VIM QUICK REFERENCE CARD

Frequently used VIM commands – Version v1.3 September 2015

Vim cheatsheet based on the version by Michael Goerz

Helpsections

`:h <topic>` will open the vimdocs to each *topic*s helppage.

`:h /zero-width` matches with 'zero-width' $\langle @! \rangle$ 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

`g0 gm` .... beginning, middle of *screen* line

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

`fc Fc` .... next, previous occurence of character c

`tc Tc` .... before next, previous occurence 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

`nˆO nˆI` ..... go to n^{th} 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

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

Advanced Scrolling

`nˆY nˆE` ...... scroll window n lines up, downwards

`ˆD ˆU` .... scroll half a page up, down

`ˆF ˆB` .... 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

`ˆVc ˆVn` ..... insert char c literally, decimal value n

`ˆA ˆ@` .... insert previously inserted text, stop insert → command mode

`ˆN ˆP` .... text completion before, after cursor

`ˆW ˆU` .... delete word before cursor, to start of line

`ˆD ˆT` .... shift left, right one shift width

`ˆOc` .... execute c in temporary command mode

`⟨esc⟩` *or* `ˆ]` .... abandon edition → command mode

CTRL-R

`ˆRˆA` .... content under cursor to command mode

`ˆR = 5*5` ..... insert 25 into text

`ˆRx ˆRˆRx` ... insert content of register x, literally

CTRL-X (Command Mode Completions)

`ˆXˆL` .... whole lines

`ˆXˆN ˆXˆI` .... keywords in current file, plus included files

`ˆXˆK ˆXˆN` .... keywords in dictionary, thesaurus

`ˆX^]` .... tags

`ˆXˆF` .... file names

`ˆXˆD` .... definitions or macros

`ˆXˆV` .... vim command line

`ˆXˆU` .... user defined completion

`ˆXˆO` .... omni completion

Visual Mode
refer to `:h object-select`

v V ˆV .... start *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 { }

gˆG .... Count words, character lines and bytes of selection

Delete, Copy to Registers

Deletion
refer to :h und-tree

x X .... delete character under, before cursor

d*m* .... delete text of movement command *m*

dd D .... delete 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

. ˆR .... repeat last *n* changes, redo last undo

Copying

"*x* .... use register *x* for next delete, yank, put

y*m* .... 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

:reg*ister* *x* show content of single register *x*

q*x* q*X* 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

q*x*q.......... 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 *r*anges (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

:*r*s/*p*/*q*/g ... substitute all *p* by *q* in range *r*

:*r*s *q* .... repeat substitution with new *r* & *q*

:*r*g/*x*/e :*r*v/*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*,'*b*g/*x*/-1j

:*r*g/*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

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

Patterns (differences to Perl)

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

\< \>	start, end of word
\i \k \I \K	an identifier, keyword; excl.digits
\f \p \F \P	a file name, printable char.; excl.digits
\e \t \r \b	⟨esc⟩, ⟨tab⟩, ⟨↩⟩, ⟨←⟩
\= * \+	match 0..1, 0..∞, 1..∞ of preceding atoms
\{n,m\}	match n to m occurrences
\{-\}	non-greedy match
|	separate two branches (≡ or)
\(\)	group patterns into an atom
& \1	the whole matched pattern, 1st () group
\&	a *branch*: matches last concat, but only if all preceding concats also match at the same position

⇒ the following pattern finds all lines that contain both "red" and "blue", in any order: /.*red\&.*blue

\u \l	upper, lowercase character
\U \L	id., whole pattern
\c \C	ignore, match case on pattern
\@= \@!	*char*(?=pattern) *char*(?!pattern)

⇒ matches pattern, only when line is not ending in 'foo': *pattern*\(foo\)\@!$

\@<= \@<!	(?=pattern)*char* (?!pattern)*char*

⇒ everything before the comment '#' is excluded from pattern: /\(#.*\)\@<=*pattern*

\@>	(?>pattern)
_^ _$	start-of-line, end-of-line, anywhere in pattern
_.	any single char, including end-of-line

⇒ find any pattern *foo bar*, even when divided by linebreak: foo_\s*bar

\zs \ze	set start, end of pattern
\%^ \%$	match start, end of file
\%nl \%nc \%nv	matches specific line, column, virtual column n
\%x	match hex character
\%V	match inside visual area
\'m	match with position of mark m
\%(\)	unnamed grouping
_[ ]	collection with end-of-line included
\%[ ]	sequence of optionally matched atoms
\v	very magic: patterns almost like perl

Advanced Operations

Special Text Operations

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

Marks and Tags

:tags print tag list, :marks print the active marks list

mc	mark current position with mark $c \in [a..Z]$
'c 'C	go to mark c in current, C in any file

Ex Commands (↩)

Tags

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

Reading from & writing to files

:edit f	edit file f, reload current file if no f
:args $f_1..f_n$	load n files to buffer in background
:rwrite f	write range r to file f (this file if no f)
:rwrite >> f	append range r to file f
:quit :quit!	quit and confirm, discard changes
:wq *or* :x *or* ZZ	write to current file and exit
:rdelete :rdelete x	delete range r lines, into register x

Filter Lines

!mc↩	filter lines of movement m through command c
n!!c↩	filter n lines through command c
:r!c	filter range r lines through command c

Insert, Send Content

:r f	insert content of file f below cursor
:r! c	insert output of command c below cursor
:rcopy *or* :rt a	copy range r below line a

⇒ copy all lines containing *foobar* to EOF: :g/foobar/t$

:rmove a	id. but move
:rhardcopy > file.ps	print range to ps file
:rha rw!lp	sending r 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.)

`:bn` .... switch 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 gf` .... open file under cursor in new, current window

`^Ww ^W^W` ..... move to window below, above (wrap)

`^Wj ^Wk` ...... move to window below, above

`^Wt ^Wb` ...... move to top, bottom window

`^Wc ^Wo` ...... close current, all other window(s)

`^Ws ^Wv` ...... split 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

`:tablast` .... go 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 f_1 f_2`
diff $file_1$ + $file_2$ using synchronized split windows

`vim -o/-O f_1 f_2..f_n`
open $files$ in horiz, vert split mode

`vim +n file` . open $file$ at nth 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

ˆA ˆX . . . . increment, decrement number under cursor

ˆL . . . . redraw screen

ga . . . . show ASCII value of character under cursor

gf . . . . open filename under cursor

gˆG . . . . count words, characters, bytes (in selection or buffer)

ˆKc_1c_2 _or_ $c_1{\leftarrow}c_2$
　　　　　　enter digraph $\{c_1, c_2\}$

⇒　for a complete list of all digraphs enter: :digraphs _or_ :h digraph-table