Vim and Ctags

October 31, 2012

Combining vim with ctags yields a powerful combination for working with large or unfamiliar codebases.

Ctags is a tool that will sift through your code, indexing methods, classes, variables, and other identifiers, storing the index in a `tags` file. The tags file contains a single tag per line. Depending on command line arguments and the language ctags is run against, a lot of information can be obtained from this index.

Ctags currently supports 41 programming languages, and it's relatively easy to add definitions for more.

Ctags makes it much easier to navigate a larger project, particularly if the code you're working with is unfamiliar. If you're unsure of what a method does or how it's supposed to be called, you can jump straight to it's definition. If you're in the downward spiral of a 500+ line Perl script and want to know where a variable was defined three hours ago, you can jump right back to it. And afterwards, you can jump right back to where you were working.

Installing ctags

You can install ctags using a package manager.[1] If you're on OS X, use homebrew:

```
brew install ctags
```

OS X comes with a `ctags` executable, but it's not exuberant-ctags, and is missing most of the useful features.

Using ctags

If you're currently sitting in the directory you want to index, just run:

```
ctags -R .
```

Ctags will walk through the directory recursively, tagging all source files it encounters. For very large projects, this might take a while, but normally it's pretty fast.

I normally don't like having a `tags` file in plain sight in the source directory, so I keep it a little bit more hidden, in the `.git` folder:

```
ctags -R -f ./.git/tags .
```

This can be a pain in the ass to run regularly, so you might like to bind it to a vim keyboard shortcut so you can run it every so often, or add some git hooks to run ctags every time you check out, commit, or fetch with git.

Using ctags from vim

If you've already run ctags in the current project folder, vim will automatically pick up your tags file (even in the `.git` directory), and you can start using it right away. Take this bit of Ruby code, for example:

```
class TestCase
  def assert_equal(expected, value)
```

```
    # do_stuff_with_args
  end
end

class ModelTest < TestCase
  assert_equal true, model_function
end
```

If you put your cursor over `ModelTest`'s `assert_true` call in normal mode and press `<C-]>`, vim will jump to `assert_true`'s definition in the `TestCase` class, even if they're in different files. You can continue down this rabbit hole, if you choose, and when you're done, press `<C-t>` to climb back up the tree.[2]

You can also go directly to a tag's definition by entering one of the following in vim's command mode:

```
:tag function_name
:ta function_name
```

These commands will also accept regular expressions, so, for example, `:tag /^asserts_*` would find all tags that start with 'asserts_'. By default vim will jump to the first result, but a number of commands can be used to sort through the list of tags:

- `:ts` or `:tselect` shows the list
- `:tn` or `:tnext` goes to the next tag in that list
- `:tp` or `:tprev` goes to the previous tag in that list
- `:tf` or `:tfirst` goes to the first tag of the list
- `:tl` or `:tlast` goes to the last tag of the list

To show the tags you've traversed since you opened vim, run `:tags`.

Vim + Ctags + Ctrlp

If you're using the Ctrlp plugin for vim, running `:CtrlPTag` will let you search through your `tags` file and jump to where tags are defined. Very useful when you need to jump around a project in a hurry.

It's also handy to bind this to a keyboard shortcut. I use this in my `~/.vimrc`:

```
nnoremap <leader>. :CtrlPTag<cr>
```

Vim + Ctags + Tagbar

Tagbar is another useful vim plugin for working with a `tags` file. Install it, and map a key to it (I use `,b`):

```
nnoremap <silent> <Leader>b :TagbarToggle<CR>
```

When the tagbar is toggled, it will pop up on the right side of the vim window and show the tags picked up by `ctags` for the current file, organized by tag type, e.g. function, variable, class, etc. This effectively gives you a birds-eye view of the code you're working on.

.ctags

You can put extra configuration for ctags in a `~/.ctags` file, and ctags will use the contents as arguments every time it's run. You can use this to establish basic options, define regular expressions for additional languages, and set options for specific languages.

For example, here's my `~/.ctags` file:

```
# Basic options
--recurse=yes
--tag-relative=yes
--exclude=.git

# Regex for Clojure
--langdef=Clojure
--langmap=Clojure:.clj
--regex-clojure=/\([ \t]*create-ns[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/n,namespac
--regex-clojure=/\([ \t]*def[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/d,definition/
--regex-clojure=/\([ \t]*defn-?[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/f,function/
--regex-clojure=/\([ \t]*defmacro[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/m,macro/
--regex-clojure=/\([ \t]*definline[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/i,inline/
--regex-clojure=/\([ \t]*defmulti[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/a,multimeth
--regex-clojure=/\([ \t]*defmethod[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/b,multimet
--regex-clojure=/\([ \t]*defonce[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/c,definitior
--regex-clojure=/\([ \t]*defstruct[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/s,struct/
--regex-clojure=/\([ \t]*intern[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/v,intern/
--regex-clojure=/\([ \t]*ns[ \t]+([-[:alnum:]*+!_:\/.?]+)/\1/n,namespace/

# PHP
--langmap=php:.engine.inc.module.theme.install.php --PHP-kinds=+cf-v
```

1. It might be available as `exuberant-ctags`, depending on your package manager. [return]
2. For more on this stuff, run `:help tags` and `:help CTRL-]`. [return]