# Contribution Guide

This document offers a set of guidelines for contributing to VSCodeVim.

These are just guidelines, not rules; use your best judgment and feel free to propose changes to this document.

If you need help, drop by on [Slack](https://vscodevim.herokuapp.com/).

Thanks for helping us in making VSCodeVim better! :clap:

## Submitting Issues

The [GitHub issue tracker](https://github.com/VSCodeVim/Vim/issues) is the preferred channel for tracking bugs and enhancement suggestions.

When creating a new bug report do:

- Search against existing issues to check if somebody else has already reported your problem or requested your idea

- Fill out the issue template.

## Submitting Pull Requests

Pull requests are \_awesome\_.

If you're looking to raise a PR for something which doesn't have an open issue, consider creating an issue first.

When submitting a PR, please fill out the template that is presented by GitHub when a PR is opened.

## First Time Setup

1. Install prerequisites:

- latest [Visual Studio Code](https://code.visualstudio.com/)

- [Node.js](https://nodejs.org/) v12.0.0 or higher

- \_Optional\_: [Docker Community Edition](https://store.docker.com/search?type=edition&offering=community) ?

1. In a terminal:

```bash

# fork and clone the repository

git clone git@github.com:<YOUR-FORK>/Vim.git

cd Vim

# Install the dependencies

npm install

# Open in VSCode

code .

# Choose the "Build, Run Extension" in the dropdown of VSCode's

# debug tab to build and run the extension.

# Or run tests by selecting the appropriate drop down option

# Alternatively, build and run tests through gulp and npm scripts

npx gulp build # build

npm test # test (must close all instances of VSCode)

# Only available if Docker is installed and running

npx gulp test # run tests inside Docker container

npx gulp test --grep testSuite # run only tests/suites filtered by js regex inside container

```

## Code Architecture

The code is split into two parts:

- ModeHandler - Vim state machine

- Actions - 'actions' which modify the state

### Actions

Actions are all currently stuffed into actions.ts (sorry!). There are:

- `BaseAction` - the base Action type that all Actions derive from.

- `BaseMovement` - A movement (e.g.`w`, `h`, `{`, etc.) \_ONLY\_ updates the cursor position or returns an `IMovement`, which indicates a start and stop. This is used for movements like `aw` which may actually start before the cursor.

- `BaseCommand` - Anything which is not just a movement is a Command. That includes motions which also update the state of Vim in some way, like `\*`.

At one point, I wanted to have actions.ts be completely pure (no side effects whatsoever), so commands would just return objects indicating what side effects on the editor they would have. This explains the giant switch in handleCommand in ModeHandler. I now believe this to be a dumb idea and someone should get rid of it.

### The Vim State Machine

Consists of two data structures:

- `VimState` - this is the state of Vim. It's what actions update.

- `RecordedState` - this is temporary state that will reset at the end of a change.

#### How it works

1. `handleKeyEventHelper` is called with the most recent keypress.

2. `Actions.getRelevantAction` determines if all the keys pressed so far uniquely specify any action in actions.ts. If not, we continue waiting for keypresses.

3. `runAction` runs the action that was matched. Movements, Commands and Operators all have separate functions that dictate how to run them - `executeMovement`, `handleCommand`, and `executeOperator` respectively.

4. Now that we've updated VimState, we run `updateView` with the new VimState to "redraw" VSCode to the new state.

#### vscode.window.onDidChangeTextEditorSelection

This is my hack to simulate a click event based API in an IDE that doesn't have them (yet?). I check the selection that just came in to see if it's the same as what I thought I previously set the selection to the last time the state machine updated. If it's not, the user \_probably\_ clicked. (But she also could have tab completed!)

## Release

To push a release:

```bash

npx gulp release --semver [SEMVER] --gitHubToken [TOKEN]

git push --follow-tags

```

The above Gulp command will:

1. Bump the package version based off the semver supplied. Supported values: patch, minor, major.

2. Create a changelog using [github-changelog-generator](https://github.com/github-changelog-generator/github-changelog-generator).

3. Create a Git commit with the above changes.

4. Create a Git tag using the new package version.

In addition to building and testing the extension, when a tag is applied to the commit, the CI server will also create a GitHub release and publish the new version to the Visual Studio marketplace.

## Troubleshooting

### Visual Studio Code Slowdown

If you notice a slowdown and have ever run `npm test` in the past instead of running tests through VSCode, you might find a `.vscode-test/` folder, which VSCode is continually consuming CPU cycles to index. Long story short, you can speed up VSCode by:

```bash

$ rm -rf .vscode-test/

```

## Styleguide

Please try your best to adhere to our [style guidelines](https://github.com/VSCodeVim/Vim/blob/master/STYLE.md).