go to mark <i>c</i> in current, <i>C</i> in any file

Ex Commands (\leftrightarrow)

refer to `:help holy-grail` for list of all commands

Tags

<code>:tselect t</code>	..list matching tags and select one for jump
<code>:tjump t</code> jump to tag or select one if multiple matches
<code>:tag ^[</code> jump to tag (under cursor)

Reading from & writing to files

<code>:edit f</code> edit file <i>f</i> , reload current file if no <i>f</i>
<code>:args f1..fn</code>	load <i>n</i> files to buffer in background
<code>:rwrite f</code>	... write range <i>r</i> to file <i>f</i> (this file if no <i>f</i>)
<code>:rwrite >> f</code>	append range <i>r</i> to file <i>f</i>
<code>:quit :quit!</code>	quit and confirm, discard changes
<code>:wq</code> <i>or</i> <code>:x</code> <i>or</i> <code>:ZZ</code>	write to current file and exit
<code>:rdelete :rdelete x</code>	delete range <i>r</i> lines, into register <i>x</i>

Filter Lines

<code>!mc↔</code> filter lines of movement <i>m</i> through command <i>c</i>
<code>n!!c↔</code> filter <i>n</i> lines through command <i>c</i>
<code>:r!c</code> filter range <i>r</i> lines through command <i>c</i>

Insert, Send Content

<code>:r f</code> insert content of file <i>f</i> below cursor
<code>:r! c</code> insert output of command <i>c</i> below cursor
<code>:rcopy</code> <i>or</i> <code>:rt a</code>	copy range <i>r</i> below line <i>a</i>
\Rightarrow	copy all lines containing <i>foobar</i> to EOF: <code>:g/foobar/t\$</code>
<code>:rmov a</code> id. but move
<code>:rhardcopy > file.ps</code>	print range to ps file
<code>:rha rw!lp</code>	..sending <i>r</i> to printer (printout)

Compile

:clist :cfile
list all errors, read errors from file

:cnext :cprevious
display the next, previous error

:compiler c .set, show compiler plugins

:copen navigate errors from make

:make run **makeprg**, jump to first error

Standard Mode Formatting, Filtering

leave out *m* for visual mode commands

Indentation

set indent-foldmethod by **:set fdm=indent**

< m > m ...shift left, right text of movement *m*

n> n< = ...indent, unindent *n* levels, reindent

n << n >> .. shift *n* lines left, right

Alignment

gqm gqgq ... format movement *m*, current line

:rce w center lines in range *r* to width *w*

:rri w rightalign lines in range *r* to width *w*

:rle i left align lines in range with indent *i*

Folds

zfm create fold of movement *m*

:rfold create fold for range *r*

zd zE delete fold at cursor, all in window

zo zc zO zC open, close one fold; recursively

zj zk move down, up to start, end of next fold

zm zM fold more, close all folds

zr zR fold less, open all folds

zn zN zi fold non, fold normal, invert folding

:set fdc=n . show foldcolumn to level *n*

Multiple Files, Buffers, Tabs (↔)

Generic Buffer Commands

:tab ball ...show buffers as tablist

:buffers show list of buffers

:on make current window one on screen

:new :vnew ..create new empty window (vert.)

:bnswitch to buffer *n*

:bn :bp :bf :bl
buffer movement next, prev, first, last

:bdn delete buffer *n* (also with filename)

⇒ Delete all Buffers with Extension 'ext':
:bd *.ext ^A

:badd f.txt load file into new buffer

:bufdo cmd.. execute *cmd* in each buffer in the buffer list.

:sbn Split window and edit buffer *n* from the bufflist

Buffer Shortcuts

refer to **:h ctrl-w**

^^ toggle between the current and the last window

^Wf gfopen file under cursor in new, current window

^Ww ^W^Wmove to window below, above (wrap)

^Wj ^Wkmove to window below, above

^Wt ^Wbmove to top, bottom window

^Wc ^Woclose current, all other window(s)

^Ws ^Wvsplit window in two (vert.)

^Wx or ^W^R... swap open buffer windows

^Wn+ ^Wn- ... increase, decrease window size by *n* lines

^Wn > ^Wn < .increase, decrease window width

^W = Make all windows equally high and wide

^W n_set window height to *n* (default: very high)

^W n| set current window width to *n*

Tab Management

:tabs list all tabs including their displayed windows

:tabfirst ...go to first tab

:tablastgo to last tab

:tabnew open a new empty tab page

:tabclose ...close current tab page

:qall :wqall quit, and save all tabs

:tabonly close all other tabs

gt gT go to next, previous Tab

ngt goto tab in position *n*

Miscellaneous

Spell Check

activate spellcheck: **:set spell spelllang=en_us**

]s [s next, previous misspelled word

zg zG add good word (to internal word list)

zug zuG undo the addition of a word to the dictionary

zw zW mark bad word (to internal word list)

z= suggest corrections

Invocation

vimdiff f₁ f₂
diff *file₁* + *file₂* using synchronized split windows

vim -o/-O f₁ f₂..f_n
open *files* in horiz, vert split mode

vim +n file . open *file* at *n*th line (eof if *n* omitted)

vim +/s file open *file* and search for *string*

vim -S name reload vim-session *name*

Special operations

Usefull (and not so usefull) operations which don't fit to any other section :-)

K run **keywordprg** (manpage) on word under cursor

$\text{\textasciix{A}}$ $\text{\textasciix{X}}$ increment, decrement number under cursor
 $\text{\textasciix{L}}$ redraw screen
 $\text{\textasciix{ga}}$ show ASCII value of character under cursor
 $\text{\textasciix{gf}}$ open filename under cursor
 $\text{\textasciix{gG}}$ count words, characters, bytes (in selection or buffer)
 $\text{\textasciix{Kc_1c_2}}$ *or* $c_1 \leftarrow c_2$
 enter digraph $\{c_1, c_2\}$
 \Rightarrow for a complete list of all digraphs enter:
 $\text{\textasciix{:digraphs}}$ *or* $\text{\textasciix{:h digraph-table}}$