VIM Cheat Sheet

Changing Vim Modes

Command	Description
`i`	Enter INSERT mode
`a`	Enter INSERT mode after the cursor (think: append)
`A`	Enter INSERT mode at the end of the line (think: Append)
`o`	Open new line below the cursor and enter INSERT mode
`O`	Open new line above the cursor and enter INSERT mode
`v`	Enter VISUAL mode
`Ctrl-v`	Enter VISUAL-BLOCK mode
`:`	Enter COMMAND-LINE mode
`R`	Enter REPLACE mode
`ESC`	Go back to NORMAL mode from other modes

Exiting

Command	Description
`:w`	Write (save) file without exiting
`:wa`	Write (save) all open files without exiting
`:q`	Quit but fail if unsaved changes exist
`:q!`	Quit and discard unsaved changes
`:wq` or `:x`	Write (save) and quit
`:wqa`	Write and quit on all open files

Moving Around Within Vim

Arrows

Command	Description
`h`	Move cursor left (left most)
`j`	Move cursor down (looks like down arrow)
`k`	Move cursor up

Command	Description
`l`	Move cursor right (right most)

Movements Within A Line

Command	Description
`$`	Move cursor to the end of the line
`0`	Move cursor to the beginning of the line
`^`	Move cursor to first non-blank character in line
`fx`	Find next occurrence of character ‘x’
`Fx`	Find previous occurrence of character ‘x’
`tx`	Go towards next occurrence of character ‘x’ (stops right before it)
`Tx`	Go towards previous occurence of character ‘x’ (stops right before it)
`;`	Repeat previous `f`, `F`, `t`, or `T` movement forwards
`,`	Repeat previous `f`, `F`, `t`, or `T` movement backwards

Word Movements

Definitions:

- word: Sequence of letters, digits and underscores OR sequence of other symbols, separated by whitespace. Use `:h word` to learn more.
- WORD: Any sequence of non-blank characters (any symbols, letters, digits, etc…), separated by whitespace. Use `:h WORD` to learn more.

Command	Description
`w`	Move cursor forwards to start of word (sequence of letters, digits, underscores OR sequence of other symbols)
`W`	Move cursor forwards to start of WORD (any sequence of non-blank characters)
`b`	Move cursor backwards to start of word (sequence of letters, digits, underscores OR sequence of other symbols)
`B`	Move cursor backwards to start of WORD (any sequence of non-blank characters)
`e`	Move cursor forwards to end of word (sequence of letters, digits, underscores OR sequence of other symbols)
`E`	Move cursor forwards to end of WORD (any sequence of non-blank characters)

Command	Description
`ge`	Move cursor backwards to end of word (sequence of letters, digits, underscores OR sequence of other symbols)
`gE`	Move cursor backwards to end of WORD (any sequence of non-blank characters)

Sentence Movements

Definitions

- sentence: A sentence ends with a ".", "!" or "?" followed by the end of the line, a space or tab. Use `:h sentence` to learn more.

Command	Description
`)`	Move cursor to next sentence
`(`	Move cursor to previous sentence

Paragraph Movements

Definitions:

- paragraph: Blocks of consecutive non-empty lines. NOTE: Line with white space is not empty. Use `:h paragraph` to learn more.

Command	Description
`}`	Move cursor to next paragraph (block of consecutive non-empty lines)
`{`	Move cursor to previous paragraph (block of consecutive non-empty lines)

Moving To Specific Lines

Note: Replace `{number}` with an actual number. You can also use numbers in front of other cursor movements like `{number}w`, `{number}b` or `{number}` and many others.

Command	Description
`gg`	Move cursor to first line of document
`G`	Move cursor to last line of document
`{number}G`	Move cursor to line `{number}`
`{number}j`	Go `{number}` lines down
`{number}k`	Go `{number}` lines up
`H`	Move cursor to line at the top of the window

Command	Description
`M`	Move cursor to the line at the middle of the window
`L`	Move cursor to the line at the bottom of the window

Parenthesis, Bracket, Curly Brace and Method Navigation

Command	Description
`%`	Find next parenthesis, bracket or curly brace in front of or under the cursor and jump to its match
`[(`	Go to previous unmatched `(`
`[{`	Go to previous unmatched `{`
`])`	Go to next unmatched `)`
`]}`	Go to next unmatched `}`
`]m`	Go to next start of method (Java like languages)
`]M`	Go to next end of method
`[m`	Go to previous start of method
`[M`	Go to previous end of method

Screen Related Cursor Movements

Command	Description
`Ctrl-F`	Move cursor forwards one full screen
`Ctrl-B`	Move cursor backwards one full screen
`Ctrl-D`	Move cursor down half a screen
`Ctrl-U`	Move cursor up half a screen

Scrolling While Leaving Cursor In Place

Command	Description
`zz`	Place current cursor line in the middle of the window
`zt`	Place current cursor line at the top of the window
`zb`	Place current cursor line at the bottom of the window
`Ctrl-E`	Scroll down a single line, leaving cursor in place
`Ctrl-Y`	Scroll up a single line, leaving cursor in place

Search Movements

Note: Use `:set ignorecase` for case insensitive searching and `:set smartcase` to override case insensitivity if search pattern has upper case characters.

Command	Description
`/pattern`	Search forward for pattern
`?pattern`	Search backward for pattern
`*`	Search forward for the word under or in front of the cursor
`#`	Search backward for the word under or in front of the cursor
`n`	Repeat last search in same direction
`N`	Repeat last search in opposite direction

Tip: Use `:nohl` after a search to temporarily turn off search highlights until another search command is used.

Navigating The Jump List

Context

Certain vim movements that move the cursor several lines away will add entries to the jumplist. You can display the jumplist with `:jump`.

Common commands that will add entries to the jumplist: `G`, `gg`, `[number]G`, `/`, `?`, `n`, `N`, `%`, `(`, `)`, `{`, `}`, `:s`, `L`, `M`, `H`. Navigating to a different file/buffer in the same window also works.

IMPORTANT: `[number]j` and `[number]k` will not add entries to the jumplist.

For a complete list of commands that add entries to the jumplist use `:h jump-motions`.

I use this often to navigate from and to different buffers/files in the same window.

Command	Description
`Ctrl-O`	Go to the previous cursor position in the jump list
`Ctrl-I`	Go to the next cursor position in the jump list

Editing Text

Now that you know how to move within Vim, editing text will go hand in hand with Vim movements.

Note: A `{motion}` is a Vim command that moves the cursor. Like many of the commands described in the previous section.

Deletion

Command	Description
`d{motion}`	Delete the text that the `{motion}` command moves over and copy into register.
`dd`	Delete whole current line and copy into register.
`D`	Delete from under the cursor to the end of the line and copy into register.

Tip: You can also use a `{number}` before these commands to execute the deletion that `{number}` of times.

Some Examples

Example	Description
`dw`	Delete from cursors current position to start of next word
`de`	Delete from cursors current position to end of word
`dG`	Delete from cursors current position to end of file
`d]}`	Delete from cursors current position to next unmatched `}`
`2dd`	Delete whole line under cursor and line below it.

The possible combinations are endless…

Undo & Redo

Command	Description
`u`	Undo last change
`Ctrl-R`	Redo changes that have been undone with `u`

Tip: You can also use `{number}` before these undo & redo commands to execute it that `{number}` of times.

Changing Text

Note: Executing a change command with `c` is pretty much the same as `d` except that it takes you into insert mode afterwards.

Command	Description
`c{motion}`	Delete the text that the `{motion}` command moves over, copy into register and enter insert mode.
`cc`	Delete whole current line, copy into register and enter insert mode.
`C`	Delete from under the cursor to the end of the line, copy into register and enter insert mode.

Examples would be the same as with delete but changing `c` to `d`.

Repeating a File Change

Command	Description
`.`	Repeat the last change you made to the file

Tip: You can use `{number}` before `.` to repeat the change that `{number}` of times.

This is a really nice one!

Replacing & Deleting Characters

Note: When executing, substitute `{character}` with an actual character.

Command	Description
`r{character}`	Replace current character under cursor with `{character}`
`R`	Enter replace mode and start replacing characters by typing until `ESC` is pressed
`x`	Delete current character under the cursor and copy into register

Tip: You can use `{number}` before `r` and `x` to execute that `{number}` of times.

Yank (Copy) and Paste (Put)

Command	Description
`y{motion}`	Yank or copy text that the motion command moves over into register
`yy`	Yank or copy whole current line into register
`Y`	Yank or copy from under the cursor to the end of the line into register
`p`	Put or paste the text found in register (register x) after the cursor
`P`	Put or paste the text found in register (register x) before the cursor

Tip: You can use `{number}` before `y` or `p` to repeat the yank (copy) or put (paste) command that `{number}` of times.

Note: Commands such as `d`, `c` and `x` mentioned above also copy text into a register. These as well as the `y` command copy into register x by default.

Changing Case

Command	Description
`~`	Switch case of character under cursor and move cursor to the right
`~{motion}`	Switch the case of the text that the `{motion}` command moves over
`gu{motion}`	Change the text that the `{motion}` command moves over to lowercase
`guu`	Make whole current line lower case
`gU{motion}`	Change the text that the `{motion}` command moves over to uppercase
`gUU`	Make whole current line upper case

Search/Replace

Command	Description
`:%s/old/new/g`	Replace all occurrences of “old” with “new” in whole file
`:%s/old/new/gc`	Replace all occurrences of “old” with “new” in whole file, asking for confirmation
`:%s/old/new/gi`	Replace all occurrences of “old” with “new” in whole file, ignoring case

Working With Text Objects and Inside/Around 🚀

I personally love using this feature!

You can use Vim text objects to execute an operator on that object or select it with Visual mode (Use `v` to enter visual mode).

First, here’s a list of handy text objects to remember.

Text Objects To Remember

Object	Description
`a"`	A double quoted string, including the quotes
`i"`	A double quoted string, excluding the quotes
`a'`	A single quoted string, including the quotes
`i'`	A single quoted string, excluding the quotes
`a(` or `a)`	A block surrounded by parenthesis, including the parenthesis

Object	Description
`i(` or `i)`	A block surrounded by parenthesis, excluding the parenthesis
`a[` or `a]`	A block surrounded by brackets, including the brackets
`i[` or `i]`	A block surrounded by brackets, excluding the brackets
`a{` or `a}`	A block surrounded by curly braces, including the curly braces
`i{` or `i}`	A block surrounded by curly braces, excluding the curly braces
`a<` or `a>`	Text surrounded by `<>`, including the opening `<` and the closing `>`
`i<` or `i>`	Text surrounded by `<>`, excluding the opening `<` and the closing `>`
`at`	A block surrounded by xml/html tags, including the tags
`it`	A block surrounded by xml/html tags, excluding the tags
`aw`	A word including the surrounding whitespace
`iw`	A word excluding the surrounding whitespace
`ap`	A paragraph including the surrounding whitespace
`ip`	A paragraph including the surrounding whitespace

How to use text objects

To use text objects, place the cursor anywhere inside one and type an `{operator}` followed by the `{text object}` to execute the operator on that object.

Tip: Think of ‘a’ as around and ‘i’ as inside.

Examples

Command	Description
`diw`	Delete word that cursor is in, keeping surrounding whitespace (Think: “delete inside word”)
`daw`	Delete word that cursor is in as well as surrounding whitespace (Think: “delete around word”)
`di(`	Delete everything within parenthesis surrounding cursor, keeping the surrounding parenthesis (Think: “delete inside parenthesis”)
`da(`	Delete everything within parenthesis surrounding cursor as well as the surrounding parenthesis (Think: “delete around parenthesis”)
`di"`	Delete everything within double quotes surrounding cursor, keeping the surrounding double quotes (Think: “delete inside double quotes”)

Command	Description
`da"`	Delete everything within double quotes surrounding cursor as well as the surrounding double quotes (Think: “delete around double quotes”)
`dit"`	Delete everything within tags surrounding cursor, keeping the surrounding tags (Think: “delete inside tags”)
`dat`	Delete everything within tags surrounding cursor as well as the surrounding tags (Think: “delete around tags”)

Of course these examples from above can be applied to other operators like `c` or `y` for changing, copying, etc… or using `v` instead of an operator to select the text object in visual mode.

Indentation

Command	Description
`>{motion}`	Indent text that the `{motion}` command moves over, to the right
`>>`	Indent whole current line to the right

Tip: You can use `{number}` before `>` and `>>` to execute the indentation that `{number}` of times. For example, use `2>>` to indent the current line and the line below it. Tip # 2: You can also use text objects with `>`