1. Code editor:

What functions do we need for a good code editor?

Common operation for a general text editor

- Insertion
- Deletion
- Modification
- Copy/Cut/Paste

Specific requirement for a code editor

- Searching: files, variables, strings etc.
- Completion:
- Navigation: scroll, jump-to-definition etc.
- Snippet
- Integration with Git
- Built-in shell
- Debugging? (I don’t use it)

How to perform non-typing functions in different code editors?

1. Using mouse and buttons: IDEs like Visual Studio or Xcode
2. Ctrl/Shift/Alt/Tab/Command + alphabet keys: like Emacs
3. Commands: Ctrl+Shift+P in VSCode, Sublime etc.
4. Mode-dependent key interpretation: Vim way!

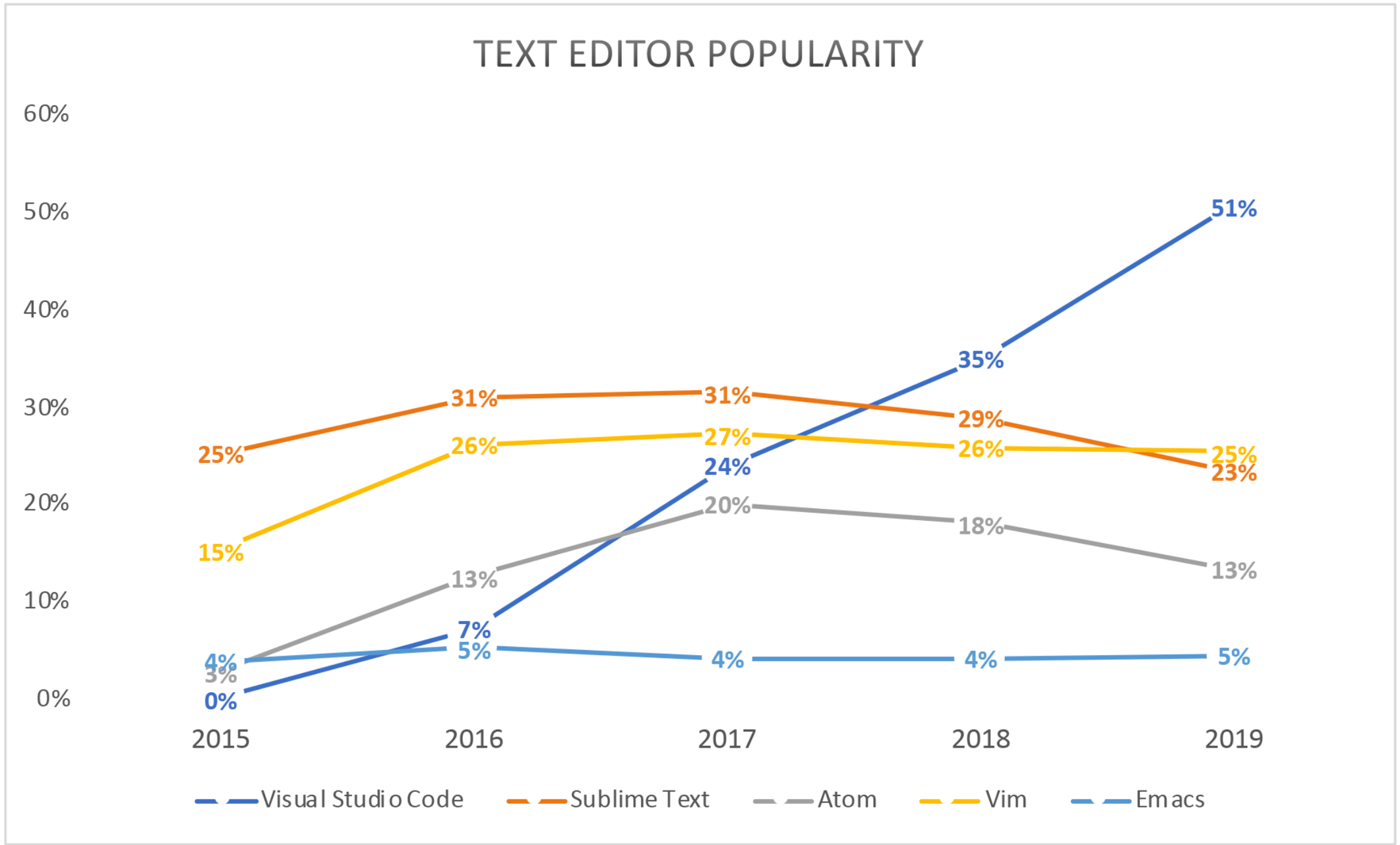
2. What does Vim looks like?

```
 12     void Calibrate();
 11     const CDistriFunc &csmf_;
 10     const CDistriFunc &cphmf_;
  9     vector<double> hm_edges_;
  8     vector<vector<LinearInterp>> fp_;
  7 };
  6
  5 class QuenchFrac
  4 {
  3    public:
  2     QuenchFrac(const vector<double> &sm_edges, int num_snaps);
  1     QuenchFrac(H5Group &i_grp);
315     void Save(H5Group &o_grp);
  1     void CalcFq();
  2     inline const vector<double> &GetSmEdges() const { return sm_edges_; }
  3     inline const vector<vector<double>> &GetQF() const { return f_q_; }
  4     inline void Add(int snap, double log_sm, bool if_q)
  5     {
  6         if (std::isnan(log_sm) | (log_sm < sm_edges_.front()) | (log_sm >= sm_edges_.back()))
  7             return;
  8         int bin_id = 0;
  9         while (log_sm >= sm_edges_[bin_id + 1])
 10             ++bin_id;
 11         if (if_q)
 12             ++cnt_q_[snap][bin_id];
A  sham_cpp/sham.hpp  10:315 (82% 384L)                    cc ⊘ 0 ⚠ 0  master ⊞0 ⊡0 ⊟0
```

```
  1 #!/usr/bin/env python3
  1 # -*- coding: utf-8 -*-
  2 # File           : sham.py
  3 # Author         : Kai Wang <wkcosmology@gmail.com>
  4 # Date           : 09.07.2021
  5 # Last Modified By: Kai Wang <wkcosmology@gmail.com>
  6
  7 import h5py
  8 import numpy as np
  9 import pandas as pd
 10 from scipy.interpolate import interp1d
 11
 12
 13 def snap2z(snap):
 14     snap2z_anch = [
 15         2, 12, 3, 11, 4, 10, 6, 9, 8, 8, 11, 7, 13, 6, 17, 5, 21, 4, 25, 3, 33, 2, 40, 1.5, 50, 1, 59, 0.7, 67, 0.5, 72,
 16         0.4, 78, 0.3, 84, 0.2, 91, 0.1, 99, 0
 17     ]
 18     snap2z_anch = np.array(snap2z_anch).reshape(-1, 2)
 19     z = np.interp(snap, snap2z_anch[:, 0], snap2z_anch[:, 1])
 20     return z
 21
 22
 23 class MultiKeyMap:
B  script/sham.py
Enter path >> /Users/wangk/Documents/Project/14.ConnectGalaxyInPC
```

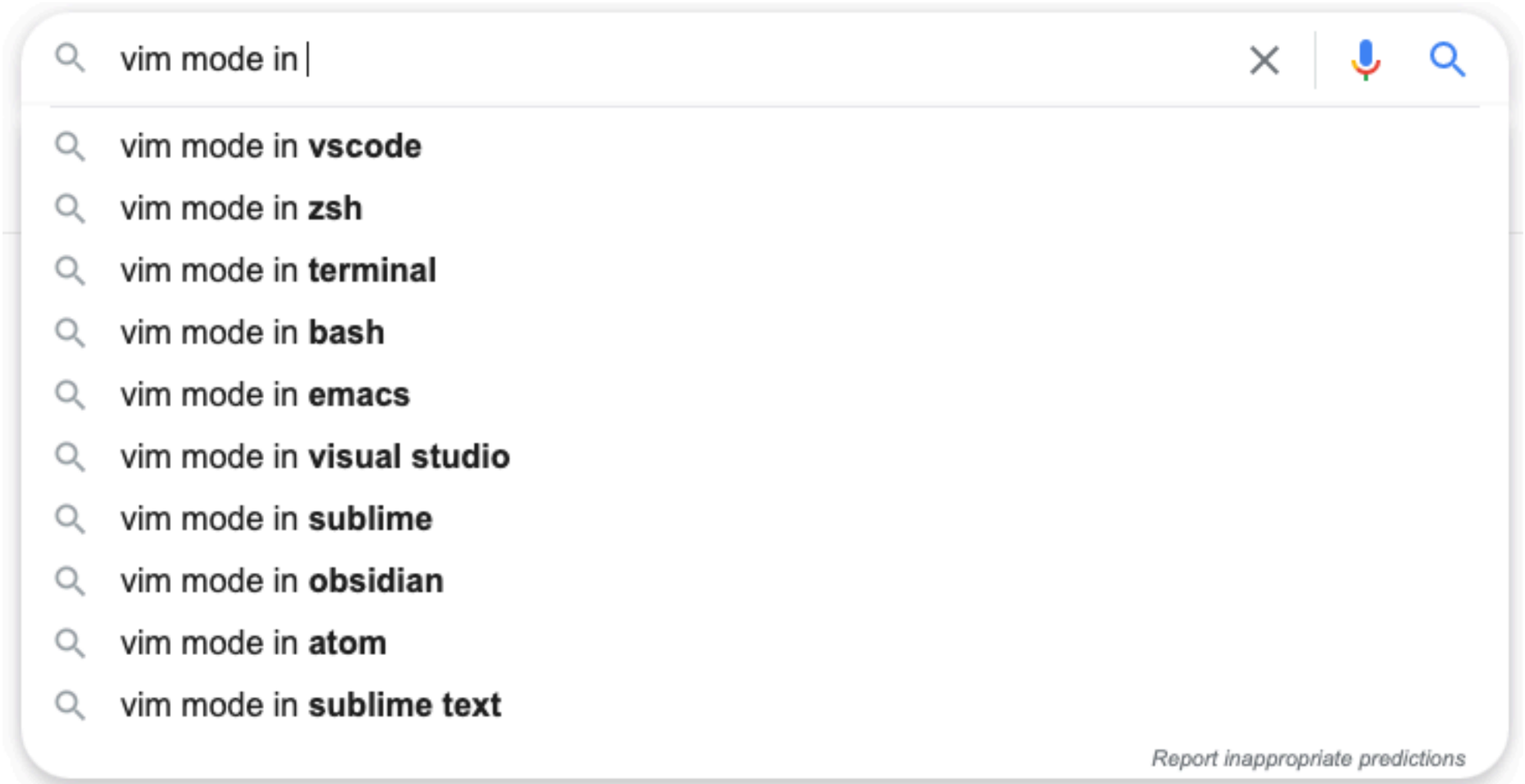
3. Why do I like Vim?

- It is old → It is stable



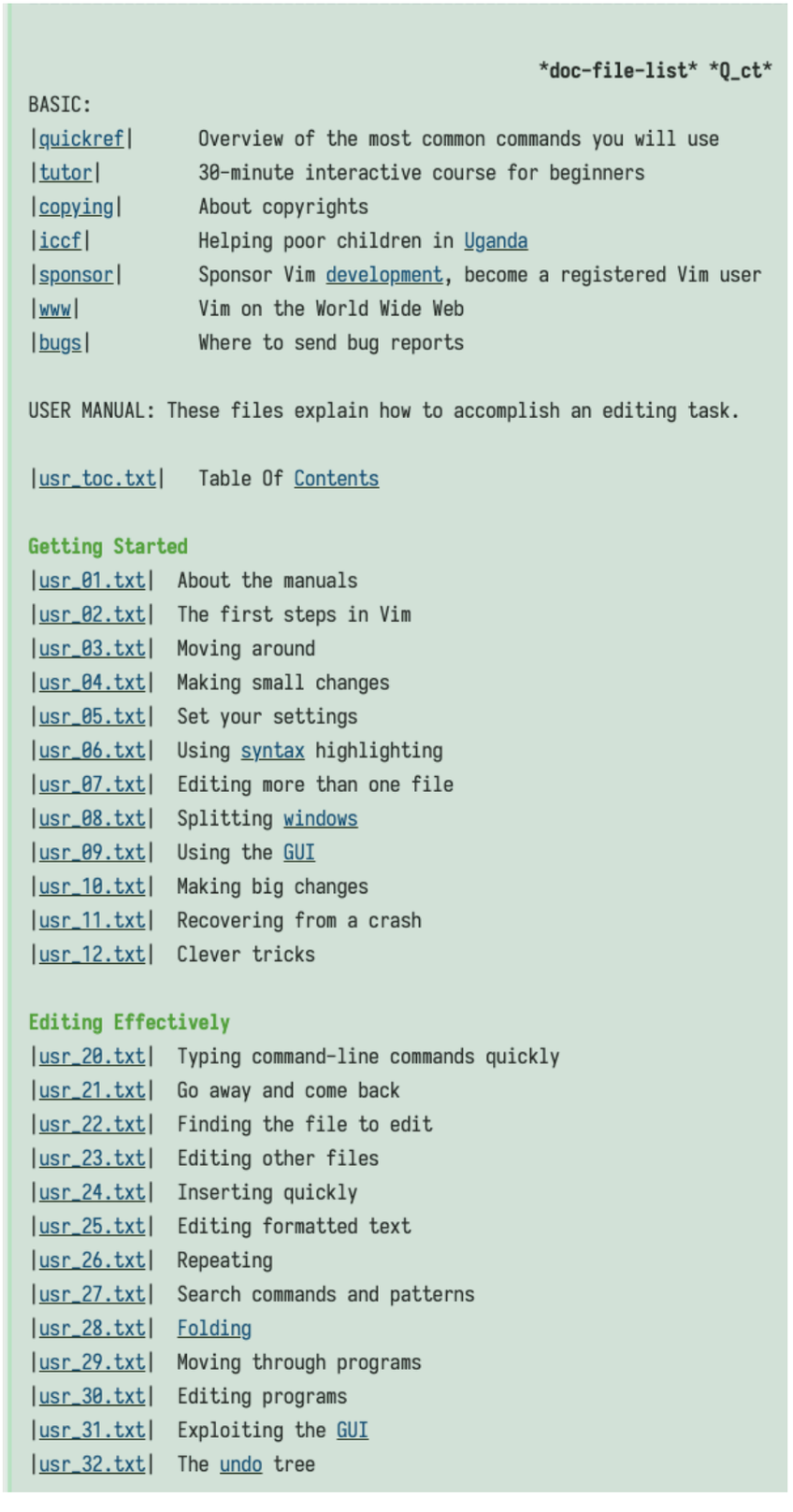
3. Why do I like Vim?

- It is everywhere!



3. Why do I like Vim?

- It is well-documented



doc-file-list *Q_ct*

BASIC:

|quickref| Overview of the most common commands you will use
|tutor| 30-minute interactive course for beginners
|copying| About copyrights
|iccf| Helping poor children in Uganda
|sponsor| Sponsor Vim development, become a registered Vim user
|www| Vim on the World Wide Web
|bugs| Where to send bug reports

USER MANUAL: These files explain how to accomplish an editing task.

|usr_toc.txt| Table Of Contents

Getting Started

|usr_01.txt| About the manuals
|usr_02.txt| The first steps in Vim
|usr_03.txt| Moving around
|usr_04.txt| Making small changes
|usr_05.txt| Set your settings
|usr_06.txt| Using syntax highlighting
|usr_07.txt| Editing more than one file
|usr_08.txt| Splitting windows
|usr_09.txt| Using the GUI
|usr_10.txt| Making big changes
|usr_11.txt| Recovering from a crash
|usr_12.txt| Clever tricks

Editing Effectively

|usr_20.txt| Typing command-line commands quickly
|usr_21.txt| Go away and come back
|usr_22.txt| Finding the file to edit
|usr_23.txt| Editing other files
|usr_24.txt| Inserting quickly
|usr_25.txt| Editing formatted text
|usr_26.txt| Repeating
|usr_27.txt| Search commands and patterns
|usr_28.txt| Folding
|usr_29.txt| Moving through programs
|usr_30.txt| Editing programs
|usr_31.txt| Exploiting the GUI
|usr_32.txt| The undo tree

Tuning Vim

|usr_40.txt| Make new commands
|usr_41.txt| Write a Vim script
|usr_42.txt| Add new menus
|usr_43.txt| Using filetypes
|usr_44.txt| Your own syntax highlighted
|usr_45.txt| Select your language

REFERENCE MANUAL: These files explain every detail of Vim. *reference_toc*

General subjects

|intro.txt| general introduction to Vim; notation used in help files
|nvim.txt| Transitioning from Vim
|help.txt| overview and quick reference (this file)
|helphelp.txt| about using the help files
|index.txt| alphabetical index of all commands
|help-tags| all the tags you can jump to (index of tags)
|tips.txt| various tips on using Vim
|message.txt| (error) messages and explanations
|develop.txt| development of Nvim
|debug.txt| debugging Vim itself
|uganda.txt| Vim distribution conditions and what to do with your money

Basic editing

|starting.txt| starting Vim, Vim command arguments, initialisation
|editing.txt| editing and writing files
|motion.txt| commands for moving around
|scroll.txt| scrolling the text in the window
|insert.txt| Insert and Replace mode
|change.txt| deleting and replacing text
|undo.txt| Undo and Redo
|repeat.txt| repeating commands, Vim scripts and debugging
|visual.txt| using the Visual mode (selecting a text area)
|various.txt| various remaining commands
|recover.txt| recovering from a crash

Advanced editing

|cmdline.txt| Command-line editing
|options.txt| description of all options
|pattern.txt| regexp patterns and search commands
|map.txt| key mapping and abbreviations
|tagsrch.txt| tags and special searches
|windows.txt| commands for using multiple windows and buffers
|tabpage.txt| commands for using multiple tab pages
|spell.txt| spell checking
|diff.txt| working with two to eight versions of the same file
|autocmd.txt| automatically executing commands on an event
|eval.txt| expression evaluation, conditional commands
|fold.txt| hide (fold) ranges of lines
|lua.txt| Lua API
|api.txt| Nvim API via RPC, Lua and VimL

Special issues

|testing.txt| testing Vim and Vim scripts
|print.txt| printing
|remote_plugin.txt| Nvim support for remote plugins

Programming language support

|indent.txt| automatic indenting for C and other languages
|lsp.txt| Language Server Protocol (LSP)
|treesitter.txt| tree-sitter library for incremental parsing of buffers
|diagnostic.txt| Diagnostic framework
|syntax.txt| syntax highlighting
|filetype.txt| settings done specifically for a type of file
|quickfix.txt| commands for a quick edit-compile-fix cycle
|provider.txt| Built-in remote plugin hosts
|ft_ada.txt| Ada (the programming language) support
|ft_ps1.txt| Filetype plugin for Windows PowerShell
|ft_raku.txt| Filetype plugin for Raku
|ft_rust.txt| Filetype plugin for Rust
|ft_sql.txt| about the SQL filetype plugin

Language support

|digraph.txt| list of available digraphs
|mbyte.txt| multibyte text support
|mlang.txt| non-English language support
|rileft.txt| right-to-left editing mode
|arabic.txt| Arabic language support and editing
|hebrew.txt| Hebrew language support and editing
|russian.txt| Russian language support and editing

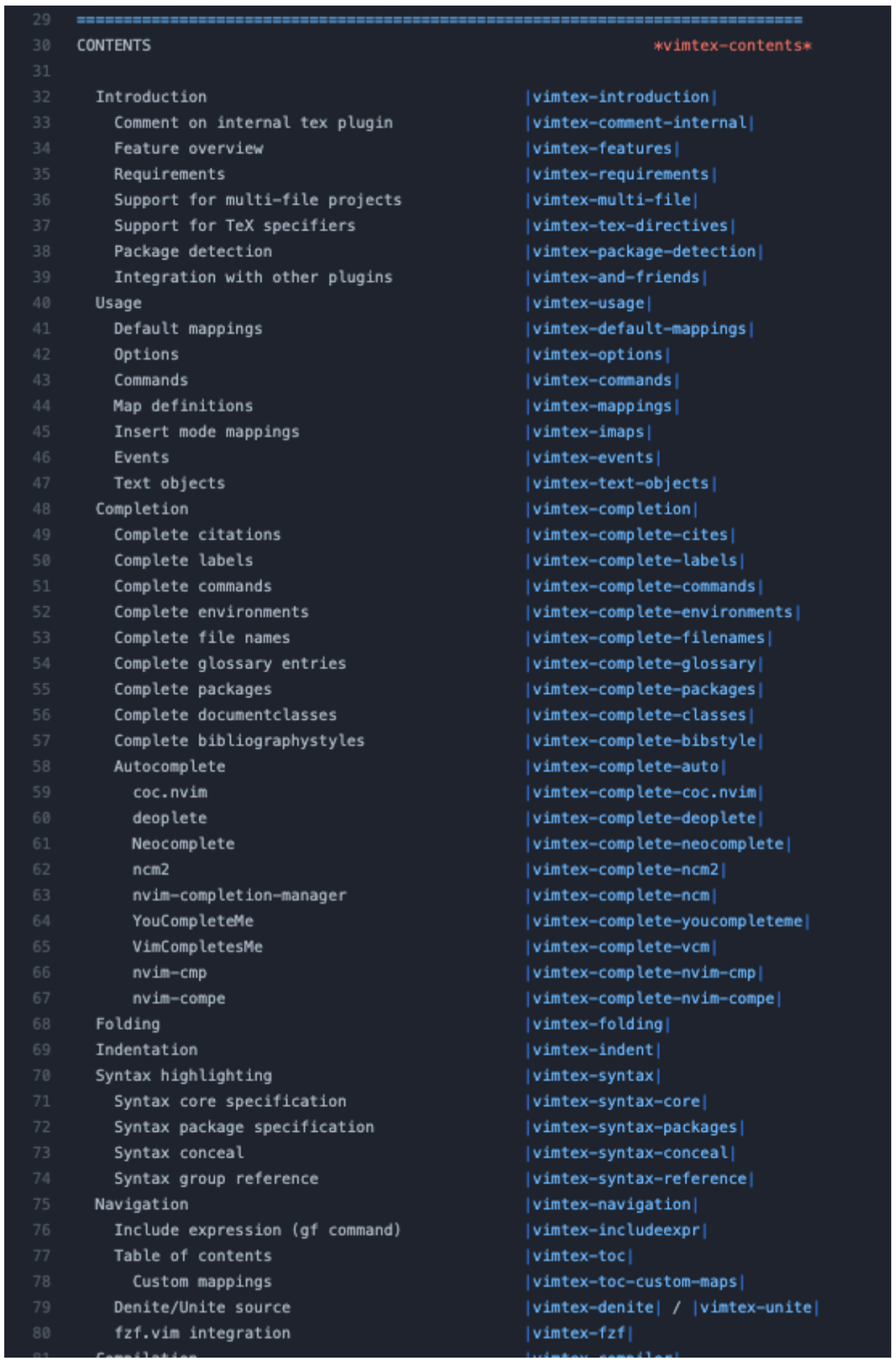
GUI

|gui.txt| Graphical User Interface (GUI)

3. Why do I like Vim?

- Even most of the plugins are well-documented

```
*hop.txt*    For Neovim version 0.5                Last change: 2021 Nov 02

                                    __
                                   / /_  ____  ____
                                  / __ \/ __ \/ __ \
                                 / / / / /_/ / /_/ /
                                /_/ /_/\____/ .___/
                                           /_/
                              · Neovim motions on speed! ·

==============================================================================
CONTENTS                                                        *hop-contents*

  Introduction ····· |hop-introduction|
  Requirements ····· |hop-requirements|
  Usage ····· |hop-usage|
    Commands ····· |hop-commands|
    Lua API ····· |hop-lua-api|
    Jump target API ····· |hop-jump-target-api|
  Configuration ····· |hop-config|
  Extension ····· |hop-extension|
  Highlights ····· |hop-highlights|
  License ····· |hop-license|

==============================================================================
INTRODUCTION                                        *hop* *hop-introduction*

Hop is an “EasyMotion” like plugin allowing you to jump anywhere in a document
with as few keystrokes as possible. It does so by annotating text in your
buffer with |hints|, short string sequences for which each character represents
a key to type to jump to the annotated text. Most of the time, those
sequences’ lengths will be between 1 to 3 characters, making every jump target
in your document reachable in a few keystrokes.

Hop is a complete from-scratch rewrite of EasyMotion, a famous plugin to
enhance the native motions of Vim. Even though EasyMotion is usable in
Neovim, it suffers from a few drawbacks making it not comfortable to use with
Neovim version >0.5 - at least at the time of writing these lines:

- EasyMotion uses an old trick to annotate jump targets by saving the
  contents of the buffer, replacing it with the highlighted annotations and
  then restoring the initial buffer after jump. This trick is dangerous as it
  will change the contents of your buffer. A UI plugin should never do anything
  to existing buffers’ contents.
- Because the contents of buffers will temporarily change, other parts of the
  editor and/or plugins relying on buffer change events will react and will go
  mad. An example is the internal LSP client implementation of Neovim >0.5 or
  its treesitter native implementation. For LSP, it means that the connected
  LSP server will receive a buffer with the jump target annotations… not
  ideal.
```

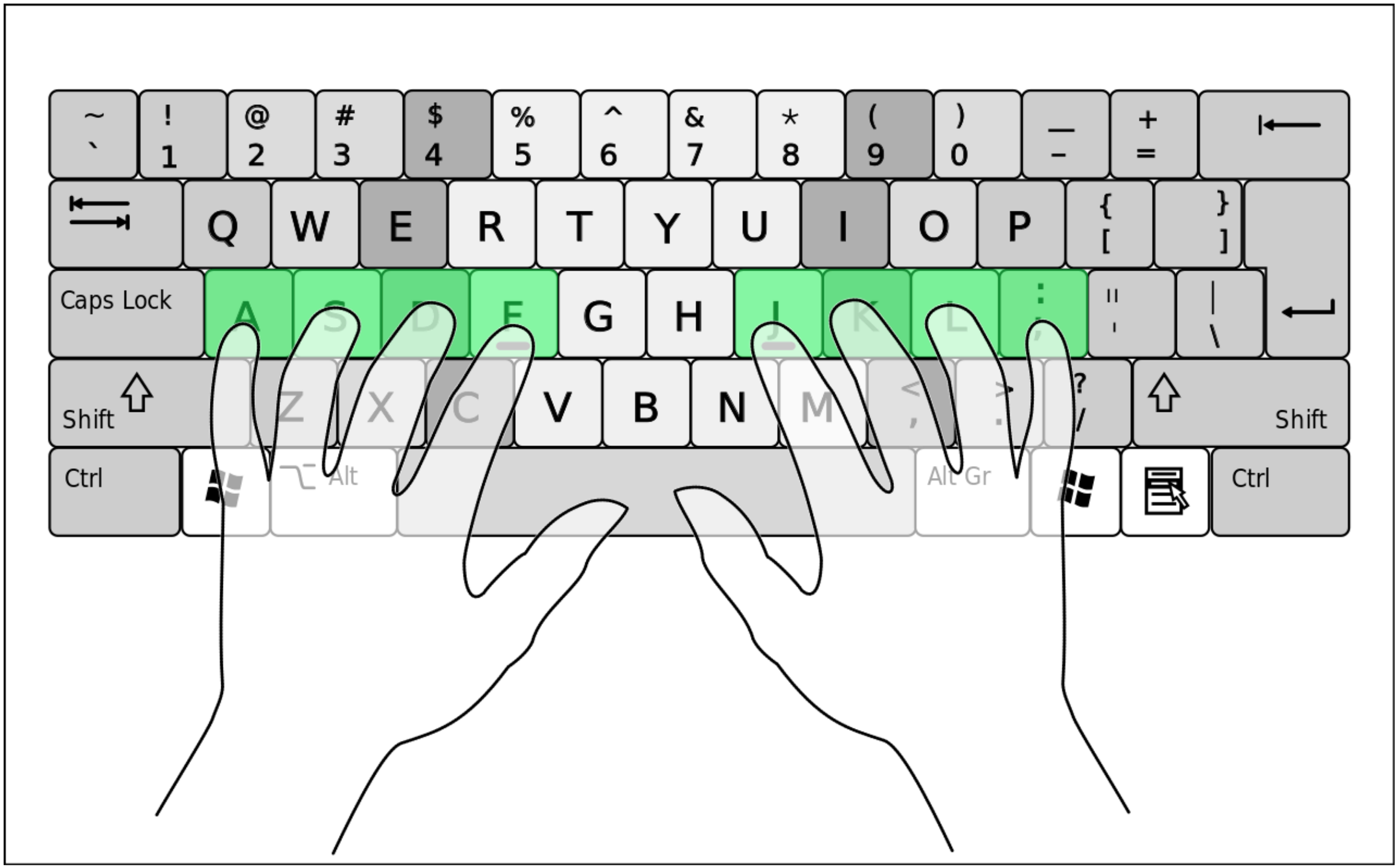


```
==============================================================================
CONTENTS                                                      *vimtex-contents*

  Introduction                          |vimtex-introduction|
    Comment on internal tex plugin      |vimtex-comment-internal|
    Feature overview                    |vimtex-features|
    Requirements                        |vimtex-requirements|
    Support for multi-file projects     |vimtex-multi-file|
    Support for TeX specifiers          |vimtex-tex-directives|
    Package detection                   |vimtex-package-detection|
    Integration with other plugins      |vimtex-and-friends|
  Usage                                 |vimtex-usage|
    Default mappings                    |vimtex-default-mappings|
    Options                             |vimtex-options|
    Commands                            |vimtex-commands|
    Map definitions                     |vimtex-mappings|
    Insert mode mappings                |vimtex-imaps|
    Events                              |vimtex-events|
    Text objects                        |vimtex-text-objects|
  Completion                            |vimtex-completion|
    Complete citations                  |vimtex-complete-cites|
    Complete labels                     |vimtex-complete-labels|
    Complete commands                   |vimtex-complete-commands|
    Complete environments               |vimtex-complete-environments|
    Complete file names                 |vimtex-complete-filenames|
    Complete glossary entries           |vimtex-complete-glossary|
    Complete packages                   |vimtex-complete-packages|
    Complete documentclasses            |vimtex-complete-classes|
    Complete bibliographystyles         |vimtex-complete-bibstyle|
    Autocomplete                        |vimtex-complete-auto|
      coc.nvim                          |vimtex-complete-coc.nvim|
      deoplete                          |vimtex-complete-deoplete|
      Neocomplete                       |vimtex-complete-neocomplete|
      ncm2                              |vimtex-complete-ncm2|
      nvim-completion-manager           |vimtex-complete-ncm|
      YouCompleteMe                     |vimtex-complete-youcompleteme|
      VimCompletesMe                    |vimtex-complete-vcm|
      nvim-cmp                          |vimtex-complete-nvim-cmp|
      nvim-compe                        |vimtex-complete-nvim-compe|
  Folding                               |vimtex-folding|
  Indentation                           |vimtex-indent|
  Syntax highlighting                   |vimtex-syntax|
    Syntax core specification           |vimtex-syntax-core|
    Syntax package specification        |vimtex-syntax-packages|
    Syntax conceal                      |vimtex-syntax-conceal|
    Syntax group reference              |vimtex-syntax-reference|
  Navigation                            |vimtex-navigation|
    Include expression (gf command)     |vimtex-includeexpr|
    Table of contents                   |vimtex-toc|
      Custom mappings                   |vimtex-toc-custom-maps|
    Denite/Unite source                 |vimtex-denite| / |vimtex-unite|
    fzf.vim integration                 |vimtex-fzf|
```

4. Mode in Vim

- **Insert**
  - Press ‘i’, ‘I’, ‘a’, ‘A’, ‘o’, ‘O’ in normal mode to enter
  - Insert, delete, just like any normal editor
- **Normal**
  - Press <Esc> or <Ctrl-[> in insert, visual or command-line mode to enter
  - Each alphabeta key is interpreted differently.
  - dd → delete a line
  - u → undo the last editing
  - yy → yank (copy) a line
  - h, j, k, l → move the cursor left, down, up, right
  - <Ctrl-d> (<Ctrl-u>) → scroll down (up)
  - You can add your own mapping!
  - Space-ff: fuzzy search current project
  - Space-fr: fuzzy search recently opened files
  - Space-sw: fuzzy search the word under cursor in current project
- **Command-Line**
  - press ‘:’ in normal mode to enter
  - :w → write (save)
  - :q → quit
  - :qw → save and quit
  - :%s/vscode/vim/g → substitute ‘vscode’ with ‘vim’ in current buffer
- **Visual**
  - press ‘v’ in normal mode to enter
  - use hjkl or arrow key to move cursor for selection
- **Line-Visual**
  - press <Shift-v> in normal mode to enter
  - use jk or arrow key to select lines
- **Block-Visual**
  - press <Ctrl-v> in normal mode to enter
  - use hjkl or arrow key to select blocks

5.1 Keep your fingers in home row and type blindly



5.2 Use . and macro to repeat

5.3 Text object & Operator + Motion = Action

- Text object
  - ‘iw’: a word
  - ‘iW’: a WORD
  - ‘aw’: a word with leading OR trailing space
  - ‘ib’: inside a bracket
  - ‘ab’: inside a bracket AND the bracket
  - ‘i(’ or ‘i)’, ’i[’ or ‘i]’ and others...
- Operator + Motion = Action
  - ‘y3j’ → yank 3 lines below (include current)
  - ‘d5k’ → delete 5 lines below (include current)
  - ‘daw’ → delete a word with leading OR trailing space
  - ‘ciw’ → delete a word and enter insert mode
  - ‘cib’ → delete anything inside a bracket and enter insert mode
  - ...

6. Configuration

:help vimrc

```
                                                        *vimrc* *exrc*
A file that contains initialization commands is called a "vimrc" file.
Each line in a vimrc file is executed as an Ex command line.  It is
sometimes also referred to as "exrc" file.  They are the same type of
file, but "exrc" is what Vi always used, "vimrc" is a Vim specific
name.  Also see |vimrc-intro|.

Places for your personal initializations:
        Unix            $HOME/.vimrc or $HOME/.vim/vimrc
        MS-Windows      $HOME/_vimrc, $HOME/vimfiles/vimrc
                        or $VIM/_vimrc
        Amiga           s:.vimrc, home:.vimrc, home:vimfiles:vimrc
                        or $VIM/.vimrc
        Haiku           $HOME/config/settings/vim/vimrc

The files are searched in the order specified above and only the first
one that is found is read.

RECOMMENDATION: Put all your Vim configuration stuff in the
$HOME/.vim/ directory ($HOME/vimfiles/ for MS-Windows).  That makes it
easy to copy it to another system.

If Vim was started with "-u filename", the file "filename" is used.
All following initializations until 4. are skipped.  $MYVIMRC is not
set.
"vim -u NORC" can be used to skip these initializations without
reading a file.  "vim -u NONE" also skips loading plugins.  |-u|

If Vim was started in Ex mode with the "-s" argument, all following
initializations until 4. are skipped.  Only the "-u" option is
interpreted.
```

:help config

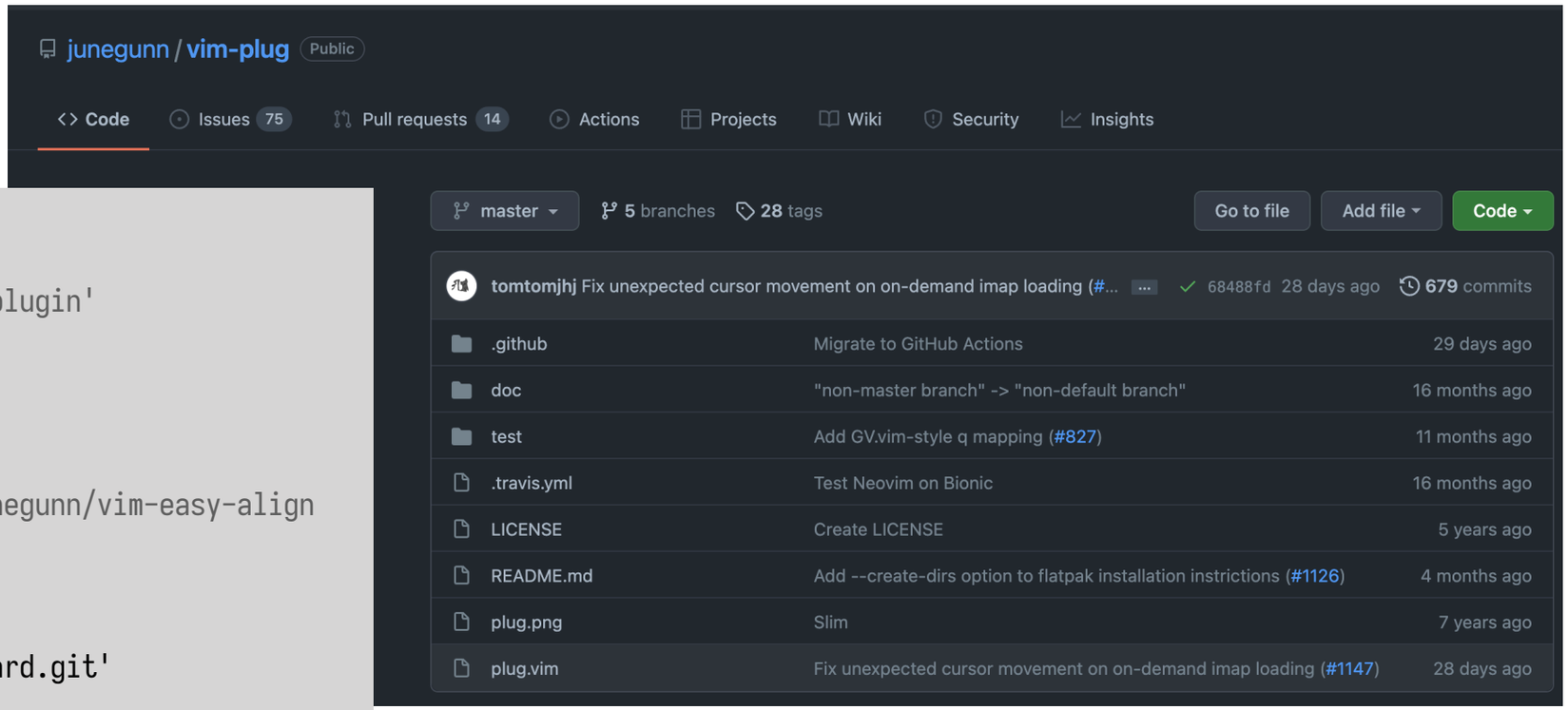
```
5. Load user config (execute Ex commands from files, environment, …).
        $VIMINIT environment variable is read as one Ex command line (separate
        multiple commands with '|' or <NL>).
                                                        config init.vim init.lua vimrc exrc
        A file containing initialization commands is generically called
        a "vimrc" or config file.  It can be either Vimscript ("init.vim") or
        Lua ("init.lua"), but not both. E5422
        See also vimrc-intro and base-directories.

        The config file is located at:
        Unix                    ~/.config/nvim/init.vim         (or init.lua)
        Windows                 ~/AppData/Local/nvim/init.vim   (or init.lua)
        $XDG_CONFIG_HOME        $XDG_CONFIG_HOME/nvim/init.vim  (or init.lua)

        If Nvim was started with "-u {file}" then {file} is used as the config
        and all initializations until 5. are skipped. $MYVIMRC is not set.
        "nvim -u NORC" can be used to skip these initializations without
        reading a file.  "nvim -u NONE" also skips plugins and syntax
        highlighting.  -u

        If Nvim was started with -es all initializations until 5. are
        skipped.
                                                        system-vimrc sysinit.vim
    a. The system vimrc file is read for initializations.  If
       nvim/sysinit.vim file exists in one of $XDG_CONFIG_DIRS, it will be
       used.  Otherwise the system vimrc file is used. The path of this file
       is given by the :version command.  Usually it's "$VIM/sysinit.vim".

                                                        VIMINIT EXINIT $MYVIMRC
```

- Vim has its only scripting language: Vimscript or VimL → Slow and hard to learn
- Neovim has natively integrate with a popular language: Lua

7. Plugins

- Vim is powerful, and it will be more powerful with plugins
- First, we need a plugin to manage the plugins: https://github.com/junegunn/vim-plug

```
" Specify a directory for plugins
" - For Neovim: stdpath('data') . '/plugged'
" - Avoid using standard Vim directory names like 'plugin'
call plug#begin('~/.vim/plugged')

" Make sure you use single quotes

" Shorthand notation; fetches https://github.com/junegunn/vim-easy-align
Plug 'junegunn/vim-easy-align'

" Any valid git URL is allowed
Plug 'https://github.com/junegunn/vim-github-dashboard.git'

" Multiple Plug commands can be written in a single line using | separators
Plug 'SirVer/ultisnips' | Plug 'honza/vim-snippets'

" On-demand loading
Plug 'scrooloose/nerdtree', { 'on':  'NERDTreeToggle' }
Plug 'tpope/vim-fireplace', { 'for': 'clojure' }

" Initialize plugin system
call plug#end()
```



Installation

Download plug.vim and put it in the "autoload" directory.

```
/Users/wangk/.vim/autoload
```