Navigating Ruby Files with Vim - Installation Instructions

In this document, you'll find links to each of the tools mentioned in the screencast series, *Navigating Ruby Files with Vim*.

Recommended Vim plugins

Here is the complete list of recommended Vim plugins:

- vim-ruby
- matchit
- textobj-rubyblock
- vim-bundler
- vim-rake
- vim-fugitive
- vim-unimpaired
- vim-rails

You'll find more details about each plugin below.

vim-ruby

The vim-ruby bundle is included in the standard Vim distribution, so you probably have it already. Even so, I'd recommend getting it from github to ensure you get the latest version.

The vim-ruby bundle provides motions and text objects for operating on classes, modules, and method definitions (as demonstrated in Part 1). The vim-ruby bundle is also responsible for setting Vim's `path` to include all directories listed in Ruby's `$LOAD_PATH` (as discussed in Part 2).

Also, the vim-ruby plugin configures the `suffixesadd` setting, and adjusts the behavior of the `gf` command so that you can use it anywhere on a line that uses `require` to load another Ruby library.

Matchit

The matchit.vim plugin is usually included in the Vim standard distribution, so you should have it installed already. However, the plugin has to be activated by running:

```
runtime macros/matchit.vim
```

Add that line to your `vimrc` file. Enabling the matchit plugin will enhance Vim's built-in `%` command, making it possible to jump between pairs of Ruby keywords, such as `class`, `module`, `def`, `if`, `do`, and their corresponding `end` (as demonstrated in Part 1).

textobj-rubyblock

To get the textobj-rubyblock plugin working, you also have to install textobj-user.

The textobj-rubyblock plugin adds two text objects: `ir` and `ar`, which allow you to operate on generic rubyblocks, as demonstrated in Part 1.

vim-bundler

Get the latest edition of the vim-bundler plugin from github. The vim-bundler plugin automatically configures the `path` and `tags` settings to include all libraries referenced in your `Gemfile`.

Having a properly configured `path` lets you use both `gf` and `:find` commands (as demonstrated in Part 2). Having the `tags` option configured allows you to jump to definitions with `<C-]>` and `:tag` (as demonstrated in Part 3).

To get the jump to definition commands working, you also have to ensure that your bundled gems have been indexed with ctags. See the section *Trigger ctags when installing a rubygem* below for more details.

vim-rake

Get the latest edition of the vim-rake plugin from github.

When working on a Ruby project that follows the conventional directory layout for a rubygem, the vim-rake plugin automatically configures the `path` setting to include the `lib` and `ext` directories (as demonstrated in Part 2).

vim-fugitive

Get the latest edition of the vim-fugitive plugin from github.

If your project is using git for source control, then the fugitive plugin will configure Vim to look in `.git/tags` for a ctags index. For instructions on how to automatically generate the `.git/tags` file, refer to the *Trigger ctags via git hooks* section below.

vim-unimpaired

Get the latest edition of the vim-unimpaired plugin from github.

The unimpaired plugin provides a set of commands that are generally useful, no matter what kind of project you are working on. In Part 3, I demonstrate the use of `[t`, `]t`, etc. commands for navigating the list of matching tags.

vim-rails

Get the latest edition of the vim-rails plugin from github.

The rails.vim plugin is not covered in this screencast series, but you can find more information about it in another thoughtbot screencast: Vim for Rails developers.

Tools outside of Vim

I recommend installing each of the following external tools to get full ctags support for your ruby projects:

- exuberant ctags
- git-hook scripts
- gem-ctags
- rbenv-ctags

Read on for more details about each tool. You can find fuller explanations in Part 3 of the screencast series.

Exuberant ctags

On os x, ctags can be installed via homebrew:

```
brew install ctags
```

On Ubuntu, the package is called `exuberant-ctags`:

```
sudo apt-get install exuberant-ctags
```

Other Linux distributions may call the package `ctags` or `exuberant-ctags`.

Trigger ctags via git hooks

Follow the instructions in Tim Pope's blog post: Effortless Ctags with Git. That sets up a few git hooks so that ctags will automatically index your project whenever you checkout, commit, merge or rebase.

Trigger ctags when installing a rubygem

Install gem-ctags as a standard rubygem:

```
gem install gem-ctags
```

Now you run ctags to index every rubygem installed on your system by running:

```
gem ctags
```

You won’t have to run that command again though. From now on, each time you install a rubygem it will be automatically indexed by ctags.

Trigger ctags on rbenv managed Rubies

If you’re using rbenv to manage your ruby environment, you should install Tim Pope’s rbenv-ctags plugin.

This plugin provides a convenient command for running ctags to index rubies that are managed by rbenv:

```
rbenv ctags
```

If you use ruby-build to install versions of ruby, then this plugin will automatically index your rubies as they are installed.