Vim Commands Cheat Sheet

by Kendall Conrad of Angelwatt.com

Created: 2011-08-12

Updated: 2012-05-24

Legend

`{R}`	You can supply a range e.g., 2,4 represents lines 2 - 4, using '<,'> represents the currently selected text.
`{#}`	You can use a number here.
`{char}`	A character.
`<C-key>`	Control plus a key.
`<S-key>`	Shift plus a key.
`<CR>`	Carriage return / enter.
`<M-key>`	Meta-key plus a key. Command key on Mac.
`<BS>`	The backspace key
`<Del>`	The delete key.
`{M}`	You can use motion keys: h, j, k, l, w, $.
`{V}`	From visual mode.
`{file}`	Represents a file path an name.
`{cmd}`	A command.
`{pat}`	A pattern, regular expression possible.

Getting Started

Editing a File

`:e[dit]`	Edit the current file. This is useful to re-edit the current file, when it has been changed outside of Vim.
`:e!`	Edit the current file always. Discard any changes to the current buffer. This is useful if you want to start all over again.
`:e {file}`	Edit {file}.
`:e! {file}`	Edit {file} always. Discard any changes to the current buffer.
`:e *.txt`	Open all txt files. See multiple files section to navigate.
`gf`	Edit the file whose name is under or after the cursor. Mnemonic: "goto file".
`:e .` `:Explore` `:Exp`	Open file explorer in current folder to pick a file for editing.
`:Sexplore` `:Sex`	Open file explorer in split window.
`:bro[wse] e`	Windows stlye file browser. For GUI vim.

Saving

`:w[rite]`	Write current file. Do not exit.
`:w c:/aaa/%`	Save file somewhere else.
`:e!`	Return to unmodified file.

`:sav[eas] {file}`	Saves the current file as a new file name.
`:sav! %<.bak`	Save current file with an alternate file extension. Old way.
`:sav! %:r.cfm`	Save current file with an alternate file extension.
`:sav %:s/find/repl/`	Do a substitute on a file name.
`:sav %:s/find/repl/:r.cfm`	Do a substitute on file name and extension.
`:!mv % %:r.bak`	Rename current file (DOS use rename or del).
`:help filename-modifiers`	Get help on this.

How to Exit

`:q[uit]`	Quit Vim. This fails when changes have been made.
`:q!`	Quit without writing.
`:qa[ll]`	Quit all windows.
`:cq[uit]`	Quit always, without writing.
`:wq`	Write the current file and exit.
`:wq!`	Write the current file and exit always.
`:wq {file}`	Write to {file}. Exit if not editing the last
`:wq! {file}`	Write to {file} and exit always.
`:{R}wq[!]`	[file] Same as above, but only write the lines in {R}.
`ZZ`	Write current file, if modified, and exit.
`ZQ`	Quit current file and exit (same as ":q!").

How to Suspend

`<C-z>`	Suspend Vim, like ":stop". Works in Normal and in Visual mode. In Insert and Command-line mode, the <C-z> is inserted as a normal character.
`:sus[pend][!]` `:st[!]`	Suspend Vim. If the '!' is not given and 'autowrite' is set, every buffer with changes and a file name is written out. If the '!' is given or 'autowrite' is not set, changed buffers are not written, don't forget to bring Vim back to the foreground later!

Inserting Text

Insert Basics

`a`	Append text after the cursor {#} times.
`A`	Append text at the end of the line {#} times.
`i`	Insert text before the cursor {#} times.
`I`	Insert text before the first non-blank in the line {#} times.
`gI`	Insert text in column 1 {#} times.
`gi`	Insert text in the same position as where Insert mode was stopped last time in the current buffer.
`o`	Begin a new line below the cursor and insert text, repeat {#} times.
`O`	Begin a new line above the cursor and insert text, repeat {#} times.

Maps and Abbreviations

Mappings and abbreviations can be defined for your current session as well as being added to your .vimrc file so it's always available. From the .vimrc, a : is not needed at the start.

Kinds of Maps

`cmap`	Handles command-line mappings.
`imap`	Handles insert-only mappings.
`map`	Maps keys in normal, visual, and operator-pending mode.
`map!`	Maps keys in Vim's command and insert modes.
`nmap`	Maps keys in normal mode only.
`omap`	Maps keys in operator-pending mode only.
`vmap`	Maps keys in visual mode only.

Setting Mappings

`:map` See all mappings.

`:map <F10> <Esc>:tabnew<CR>`
Maps the F10 key to open a new tab.

`:unmap <F10>`
Removes mapping for the F10 key.

`:iunmap, :ounmap, :vunmap, etc.`
Unmap for each kind.

`:noremap, :inoremap, :onoremap, etc.`
Prevents a key from being remapped.

Setting Abbreviations

Abbreviations are activated when you hit space, enter, or escape.

`:ab[breviate] helo hello`
Will auto switch "helo" to "hello" when typing.

`:iab`	Will set the abbreviation only for insert mode.
`:una helo`	Removes the abbreviation for "helo."
`:abc[lear]`	Clears all abbreviations.

Auto-Complete

`<C-n>` `<C-p>`	Complete word while in insert mode.
`<C-x>` `<C-l>`	Complete line
`:set dictionary=dict`	Define dict as a dictionary
`<C-x>` `<C-k>`	Complete with dictionary
`<C-d>`	While in command mode, it will show a list of commands that exist starting with the text you've already typed in.
`<Tab>`	While in command mode, it will auto-complete, continuing to hit it will cycle through all matches.

Inserting a File or Command

`:r[ead] {file}`	Insert the file {file} below the cursor.
`:r !{cmd}`	Execute {cmd} and insert its standard output below the cursor.
`:!`	Executes a command from the command line.

`%`	On command line, it represents the file name.
`g?{M}`	Perform ROT13 transformation on movement.
`{#}<C-a>`	Adds {#} to number beneath cursor, one by default.
`{#}<C-x>`	Subtracts {#} to number beneath cursor, one by default.
`gq{M}`	Format lines of movement to fixed width.
`:{R}ce[nter] {#}`	Center lines in range to a given width.
`:{R}le[ft] {#}`	Left align lines in range {R} with indent {#}.
`:{R}ri[ght] {#}`	Right align lines in range {R} with width {#}.
`!{M}{cmd}`	Filter lines of movement {M} through command.
`{#}!!{cmd}`	Filter {#} lines through command.
`:{R}!{cmd}`	Filter range {R} lines through command.

Editing Text

Deleting Text

`<Del>`	
`x`	Delete {#} characters under and after the cursor
`X`	Delete {#} characters before the cursor
`d{M}`	Delete text that {M} moves over
`:delete`	
`dd`	Delete {#} lines
`D`	Delete the characters under the cursor until the end of the line
`{V}x`	
`{V}d`	Delete the highlighted text (for {V} see Selecting Text).
`{V}<C-h>`	
`{V}<BS>`	When in Select mode: Delete the highlighted text
`{V}X`	
`{V}D`	Delete the highlighted lines
`:{R}d`	Delete {R} lines (default: current line)
`:{R}d {#}`	Delete n number of lines, starting with {R}

Changing (or Replacing) Text

`{#}r{char}`	Replace the character under the cursor with {char}.
`R`	Enter Insert mode, replacing characters rather than inserting
`c{#}w`	
`c{#}{M}`	
`ce`	Deletes from the current position to the end of the word and puts you in insert mode.
`~`	Switch case of the character under the cursor and move the cursor to the right. If a {#} is given, do that many characters.
`~{M}`	switch case of {M} text.
`{V}~`	Switch case of highlighted text

Copying and Moving Text

`"{a-zA-Z0-9.%#:-"}`	Use register {a-zA-Z0-9.%#:-"} for next delete, yank or put (use uppercase character to append with delete and yank) ({.%#:} only work with put).
`:reg[isters]`	Display the contents of all numbered and named registers.

`:reg {arg}`	Display the contents of the numbered and named registers that are mentioned in {arg}.
`:di[splay] {arg}`	Same as :registers.
`["x]y{M}`	Yank {M} text [into register x].
`["x]yy`	Yank {#} lines [into register x]
`["x]Y`	yank {#} lines [into register x] (synonym for yy).
`{V}["x]y`	Yank the highlighted text [into register x] (for {V} see Selecting Text).
`{V}["x]Y`	Yank the highlighted lines [into register x]
`:yank`	
`:{R}y [x]`	Yank {R} lines [into register x].
`:{R}y [x] {#}`	Yank {#} lines, starting with last line number in {R} (default: current line), [into register x].
`@{reg}`	Execute content of register.
`@@`	Repeat previous @ command.
`:@{reg}`	Execute register as an Ex command.
`:{R}g/foo/{cmd}`	Execute Ex command on range where pattern matches.
`["x]p`	Put the text [from register x] after the cursor {#} times.
`["x]P`	Put the text [from register x] before the cursor {#} times.
`["x]gp`	Just like "p", but leave the cursor just after the new text.
`["x]gP`	Just like "P", but leave the cursor just after the new text.
`]p`	
`[p`	Like p, P with indent adjusted.
`:put`	
`:{line}pu {char}`	Put the text [from register {char}] after {line} (default current line).
`:{line}pu! {char}`	Put the text [from register {char}] before {line} (default current line).

Undo/Redo/Repeat

`{#}u`	Undo {#} changes.
`:u[ndo] {#}`	Undo one change.
`{#}<C-r>`	Redo {#} changes which were undone.
`:red[o]`	Redo one change which was undone.
`U`	Undo all latest changes on one line. {Vi: while not moved off of it}
`.`	Repeat last change, with count replaced with {#}.
`g-`	Go to older text state.
`g+`	Go to newer text state.
`:ea[rlier] 4m`	Takes the document back 4 minutes earlier to show its state.
`:lat[er] 45s`	Takes you 45 seconds later in the documents state.
`:undol[ist]`	Show your undo list.

Selecting Text (Visual Mode)

To select text, enter visual mode with one of the commands below, and use motion commands to highlight the text you are interested in. Then, use some command on the text. The operators that can be used are:

`v`	Start Visual mode per character.
`V`	Start Visual mode linewise.
`<Esc>`	Exit Visual mode without making any changes
`o`	Moves the cursor position with the selected region between the front and back.

	Useful for including more text at the start or end.
`aw`	Mark a word.
`aW`	Select a WORD.
`iw`	Inner word, selects word, but not following whitespace.
`iW`	Inner WORD selection.
`as`	Selects a sentence.
`is`	Selects inner sentence.
`ap`	Select a paragraph.
`a"` `a'` `` a` ``	Select a quoted string, including the quotes.
`i"` `i'` `` i` ``	Selects text inside a quote, excluding the the quotes.
`ab` `a(` `a)`	Select all text inside () including them.
`a[` `a]`	Selects the text inside a [] block, including the brackets.
`i]` `i[`	Selects the text inside a [] block, exclusing the brackets.
`a{` `a}` `aB`	Select all text inside {} including them.
`ib` `i(` `i)`	Select only text within ().
`i{` `i}` `iB`	Select only text within {}.
`a>` `a<`	Selects text inside a <> block, including the brackets.
`i>` `i<`	Selects text inside a <> block, excluding the brackets.
`at`	Selects a tag block, including the tag.
`it`	Selects a tag block, excluding the tag.

`~`	switch case
`d`	delete
`c`	change
`y`	yank
`>`	shift right
`<`	shift left
`!`	filter through external command
`=`	filter through 'equalprg' option command
`gq`	format lines to 'textwidth' length

Substituting

```
:substitute/foo/bar/
```

`:{R}s/foo/bar/[cegpriI] {#}`
: For each line in {R} replace a match of foo with bar.

`:{R}s [cegriI] {#} :{R}&[cegriI] {#}`
: Repeat last :substitute with same search pattern and substitute string, but without the same flags. You may add extra flags

`:%s/old/new/g` Change every occurrence in the whole file.

The arguments that you can use for the substitute commands.

`c`	Confirm each substitution. Vim positions the cursor on the matching string. You can type:

`y`	to substitute this match
`n`	to skip this match [esc] to skip this match
`a`	to substitute this and all remaining matches {not in Vi}
`q`	to quit substituting {not in Vi}
`<C-e>`	to scroll the screen up {not in Vi}
`<C-y>`	to scroll the screen down {not in Vi}.

`e`	When the search pattern fails, do not issue an error message and, in particular, continue in maps as if no error occurred.
`g`	Replace all occurrences in the line. Without this argument, replacement occurs only for the first occurrence in each line.
`i`	Ignore case for the pattern.
`I`	Don't ignore case for the pattern.
`p`	Print the line containing the last substitute.

Process Found

`:{R}global/{pat}/{cmd}` `:g/{pat}/`	Execute the Ex command {cmd} (default ":p") on the lines within {R} where {pat} matches.
`:{R}g!/{pat}/{cmd}` `:{R}v/{pat}/`	Execute the Ex command {cmd} on lines where {pat} does NOT match.
`:g/find/#`	Display lines with 'find' along with line numbers.
`:v/\S/d`	Delete empty lines (and blank lines ie whitespace).
`:g/^$/,/./-j`	Compress empty lines.
`:g/^/ if line('.')%2|s/^/zz /`	Perform a substitute on every other line.
`:g/^/exec "s/^/".strpart(line(".")." ", 0, 4)` `:%s/^/\=strpart(line(".")." ", 0, 5)` `:%s/^/\=line('.'). ' '`	Insert line numbers into file.
`q{reg}`	Record typed characters into register {reg} (uppercase to append). the 'q' command is disabled wile executing a register, and it doesn't work inside a mapping and :normal. Hitting 'q' again stops recording.
`@{reg}{#}`	Execute the contents of register {#} times.
`@@`	Repeat previous @{reg}.
`d/fred/`	Delete until fred found.
`y/fred/`	Yank until fred found.
`c/fred/e`	Change until fred end.
`guu` `Vu` `gUU`	Change line to lowercase

`VU`	Change line to uppercase.
`g~~`	Flip the case of the line.
`vEU`	Upper case word.
`vE~`	Flip case for word.
`ggguG`	Lowercase the entire file.

Moving Around

Cursor Motion

`{#}h`	{#} characters to the left (exclusive).
`{#}l`	
`space`	{#} characters to the right (exclusive).
`{#}k`	
`<C-p>`	{#} lines upward
`{#}j`	
`<C-j>`	
`<CR>`	
`<C-n>`	{#} lines downward (linewise).
`0`	To the first character of the line (exclusive).
`Home`	To the first character of the line (exclusive).
`^`	To the first non-blank character of the line
`$`	
`<End>`	To the end of the line and [count - 1] lines downward
`g{M}`	Move one screen position in a given direction. Helpful for when dealing with long lines and you only want to go down a screen line versus a physical line.
`g0`	
`g<Home>`	When lines wrap ('wrap on): To the first character of the screen line (exclusive). Differs from "0" when a line is wider than the screen. When lines don't wrap ('wrap' off): To the leftmost character of the current line that is on the screen. Differs from "0" when the first character of the line is not on the screen.
`g^`	When lines wrap ('wrap' on): To the first non-blank character of the screen line (exclusive). Differs from "^" when a line is wider than the screen. When lines don't wrap ('wrap' off): To the leftmost non-blank character of the current line that is on the screen. Differs from "^" when the first non-blank character of the line is not on the screen.
`g$`	
`g<End>`	When lines wrap ('wrap' on): To the last character of the screen line and [count - 1] screen lines downward (inclusive). Differs from "$" when a line is wider than the screen. When lines don't wrap ('wrap' off): To the rightmost character of the current line that is visible on the screen. Differs from "$" when the last character of the line is not on the screen or when a count is used.
`f{char}`	To {#}'th occurrence of {char} to the right. The cursor is placed on {char} (inclusive).
`F{char}`	To the {#}'th occurrence of {char} to the left. The cursor is placed on {char} (inclusive).
`t{char}`	Till before {#}'th occurrence of {char} to the right. The cursor is placed on the character left of {char} (inclusive).
`T{char}`	Till after {#}'th occurrence of {char} to the left. The cursor is placed on the character right of {char} (inclusive).
`;`	Repeat latest f, t, F or T {#} times.
`,`	Repeat latest f, t, F or T in opposite direction {#} times.

`{#}-`	{#} lines upward, on the first non-blank character (linewise). Minus key.
`{#}+`	
`<C-m>`	
`<CR>`	{#} lines downward, on the first non-blank character (linewise).
`{#}_`	{#} - 1 lines downward, on the first non-blank character (linewise).
`<C-End>`	
`{#}G`	Goto line {#}, default last line, on the first non-blank character.
`<C-Home>`	
`gg`	Goto line {#}, default first line, on the first non-blank character.
`:5`	Go to line 5.
`80|`	Got to column 80
`<S-Right>`	
`w`	{#} words forward
`<C-Right>`	
`W`	{#} WORDS forward
`e`	Forward to the end of word {#}
`E`	Forward to the end of WORD {#}
`<S-Left>`	
`b`	{#} words backward
`<C-Left>`	
`B`	{#} WORDS backward
`ge`	Backward to the end of word {#}
`gE`	Backward to the end of WORD {#}
`%`	Finds and moves to matching braces, e.g. (), {}, [].

Screen Movement

`<C-e>`	Scroll up a line
`<C-y>`	Scroll down a line
`<C-d>`	Scroll half a page down.
`<C-u>`	Scroll half a page up.
`<C-f>`	Scroll a page forward.
`<C-b>`	Scroll a page backward.
`H`	Move the cursor to the top visible line on the screen.
`M`	Move the cursor to the middle visible line on the screen.
`L`	Move the Cursor to the last visible line on the screen.
`zt`	
`z<CR>`	Scroll current line to top of window
`zz`	
`z.`	Set current line at center of window.
`zb`	
`z-`	Set current line at bottom of window.
`zh`	Scroll one character left.
`zl`	Scroll one character to the right.
`zH`	Scroll half a screen to the left.
`zL`	Scroll half a screen to the right.

These commands move over words or WORDS.

A word consists of a sequence of letters, digits and underscores, or a sequence of other non-blank characters, separated with white space (spaces, tabs, <eol>). This can be changed with the 'iskeyword' option.

A WORD consists of a sequence of non-blank characters, separated with white space. An empty line is also considered to be a word and a WORD.

`(`	{#} sentences backward
`)`	{#} sentences forward
`{`	{#} paragraphs backward
`}`	{#} paragraphs forward
`]]`	{#} sections forward or to the next '{' in the first column. When used after an operator, then the '}' in the first column.
`][`	{#} sections forward or to the next '}' in the first column
`[[`	{#} sections backward or to the previous '{' in the first column
`[]`	{#} sections backward or to the previous '}' in the first column
`[{`	Unclosed {} backward.
`}]`	Unclosed {} forward.
`[*`	Start of /*.
`]*`	End of */.

Searching

`/{pattern}[/]<CR>`	Search forward for the {#}'th occurrence of {pattern}
`/{pattern}/{offset}<CR>`	Search forward for the {#}'th occurrence of {pattern} and go {offset} lines up or down.
`/<CR>`	Search forward for the {#}'th latest used pattern
`//{offset}<CR>`	Search forward for the {#}'th latest used pattern with new. If {offset} is empty no offset is used.
`?{pattern}[?]<CR>`	Search backward for the {#}'th previous occurrence of {pattern}
`?{pattern}?{offset}<CR>`	Search backward for the {#}'th previous occurrence of {pattern} and go {offset} lines up or down
`?<CR>`	Search backward for the {#}'th latest used pattern
`??{offset}<CR>`	Search backward for the {#}'th latest used pattern with new {offset}. If {offset} is empty no offset is used.
`n`	Repeat the latest "/" or "?" {#} times.
`N`	Repeat the latest "/" or "?" {#} times in opposite direction.
`<C-o>`	Goes back where you were before the search.
`<C-i>`	Goes forward to where you were at.
`%`	Finds and moves to matching braces, e.g. (), {}, [].
`#`	Search for word under cursor.
`*`	Search backward for word under cursor.
`g#`	Search for partial matches of word under cursor.
`g*`	Search backward for partial matches for word under cursor.
`gd`	Local definition of symbol under cursor.
`gD`	Global definition of symbol under cursor.

`\<`	Within a search, represents the beginning of a word.
`\>`	Within a search, represents the ending of a word.

Marks

`m{a-zA-Z}`	Set mark {a-zA-Z} at cursor position (does not move the cursor, this is not a motion command).
`m' or m``	Set the previous context mark. This can be jumped to with the "''" or "``" command (does not move the cursor, this is not a motion command).
`:{R}ma[rk] {a-zA-Z}`	Set mark {a-zA-Z} at last line number in {R}, column 0. Default is cursor line.
`:{R}k{a-zA-Z}`	Same as :mark, but the space before the mark name can be omitted.
`'{a-z}`	To the first non-blank character on the line with mark {a-z} (linewise).
`'{A-Z0-9}`	To the first non-blank character on the line with mark {A-Z0-9} in the correct file
`` `{a-z} ``	To the mark {a-z}
`` `{A-Z0-9} ``	To the mark {A-Z0-9} in the correct file
`:marks`	List all the current marks (not a motion command).
`:marks {arg}`	List the marks that are mentioned in {arg} (not a motion command).
`:jumps`	Print the jump list.
`{#}<C-o>`	Go to {#}th older position in jump list.
`{#}<C-i>`	Go to {#}th newer position in jump list.

Tags

`:tags`	Print tag list.
`:ta {tag}`	Jump to tag.
`:{#}ta`	Jump to {#}th newer tag in list.
`<C-]>`	Jump to the tag under cursor.
`<C-t>`	Return from tag.
`:ts {tag}`	List matching tags and select one for jump.
`:tj {tag}`	Jump to tag or select one if multiple matches.
`:{#}po`	Jump back from {#}th older tag.
`:{#}<C-t>`	Jump to {#}th older tag.
`:tl`	Jump to last matching tag.
`<C-w>}`	Preview tag under cursor.
`:pt {tag}`	Preview tag.
`<C-w>]`	Split window and show tag under cursor.
`<C-w>z`	
`:pc`	Close tag preview window.

Multiple Documents

`:e *.txt`	Open all txt files.
`:buffers`	
`:ls`	List currently open file.
`:bn`	Go to next buffer / open file.
`:bp`	Go to previous buffer / file.
`:b{#}`	Go to buffer {#}.

`:b {str}`	Go to buffer with {str} in the name.
`:rew`	Return to beginning of edited files list (:args).
`:brew`	Buffer rewind.
`:sb[all]`	Split all buffers (super).
`:wn`	Save file and move to next (super).
`:wp`	Save file and move to previous.
`:bd`	Remove file from buffer list.
`:bun`	Buffer unload, remove window but not from list.
`:badd {file}`	Add file to buffer list.
`:sav {file}`	Save current file as new name, {file}.

Split Window

`:new`	Horizontally splits the window with a new document. Doesn't have to be given a name yet.

`:sp[lit] {file}`
`<C-w> n`
	Split current file (or a given filename) into two equal size viewports.

`:vs[plit] {file}`	Splits current file vertically into equal viewports.
`:sview {file}`	Opens a file in split view for viewing (read-only).
`:vert sview {file}`	Opens a file in vertical split view for viewing (read-only).

`<C-w> o`
`:on[ly]`
	Make current window one on screen.

`:sp +/searchstring {file}`	Include a search string to move directly to the first instance of a keyword.
`:10 sp {file}`	Split viewport for {file} and give it a size of 10 lines.
`<C-w> _`	Maximise current viewport
`<C-w> =`	Makes all viewports equal in size.
`<C-w> {M}`	Move to a neighboring viewport.
`<C-w> <C-w>`	Switch to next viewport.
`<C-w> [-+]`	Reduce/Increase viewport size by one line
`<C-w> 5[-+]`	Reduce/Increase viewport size by 5 lines.
`:resize {#}`	Change window size to {#}.
`:vertical res[ize] {#}`	Resize vertical split size to {#}.
`<C-w> |`	Makes vertical split pane maximum size.
`<C-w> {#}<`	Make vertical split {#} units narrower.
`<C-w> {#}>`	Make vertical split {#} units wider.

`:q`
`<C-w> q`
`:hide`
	Close the current viewport. Vim will prompt to save if unsaved.

`:on[ly]`	Close all viewports, except current one
`<C-w> r`	Rotate a viewports position to the right/down.
`<C-w> R`	Rotates viewports positions to the left/up.
`<C-w> K`	Moves the window to the topmost position.
`:set scrollbind`	When set before doing a split, the two viewports will scroll together.

Tabs

`:tabs`	List your current tabs.

`:tabnew`
`:tabe[dit] {file}`
	Creates a new tab with an optional {file} path.

`:tabfind {file}`
`:tabf`
	Opens a new tab with {file}, searches the 'path' for the file.

`:tabn[ext] {#}`
`{#}gt`
	Show next tab, or go to next {#} tab.

`:tabp[revious] {#}`
`gT`
	Show previous tab.

`:tabc[lose] {#}`	Close the current tab, or a given numbered tab {#}.
`:tabo[nly]`	Close all other tabs, but they're buffers will still be available.

`:tabfir[st]`
`:tabr[ewind]`
	Show first tab

`:tabl[ast]`	Show last tab
`:tabm[ove] {#}`	Rearrange tabs with optional position {#}. Zero is first position, no position will move to last osition.
`:tabd[o] %s/pattern/string/g`	Execute command in all tabs
`:tab ball`	Puts all open files in tabs.
`:tab help`	Open a new help window in its own tab.
`:tab drop {file}`	Open {file} in a new tab, or jump to a window/tab containing it.
`:tab split`	Copy the current window to a new tab of its own.

FOLDING

`zf{M}`	Create fold of movement.
`:{R}fo`	Create folder for range.
`zd`	Delete fold at cursor.
`zE`	Delete all fold in window.
`zo`	Open fold.
`zc`	Close fold.
`zO`	Open fold recursively.
`zC`	Close fold recursively.
`[z`	Move to start of current open fold.
`]z`	Move to end of current open fold.
`zj`	Move down to start of next fold.
`zk`	Move up to end of previous fold.

SETTINGS

Default settings can be configured in your .vimrc file. They can be set for your current session as well though. Most settings can be undone by providing "no" to the front of the item.

Document Settings

`set autochdir`
: Automatically changes the current working directory to that of the file being edited. Alternative: `autocmd BufEnter * silent! lcd %:p:h`

`set auoindent (ai)`
: Turns on autoindent for when you go to new lines.

`set linebreak (lbr)`

If on Vim will wrap long lines at a character in '*breakat*' rather than at the last character that fits on the screen. Unlike '*wrapmargin*' and '*textwidth*', this does not insert <EOL>s in the file, it only affects the way the file is displayed, not its contents. The value of '*showbreak*' is used to put in front of wrapped lines. This option is not used when the '*wrap*' option is off or '*list*' is on.

`set list (l)`
Same as :print, but display unprintable characters with '^' and put $ after the line.

`set number (nu, #)`
Same as :print, but precede each line with its line number.

`set ruler (ru)`
Show the line and column number of the cursor position, separated by a comma. When there is room, the relative position of the displayed text in the file is shown on the far right.

`set scroll=13 (scr)`
Number of lines to scroll with <C-u> and <C-d> commands. Will be set to half the number of lines in the window when the window size changes.

`set scrolloff=6 (so)`
Minimal number of screen lines to keep above and below the cursor.

`set shiftwidth=4 (sw)`
Sets the number of spaces to use for each step of (auto)indent. Used for cindent, >>, <<, etc.

`set showmatch`
When a bracket is inserted, briefly jump to the matching one. The jump is only done if the match can be seen on the screen. The time to show the match can be set with '*matchtime*'.

`set showtabline=2`
Lets you see the tab line.

`set syntax=cpp (syntax-highlighting, coloring)`
Set the syntax style to a specific type.

`set tabstop=4 (ts)`
Set how many spaces equal a tab.

`set wrap`
This option changes how text is displayed. It doesn't change the text in the buffer, see '*textwidth*' for that. When on, lines longer than the width of the window will wrap and displaying continues on the next line.

File Settings

`set backupdir=~/tmp,. (bdir)`
Sets the directory to be used for backups.

`set directory=~/tmp,. (dir)`
List of directory names for the swap file, separated with commas.

`set encoding=utf8 (enc)`
Sets the character encoding used inside of Vim. It applies to text in the buffers, registers, Strings in expressions, text stored in the viminfo file, etc. It sets the kind of characters which Vim can work with.

`set fileencoding=utf8 (fenc, E213)`
Sets the character encoding for the file of this buffer.

`set fileencodings=utf8 (fencs)`
This is a list of character encodings considered when starting to edit an existing file. When a file is read, Vim tries to use the first mentioned character encoding. If an error is detected, the next one in the list is tried. When an encoding is found that works, 'fileencoding' is set to it. If all fail, 'fileencoding' is set to an empty string, which means the value of 'encoding' is used.

`set filetype on`

Turns on file type detection.

`set filetype=cpp (file-type)`
Sets the file type to a specific type. Generally used within a document to force a specific file type.

Font and Color Settings

`set cursorline (cul)`
Highlight the screen line of the cursor with CursorLine.

`set highlight (hi)`
Coloring.

`hi Comment ctermgf=Gray`
Set color for comments.

`hi Identifier ctermfg=Brown`
Set color for identifiers.

`hi LineNr ctermfg=white`
Sets the color of the line numbers along the left side.

`hi String ctermfg=Red`
Sets the color used for strings.

`set guifont=Monaco:h12`
Set the font for the GUI vim editor. Supply font name and the size in points.

Colorable Items

`ColorColumn`	Used for the columns set with '*colorcolumn*'.
`Conceal`	Placeholder characters substituted for concealed text.
`Cursor`	The character under the cursor
`CursorIM`	Like cursor, but used when in IME mode.
`CursorColumn`	The screen column that the cursor in in when '*cursorcolumn*' is set.
`CursorLine`	The screen line that the cursor is in when *cursorline* is set.
`Directory`	Directory names (and other special names in listings).
`DiffAdd`	diff mode: Add line.
`DiffChange`	diff mode: Changed line.
`DiffDelete`	diff mode: Deleted line.
`DiffText`	diff mode: Changed text within a changed line.
`ErrorMsg`	Error messages on the command line.
`VertSplit`	The column separating vertically split windows.
`Folded`	Line used for closed folds.
`FoldColumn`	The *foldcolumn*.
`IncSearch`	*Incsearch* highlighting; also used for text replaced with ":s///c".
`LineNr`	Line number for ":number" and ":#" commands, and when *number* or *relativenumber* option is set.
`MatchParen`	The character under the cursor or just before it, if it is a paired bracket, and its match.
`ModeMsg`	*showmode* message, e.g., -- INSERT --.
`NonText`	'~' and '@' at the end of the window, characters from *showbreak* and other characters that do not really exist in the text.
`Normal`	Normal text.
`Pmenu`	Popup menu: normal item.
`PmenuSel`	Popup menu: selected item.

`PmenuSbar`	Popup menu: scrollbar.
`PmenuThumb`	Popup menu: thumb of the scrollbar.
`Question`	The *hit-enter* prompt and yes/no questions.
`Search`	Last search pattern highlighting (hlsearch).
`SpecialKey`	Meta and special keys listed with ":map", also for text used to show unprintable characters in the text, *listchars*. Generally: text that is displayed differently from what it really is.
`SpellBad`	Word that is not recognized by the spellchecker.
`SpellCap`	Word that should start with a capital.
`SpellLocal`	Word that is recognized by the spellcheacker as one that is used in another region.
`SpellRare`	Word that is recognized by the spell checker as one that is hardly ever used.
`StatusLine`	Status line of current window.
`StatusLineNC`	Status lines of not-current windows.
`TabLine`	Tab pages line, not active tab page label.
`TabLineFill`	Tab pages line, where there are no labels.
`TabLineSel`	Tab pages line, active tab page label.
`Title`	Titles for output from ":set all", ":autocmd" etc.
`Visual`	Visual mode selection.
`VisualNOS`	Visual mode selection when vim is "Not Owning the Selection".
`WarningMsg`	Warning messages.
`WildMenu`	Current match in *wildmenu* completion.

Colors

- Black
- DarkBlue
- DarkGreen
- DarkCyan
- DarkRed
- DarkMagenta
- Brown, DarkYellow
- LightGray, LightGrey, Gray, Grey
- DarkGray, DarkGrey
- Blue, LightBlue
- Green, LightGreen
- Cyan, LightCyan
- Red, LightRed
- Magenta, LightMagenta
- Yellow, LightYellow
- White
- #rrggbb format

Other Settings.

`behave mswin`
Sets the behavior for mouse and selection. (mswin | xterm) Changes selectmode, mousemodel, keymodel, selection.

`set compatible (cp)`
This option has the effect of making Vim either more Vi-compatible, or make Vim behave in a more useful way.

`set hlsearch (hls)`

When there is a previous search pattern, highlight all its matches.

`set incsearch (is)`
: While typing a search command, show where the pattern, as it was typed so far, matches.

`set ignorecase (ic)`
: Sets ignore case for search and substitutions.

`set keymodel=startsel,stopsel`
: List of comma separated words, which enable special things that keys can do. *Startsel*: Using a shifted special key starts selection (either Select mode or Visual mode, depending on "key" being present in 'selectmode'). *stopsel*: Using a not-shifted special key stops selection. Specials keys are: end, home, pageup, and pagedown.

`set mousemodel=popup (mousem)`
: Sets the model to use for the mouse. The name mostly specifies what the right mouse button is used for. *extend*: Right mouse button extends a selection. *popup*: pops up a menu. The shifted left mouse button extends a selection. *popup_setpos*: Like "popup," but the cursor will be moved to the position where the mouse was clicked, and thus the selected operation will act upon the clicked object.

`set selection=exclusive (sel)`
: This option defines the behavior of the selection. It is only used in Visual and Select mode. Values: old, inclusive, exclusive.

`set selectmode=(mouse,key) (slm)`
: This is a comma separated list of words, which specifies when to start Select mode instead of Visual mode, when a selection is started. Values: mouse, key, cmd.

`set showcmd (sc)`
: Show (partial) command in the last line of the screen.

MISCELLANEOUS

Help

`:h[elp]`	Opens the help screen. Type :q to get out.
`:help w`	Get help for command w.
`:tab help tabpage`	Opens the help in its own tab.

Vimtutor

Instead of running vim from your shell try running vimtutor instead. This is a built in tutorial for VIM, it is a very usfull and handy tool.

File Operations

`<C-g>`	Shows file information such as current line number, total lines, column position, file name, and percentage through file.

`:pwd`
: Show the present working directory.

`:cd %:p:h`
: Change the working directory to where the current file is located. This changes for all vim windows. p represents the full path, h gives its directory, "head" of the full path.

`:lcd %:p:h`
: Change the working directory to where the current file is located, but only for the current window.

Stuff

`K`	Lookup keyword under cursor with man.
`:make`	Start make, read errors and jump to first.
`:cn`	Display next error.
`:cp`	Display previous error.
`:cfirst`	Goto first error.
`:clast`	Goto last error.
`:cl`	List all errors.
`:copen`	Opens split window with make output.
`:cw`	Opens split window with make output only if there is an error.
`:cwindow`	Read errors from file.
`:cf`	Redraw screen.
`<C-l>`	
`ga`	Show ASCII value of character under cursor.
`:redir>{file}`	Redirect output to file.
`:mkview {file}`	Save view configuration to file.
`:loadview {file}`	Load view configuration from file.

Resources

- Condensed graphical version.
- http://www.ssel.montana.edu/HowTo/
- http://vim.runpaint.org/toc/
- http://www.pixelbeat.org/vim.tips.html
- Best of Vim Tips