Vim Cheatsheet
by Blair Labatt III, page 1 of 2

1 Navigation

Cursor Motions

```
h  l          # letter left, right
j  k          # line up, down
b  e          # beginning, end of word
B  E          # Beginning, End of WORD
w  W          # next word / Word
0  $          # very beginning (0); end ($)
_  ^          # synonyms: first non-blank
-  +          # start of line: up (-), down (-)
H  M  L       # cursor to sreen High, Med, Low
:35           # goto 35th line
|13           # goto 13th line from cur
13|           # goto 13th char on line
```

Page Motions

```
^u  ^d        # ½ pg up/down
^b  ^f        # full page bef/fwd
^y  ^e        # scroll line ↑ / ↓
[#]gg  G      # go → first / last ln (or #)
^O  ^I        # jump Out, Into prev. goto's
zz  zt  zb    # center @: cursor(z), top, bot
zL  zR        # scroll Left / Righ
```

Section Motions

```
[[  ]]        # next / prev section
[]            # out of section
][            # into section
(  )          # move ← / ← 1 sentence
```

Find

```
f<char>       # single <char> search
F<char>       # find last <char>
df<char>      # delete to <char>
/<str>        # find next <str>
/\c<str>      # ibid, case-insens.
?<str>        # backwd search
:g//          # last search match
g[d|D]        # search var defs
<C-I>         # → word @ cursor
%             # → matching ()s {}s, etc
]d            # definition search
])            # next unmatched (
]}            # previous unmatched {
]m  ]M        # start / end method
```

Marks

Mark a place in the file. Buffer-specific marks ∈ [a-z]; globals ∈ [A-Z].

```
m<letter>     # mark → letter
'<letter>     # jump to mark
'<letter>     # jump to mark's line
:marks        # list all marks
:marks ab     # list marks a, b
:delmarks ab  # delete marks a, b
d'<letter>  d'<letter>  # delete → mark
c'<letter>  c'<letter>  # change, here to ...
y'<letter>  y'<letter>  # yank, here to ...
['<am>  ']<am>          # jump → "auto mark"
```

"Automarks" include cursor location:

- ' ... before last jump
-] ... before last inserted text
- [... end of last inserted text
- " ... last cursor when left file
- '< , '> ... last visual select start, stop

Quickfix Window

```
:clist        # list errors
:cnext        # next error
:cprevious    # prev. error
:clast        # last error
:crewind      # first error
:cc           # current error
:clist<rng>   # show range
```

Tags

```
:[p]tag <tag>  # jump to tag
<C-]>          # jump to cursor's tag
<C-T>          # jump back (pop stack)
:[p|t]select   #
:[p|t]jump     # jump to tag
:[p|t]next     # next tag
:[p|t]previous # prev. tag
:[p|t]rewind   # first tag
:[p|t]last     # last tag
```

Insert Mode Navigation

```
←  →                  # move left / right
<C-←>  <C-→>          # move left 1 word
<PageUp>  <PageDown>  # page up / down
```

Code Folding

Set / modify fold "method":

```
:set foldmethod=[manual|syntax|indent]

zm            # make auto-fold
z[r|R]        # remove auto-fold (Recurse
zf            # manually fold
z[d|D]        # manually delete fold
```

2 Actions

Operators

Use like: `[#]<op><mot>` *where <mot> are the simple cursor motions described above. Eg,* `y2w` *yanks 2 word;* `3dd` *deletes 3 lines;* `cf(` *changes to first open paren.*

```
#y            # yank # chars
x  X          # delete aft / bef cursor
r  R          # replace char(s) @ cursor
c  C          # change at cursor
p  P          # paste bef. / aft.
yy  dd        # yyank ddelete line(s)
cf<char>      # change thru found <char>
»  «          # shift current line in, out
==            # zero indent, current line
#=            # autoindent # lines
~             # change case
g             # change case, whole line
gu  gU        # Δ case to Upper, lower (u)
<c-A>|<c-X>   # inc (add) / dec (x) numb
```

Enter into Insert Mode

```
I  A          # beg (I), end (A) of line
i  a          # beginning (i), end (a) of cursor
O  o          # prev (O), next (o) line
```

Insert Mode Edits

```
<C-A>         # enter last typed text
<C-V><char>   # enter escaped <char>
<C-R><r>      # paste indicated register
<C-R><C-O><r> # paste reg, orig. spacing
<C-R><C-P><r> # paste reg, auto spacing
<c-O><seq>    # exec seq in normal
<c-P>         # auto-complete
<c-X>         # type-specific complet'n
<c-T>  <c-D>  # indent, delete indent
```

Find and Replace

```
:s/from/to/gc     # global, confirm
:%s/from/to/g     # ... in all lines (%)
:5,12s/foo/bar/g  # ... in lines 5-12
:.,+5s/foo/bar/g  # ... next 5 lines
:'<,'>s/foo/bar/g # ... in visual block
:'a,'bs/foo/bar/g # ... between marks
```

Macros

```
q<letter><seq>q  # record to <letter>
@<letter>        # replay macro
5@a              # replay "a" 5 times
```

Idioms

```
xp            # switch 2 letters
!'asort       # sort to mark "a"
```

3 Buffers, Files

File Exploration

`Explore`*, a part of the native* `netrw` *package, is a useful utility for browsing the filesystem; invoke with* `:(L|S|R|N|H|T)ex`*. Some commands here. Alternatively, use a plugin like* `nvim.tree` *or* `fern.vim`*. For git exploration, use* `fugitive`*.*

Buffers

```
:buffer       # list all
:[#]bnext     # next buf.
:[#]bprevious # prev. buf.
:blast        # last buf
:brewind      # first buf
:set hidden   # hide all others
:sbnext       # split & edit
:baddd <file> # add indicated buffer
:bdelete      # discard buffer
:bunload      # remove buffer
```

Files

```
:ls           # ls open files
:e[dit] <file># open file
:enew         # open new file
:view <file>  # open read-only
:vi <file>    # same as :e
:next         # cycle to next
:previous     # cycle back
:wnext[!]     # write + next
:args         # list open files
:args <#>     # edit #th cmd-line file
CTRL-^        # goto last edited
```

Windows

```
:sp[lit]      # screen split
:splitbelow   # splitbelow
:vs[plit]     # vertical split
:10vsplit     # split is 10 lines
:only         # keep only this
:hide         # close this w.
:[w|q]all[!]  # write, quit all
:all <file>*  # open all
3<C-W>w       # goto specific (3rd)
<C-W>_        # maximize current
<C-W>w        # cycle windows
<C-W>j  <C-W>k  # up, down wind.
<C-W>↑ <C-W>↓   # up, down wind.
[#]<C-W>+ <C-W>-  # expand, contract
[#]<C-W>=  <C-W>_ # sizes equal, exact
<C-r>  <C-R>  # rotate win's on screen
<C-W>O        # quit all Others
```

Registers

Unlike other editors, Vim:

- has multiple clipboards ("registers")
- can filter registers
- can cut & paste on many files

```
"ay3w         # yank 3w into reg "a"
"Byy          # append (B vs b) to reg "b"
"<reg>[p|P]   # Paste from <reg>
"*[y|P]       # y to, P from system clip.
yy            # yanks into "unnamed" reg
:registers    # show registers
```

Special Registers:

- "@ ...
- "" ... "unnamed" (last yanked)
- ". ... last inserted
- "# ... alternate file
- "0 ... last yanked
- "% ... file name
- "/ ... last search
- ": ... last command
- "* ... mouse clipboard
- "= ... enter expression
- "_ ... black hole

4 Visual Mode

Entering visual mode:

- *v ... enter visual mode*
- *V ... enter line-visual*
- *<c-V> ... block visual*

Actions d, y, c, g, J, !, ~, all work on selections in visual mode.

Selection

Normal-mode navs begin selection from cursor.

```
aw  aW          # select word, and space
iw  iW          # select only (inner) word
as  is          # select sentence
ap  ip          # select paragraph
a(  i(          # select (text)
a><  i><        # select <text>
a][  i][        # select [text]
a}{  i{}        # select {text}
o               # cursor to other end of sel
O               # other side of block vis. sel.
"<reg>d         # cut sel. into register
$               # extend sel. to EOL
gv              # toggle last 2 sel'ns
```

Visual Block Mode

See help at `:help v_b_<cmd>`

```
I<str><esc>     # all rows, insert @ left
c               # fill block
C               # fill lines
r               # repeated fill
A               # append, all lines
<  >            # shift block
```

Select Mode

Note: SM offer subsets of the other visual modes, similar to what is usually found in other text editors, allowing only replacement of selections with <BS> or typing.

```
gh              # enter character-wise SM
gH              # enter list-wise SM
g<c-H>          # enter block-wise SM
↑  ↓            # highlight up down
←  →            # highlight left right
```

5 Options

Settings

```
:set <opt>          # turn on option
:set no<opt>        # turn off option
:set <opt>!         # toggle opt state
:set <opt>?         # query <opt> state
:set <opt>=<val>    # set option value
```

Common

backspace	matchpairs	shiftwidth
binary	number	statusline
complete	numberwidth	syntax
highlight	operatorfunc	textwidth
ignorecase	paste	wildchar
filetype	scroll[bind]	winheight
formatoptions	shiftround	wrap
makeprg	[inc|hl]search	wrapmargin
matchtime	[smart]indent	
	showmatch	

Saving

.viminfo saves marks, registers, vars, options `:set viminfo=<opt>*` *where <opt> ∈:*

```
'<#>            # max # files to save marks
f               # save globals
r<loc>          # save removeable media
h               # :nohlsearch
\"<#>           # # lines saved per buff.
%               # restore buff. list
@#              # size, input-line history
```

Indenting & Tabs

Set tabs using softtabstop, softtab, tabstop, expandtab. Set indentation with equalprog. Command :retab retroactively exchanges tabs & spaces in an open file.

6 Command Mode

Common Commands

```
:w[rite][q]         # write [&quite]
ZZ  ZQ              # ibid, alias
:help               # vim help
:help rust          # on a topic
u  <C-R>            # undo / redo
<C-K><char><char>   # insert digraph
<C-V>u<unicode>     # insert raw unicode
:digraphs           # list digraphs
<C-G>               # file loc
:source <file>      # load a file
@:                  # repeat last cmd
:!<cmd>             # run ext cmd
```

ascii	go	redraw!
digraph	gre	shell
echo[m]	ls	sleep
find	mak	statusline*
fixdel	normal	visual

Statement Evaluation

```
:execute "<str>"
```
... for example, to insert ";" at EOL & return:
```
:execute "normal! mqA;\<esc>`q"
```

Printing

```
:hardcopy > out.pdf     # to pdf
:set printfont=courier:h10  # font
```

Abbreviations

```
:abbreviations      # show abbrevs
:ab <f> <t>         # set an abbrev.
```

Syntax Highlighting

:highlight <grp> <ttype>=<attr>
where <ttype> ∈ term, cterm, gui & attr ∈:

bold	underline	reverse
[#rrggbb]	[col_name]	standout

Autocommands

Syntax:
1. :autocmd <event>* <ft_glob> <cmd>
2. :autocmd FileType <ft_glob> <cmd>

Examples:

```
:autocmd BufNewFile *.txt :write
:autocmd BufWritePre,BufRead * ...
```

Best to use "augroups":

```
:augroup <grp_name>
:    autocmd! ...
:augroup END
```

Filters

```
!15G<cmd>       # filter, here to line 15
!15G sort       # example filter
!!<cmd>         # single-line filter
!!date          # idiomatic: insert date
!!ls            # idiomatic: f.s. contents
```

Built-in Functions

append	getwinposx	matchend
arg[c|v]	glob[path]	matchstr
browse	has	nr2char
bufexists	histadd	rename
bufloaded	histdel	setline
bufname	histget	shellescape
buf[win]nr	histnr	strftime
byte2ln	hlexists	strlen
char2nr	hlID	strpart
col	hostname	strtrans
confirm	input	substitute
delete	isdirectory	synID
escape	libcal	synIDattr
exists	line	system
expand	line2byte	tempname
filereadable	localtime	visualmode
fnamemod.	maparg	virtcol
getcwd	mapcheck	winbufnr
getftime	match	winheight
getline		winnr

Sessions

```
:mksession <file># save to file
:source <file>  # restore session
```

Setting :set sessionoptions=<opt> where <opt> ∈ {buffers, globals, winpos, winsize, resize, etc}, allows you to selectively save parAts of a session.*

7 Scripting

Setup

```
:set syntax on          #
:set filetype=<lang>    #
```

Mapping

Use the form (optional <buffer> is literal):
:[|o|n|v|i]noremap <buffer>? <seq>* <cmd>
where prefix ∈: operator pending, normal, visual, insert mode.

```
:nmap -x dd     # x seq maps to dd
:map -r CTRL-R  # pplies in all modes
:nnoremap -x dd # void recursion
```

Leaders

```
:let mapleader = "<seq>"    # set
localleader = "<seq>"       # set
:nnoremap <leader>y y1W     # use
```

Special Chars

<space>	<cWORD>	<Bslash>
<C-x>	<F7>	<Up>
<Esc>	<BS>	<CR>
<cword>	<Del>	<nop>

Variables

```
:let var="foo"      # bind to var
:echo foo           # inspect variable
:echo &textwidth    # & deref's options
:let @a="hi!"       # bind to reg "a"
:let b:foo="bar"    # buffer scoped
```

Scope ∈ {args, buff, locl, scrpt, globl, wind, vim}

Functions

```
:function <fname>()     # name capitalz'd
: <cmd>*                # as needed
: return                # if necessary
:endfunction            #
```
Ref args by post'n: a:1, a:2, ..., or a:000 (all).

Conditionals

```
:if <cond>      # see operators below
: <cmd>*        # repeat as needed
:elseif         # or :else
:endif          #
```

mode-depend	insensitive	sensitive
==	==#	==?
>	>#	>?
<	<#	<?

Types

Numbers ∈ ℤ, ℚ (int or float), and displayed as hex (eg 0xff), oct (eg 015), or sci (3.0e8).
Strings denoted with "; concat'd with . ; special chars escaped with \.
Dictionaries very much like json, keys are strings; deref with dot-notation or []s.
Lists defined [1,2,3], indexed lst[3]; concat'd with +, & can nest [1,[3,4],2] or slice [1:3].
Manipulate with funcs (:help functions) like len(), get(), index(), keys(), join(), reverse(), etc.

Loops

```
:let c = 0          # for setup
:for i in [1,2,3]   # or :while
: <cmd>*            # as needed
:endfor             # or :endwhile
```

Plugins

Located in ~/.vim/plugin/ or ~/.config/nvim/.