A Great Vim Cheat Sheet

I've compiled a list of essential Vim commands that I use every day. I have then given a few instructions on how to make Vim as great as it should be, because it's painful without configuration.

Essentials

Cursor movement (Normal/Visual Mode)

- `h` `j` `k` `l` - Arrow keys
- `w` / `b` - Next/previous word
- `W` / `B` - Next/previous word (space seperated)
- `e` / `ge` - Next/previous end of word
- `0` / `$` - Start/End of line
- `^` - First non-blank character of line (same as `0w`)

Editing text

- `i` / `a` - Start insert mode at/after cursor
- `I` / `A` - Start insert mode at the beginning/end of the line
- `o` / `O` - Add blank line below/above current line
- `Esc` or `Ctrl+[` - Exit insert mode
- `d` - Delete
- `dd` - Delete line
- `c` - Delete, then start insert mode
- `cc` - Delete line, then start insert mode

Operators

- Operators also work in Visual Mode
- `d` - Deletes from the cursor to the movement location
- `c` - Deletes from the cursor to the movement location, then starts insert mode
- `y` - Copy from the cursor to the movement location
- `>` - Indent one level
- `<` - Unindent one level
- You can also combine operators with motions. Ex: `d$` deletes from the cursor to the end of the line.

Marking text (visual mode)

- `v` - Start visual mode
- `V` - Start linewise visual mode
- `Ctrl+v` - Start visual block mode
- `Esc` or `Ctrl+[` - Exit visual mode

Clipboard

- `yy` - Yank (copy) a line
- `p` - Paste after cursor
- `P` - Paste before cursor
- `dd` - Delete (cut) a line
- `x` - Delete (cut) current character
- `X` - Delete (cut) previous character
- `d` / `c` - By default, these copy the deleted text

Exiting

- `:w` - Write (save) the file, but don't quit
- `:wq` - Write (save) and quit
- `:q` - Quit (fails if anything has changed)
- `:q!` - Quit and throw away changes

Search/Replace

- `/pattern` - Search for pattern
- `?pattern` - Search backward for pattern
- `n` - Repeat search in same direction
- `N` - Repeat search in opposite direction
- `:%s/old/new/g` - Replace all old with new throughout file (`gn` is better though)
- `:%s/old/new/gc` - Replace all old with new throughout file with confirmations

General

- `u` - Undo
- `Ctrl+r` - Redo

Advanced

Cursor movement

- `Ctrl+d` - Move down half a page
- `Ctrl+u` - Move up half a page
- `}` - Go forward by paragraph (the next blank line)
- `{` - Go backward by paragraph (the next blank line)
- `gg` - Go to the top of the page
- `G` - Go the bottom of the page
- `: [num] [enter]` - Go to that line in the document
- `ctrl+e / ctrl+y` - Scroll down/up one line

Character search

- `f [char]` - Move forward to the given char
- `F [char]` - Move backward to the given char
- `t [char]` - Move forward to before the given char
- `T [char]` - Move backward to before the given char
- `;` / `,` - Repeat search forwards/backwards

Editing text

- `J` - Join line below to the current one
- `r [char]` - Replace a single character with the specified char (does not use Insert mode)

Visual mode

- `O` - Move to other corner of block
- `o` - Move to other end of marked area

File Tabs

- `:e filename` - Edit a file
- `:tabe` - Make a new tab
- `gt` - Go to the next tab
- `gT` - Go to the previous tab
- `:vsp` - Vertically split windows
- `ctrl+ws` - Split windows horizontally
- `ctrl+wv` - Split windows vertically
- `ctrl+ww` - Switch between windows
- `ctrl+wq` - Quit a window

Marks

- Marks allow you to jump to designated points in your code.
- `m{a-z}` - Set mark {a-z} at cursor position
- A capital mark {A-Z} sets a global mark and will work between files
- `'{a-z}` - Move the cursor to the start of the line where the mark was set
- `''` - Go back to the previous jump location

Text Objects

- Say you have `def (arg1, arg2, arg3)` , where your cursor is somewhere in the middle of the parenthesis.
- `di(` deletes everything between the parenthesis. That says "change everything inside the nearest parenthesis". Without text objects, you would need to do `T(dt)` .
- Learn more

General

- `.` - Repeat last command
- `Ctrl+r + 0` in insert mode inserts the last yanked text (or in command mode)
- `gv` - reselect (select last selected block of text, from visual mode)
- `%` - jumps between matching `()` or `{}`

Making VIM actually useful

Vim is quite unpleasant out of the box. It's an arcane experience:

- Autocomplete is missing
- System clipboard is not used
- Act of typing `:w` to save is cumbersome
- Mouse doesn't work
- Management of multiple files is tricky
- Ability to indent multiple lines is missing

It does have a significant strength though: your fingers can stay on the main keyboard keys to do most editing actions. This is faster and more ergonomic. I find that the toughest part about VIM is guiding people towards getting the benefits of VIM without the drawbacks. Here are two ideas on how to go about this.

Switch caps lock and escape

- I highly recommend you switch the mapping of your caps lock and escape keys. You'll love it, promise! Switching the two keys is platform dependent.

Visual Studio Code

- VSCode is the simplest way to give you a fantastic editor that also gives you the benefits of VIM. Just install the VIM extension.
- I made a few slight changes which improved the experience for me.

Configure native VIM

For all the given limitations, you'll need to find a solution. You can either solve the issues one by one, or you can use a reference .vimrc settings file that fix most of the issues out-of-the-box.

- My .vimrc file could be a good starting point. Honestly, it's a bit old and not the best. I now use VSCode mainly so I haven't kept a great vimrc.

Using the system clipboard

- `"+y` copy a selection to the system clipboard
- `"+p` paste from the system clipboard
- If this doesn't work, it's probably because Vim was not built with the system clipboard option. To check, run `vim --version` and see if `+clipboard` exists. If it says `-clipboard` , you will not be able to copy from outside of Vim.
  - For Mac users, homebrew install Vim with the clipboard option. Install homebrew and then run `brew install vim` .
    - then move the old Vim binary: `$ mv /usr/bin/vim /usr/bin/vimold`
    - restart your terminal and you should see `vim --version` now with `+clipboard`

Sublime Text

- Another option is to use Vintageous in Sublime Text (version 3). This gives you Vim mode inside Sublime. I suggest this (or a similar setup with the Atom editor) if you aren't a Vim master. Check out Advanced Vim if you are.
- Vintageous is great, but I suggest you change a few settings to make it better.
  - Clone this repository to `~/.config/sublime-text-3/Packages/Vintageous` , or similar. Then check out the "custom" branch.
    - Alternatively, you can get a more updated Vintageous version by cloning the official repository and then copying over this patch.
  - Change the user settings (`User/Preferences.sublime-settings`) to include:
    - `"caret_style": "solid"`
    - This will make the cursor not blink, like in Vim.
    - Sublime Text might freeze when you do this. It's a bug; just restart Sublime Text after changing the file.
  - `ctrl+r` in Vim means "redo". But there is a handy Ctrl + R shortcut in Sublime Text that gives an "outline" of a file. I remapped it to alt+r by putting this in the User keymap
    - `{ "keys": ["alt+r"], "command": "show_overlay", "args": {"overlay": "goto", "text": "@"} },`
  - Add the ability to toggle Vintageous on and off
  - Mac users: you will not have the ability to hold down a navigation key (like holding j to go down). To fix this, run the commands specified here: https://gist.github.com/kconragan/2510186
- Now you should be able to restart sublime and have a great Vim environment! Sweet Dude.

Other

I don't personally use these yet, but I've heard other people do!

- `:wqa` - Write and quit all open tabs (thanks Brian Zick)

Additional resources

- Advanced Vim
- Practical Vim is a fantastic resource on many of the useful hidden features of vim.