[Coming Home to Vim](http://stevelosh.com/blog/2010/09/coming-home-to-vim/)

Posted 6 years ago on September 20, 2010.

I’m a programmer. I work with text files for 6-12 hours every weekday so I care about the text editor I use. If switching to a different editor can increase my efficiency by even 10% it would save a good chunk of my time and let me get back to making cool things.

I don’t buy the “you’re thinking 90% of the time and only typing 10% of the time, so your editor doesn’t really matter” argument. Even if the premise is true, the conclusion is wrong.

If I think for 10 minutes and then start typing, I want the typing to take the shortest time possible so I can get back to thinking. Any time I spend typing is an interruption that I want to minimize so I can keep my train of thought.

I recently started using [Vim](http://www.vim.org/) as my primary editor. As I’m adjusting I’m finding a lot of [the](http://jeetworks.org/node/89) [blog](http://www.viemu.com/a-why-vi-vim.html) [posts](http://wekeroad.com/2010/07/29/vim-is-your-daddy/) [people](http://yehudakatz.com/2010/07/29/everyone-who-tried-to-convince-me-to-use-vim-was-wrong/) [have](http://nvie.com/posts/how-i-boosted-my-vim/) [written](http://stackoverflow.com/questions/1218390/what-is-your-most-productive-shortcut-with-vim/1220118#1220118) [about](http://weblog.jamisbuck.org/2008/10/10/coming-home-to-vim) Vim very helpful, so I’m hoping this post will help people too.

* + [Some Background About Me](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#some-background-about-me)
  + [Why I Switched to TextMate](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#why-i-switched-to-textmate)
  + [Why I Came Back to Vim](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#why-i-came-back-to-vim)
  + [Core Differences](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#core-differences)
  + [Vim’s “Feeling”](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#vim-s-feeling)
    - [A “Language” of Text Editing](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#a-language-of-text-editing)
    - [The Physics of Text](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#the-physics-of-text)
    - [“Programming” Your Editing](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#programming-your-editing)
  + [Getting Started](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#getting-started)
  + [Making Vim More Useful](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#making-vim-more-useful)
    - [Important .vimrc Lines](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#important-vimrc-lines)
    - [Using the Leader](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#using-the-leader)
    - [Quicker Escaping](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#quicker-escaping)
    - [Working With Split Windows](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#working-with-split-windows)
  + [Aesthetics](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#aesthetics)
  + [Bundles I Use](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#bundles-i-use)
    - [Pathogen](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#pathogen)
    - [PeepOpen](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#peepopen)
    - [NERDTree](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#nerdtree)
    - [NERDCommenter](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#nerdcommenter)
    - [Ack](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#ack)
    - [Snipmate](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#snipmate)
    - [Sparkup](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#sparkup)
    - [Yankring](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#yankring)
    - [Surround (and Repeat)](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#surround-and-repeat)
    - [Slime](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#slime)
    - [Scratch](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#scratch)
    - [Rainbow Parentheses](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#rainbow-parentheses)
  + [Things I Want](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#things-i-want)
    - [HTML Indentation that Doesn’t Suck](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#html-indentation-that-doesn-t-suck)
    - [Python Support for Slime](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#python-support-for-slime)
    - [Gundo](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#gundo)
    - [A Mercurial Version of Fugitive](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#a-mercurial-version-of-fugitive)
  + [Overall Thoughts](http://stevelosh.com/blog/2010/09/coming-home-to-vim/#overall-thoughts)

Some Background About Me

I used Vim exclusively from about 2002 to 2005. I never really learned much about it and mostly used it as a glorified [nano](http://en.wikipedia.org/wiki/Nano_(text_editor)).

In 2006 I decided it was time for a change. I wanted an editor that “fit” with OS X. I tried [SubEthaEdit](http://www.codingmonkeys.de/subethaedit/) for a while and it was kind of cool. The collaborative editing was more polished than any other version I’ve seen, though I didn’t have much use for it.

As cool as SubEthaEdit was, I always felt like there was a better editor out there for me. Then I found [TextMate](http://macromates.com/).

At the time I thought TextMate was amazing (and it really was). It *looked* like a Mac application. It was super easy to learn (all the OS X text movement commands just worked). There were bundles for everything.

I still used Vim occasionally (like when I had a job and only had a Windows machine), but for the most part I was a TextMate kid.

After using TextMate for four years I decided it was time for another change. Development on TextMate has stalled while TextMate Forever (err, TextMate 2) is in the works. I started thinking about Vim (and reading blog posts by others about how they switched back) and decided to give it another try.

A few months ago I bought [Learning the vi and Vim Editors](http://amzn.com/059652983X) and went cold-turkey with Vim as my main text editor. I haven’t looked back since.

Why I Switched to TextMate

As I mentioned before, there were a few reasons I switched to TextMate in the first place.

First of all: it looks and acts like a real OS X application. You can drag files onto the icon to open them, it supports all the default OS X text movement commands, and the chrome looks like it belongs on OS X. I don’t know if[MacVim](http://code.google.com/p/macvim/) was around at the time, but if it was I never found it.

This matters. The effort required to get started with a new editor is dramatically reduced if that editor supports all the conventions of the operating system you use every day.

It also has a vibrant community of people writing bundles for it, so there was support for almost anything I wanted to use.

Why I Came Back to Vim

I came back to Vim for a number of reasons.

First of all: I started reading about and desiring features that didn’t seem to be coming to TextMate any time soon. The biggest of those was split windows. I saw the appeal of split windows right away and now that I use them I can’t imagine not having them.

Another reason is version control. A while ago I started keeping my [dotfiles](http://bitbucket.org/sjl/dotfiles/)in a [Mercurial](http://hg-scm.org/) repository which makes it extremely easy to set up new machines. Vim keeps all of its configuration in two simple places: a simple~/.vimrc file and a ~/.vim directory full of plain text files.

Vim’s plain text configuration files and plugins are very easy to version control. They diff well, unlike TextMate’s multiple Bundles directories and ugly XML file format.

Vim’s community has been around far longer than TextMate’s so there’s an even richer set of plugins, bundles and syntax files available.

Another reason people give for enjoying Vim is that it runs everywhere. Yes, it’s nice to have my favorite editor available when I SSH into a machine, but it’s not a huge deal if I don’t. I almost always edit files on my own machine and then deploy them with an automated script of some kind, so editing on a server is very rare for me.

Core Differences

For me, Vim and TextMate are very different. TextMate relies mainly on keyboard shortcuts (involving Shift, Ctrl, Cmd and Alt) to do more than simply edit text. Vim uses a “modal editing” idea to accomplish the same task.

TextMate’s philosophy is much the same as any normal OS X application’s, and I said that acting like the OS reduces the barrier of entry for a text editor, so why do I prefer Vim’s philosophy?

First I should prefix my answer with this: I’m not averse to a learning curve for something as important to me as a text editor. Yes, it’s awesome if I don’t need to unnecessarily rewire my brain, but if it’s going to save me a lot of time then I’m willing to cut a bit of slack here.

My problem with TextMate’s philosophy can be summed up in one word: “shadowing”.

If I want to define a new command in a bundle (or if someone else has already done it), I’m never really sure if I’m “overwriting” some other command that my fingers know but my brain doesn’t really register.

By the time I try to use the default command it could be days or weeks after I defined the custom one, and I need to figure out what I did all over again. It’s quite frustrating.

Vim’s “insert” mode means that when I’m editing text I’m using the normal OS X text movement shortcuts that I know and love. When I want to do something special I enter normal mode and don’t have to worry about shadowed commands (Vim’s leader key also helps with this).

Vim’s “Feeling”

Vim’s normal mode has a unique “feeling” that I haven’t seen in any other text editor (although [E](http://www.e-texteditor.com/) may [change that](http://e-texteditor.com/blog/2010/beyond-vi)). I’ve heard it described in a number of ways, and I think that while no one analogy fully explains it, together they give a pretty good description.

A “Language” of Text Editing

One way to think about Vim’s normal-mode commands is like a language. You have “verbs” and “nouns”. For example: the “change” command (c) would be a verb and the “word” item (w) is a noun. You can combine them to form imperative sentences that talk about what you want to do with your text.

The wonderful part about this is that whenever you learn a new verb (like “delete” (d)) you can immediately apply it to all the nouns you know, and vice versa.

Vim also has “adjectives” like “inside” and “around” (i and a) that let you craft sentences like “change inside parenthesis” (ci( or cib). Once you learn one of these you can immediately apply it to all the verbs and nouns you already know.

The Physics of Text

The second way to describe the feeling of Vim is “physics”. It’s a much less concrete description but I think it’s still useful.

If I throw a bowling ball into the air it behaves much the same way as if I throw a turkey sandwich. They might have very different effects when they land on someone but the act of throwing them is pretty much the same.

Vim’s operators/actions/verbs act the same way. daw has a very different effect than da{ but they can both be understood with a basic underlying principle: da<something> will “delete around <something>”.

“Programming” Your Editing

The last way that I hear people talking about Vim’s editing is: “it feels like you’re ‘programming’ your text”.

This analogy is often made by programmers (obviously) and I think it works on more than one level.

First: Vim’s basic editing commands can be likened to function calls. daw can be thought of a running a function like delete(type='word', around=True). I don’t think this way (I prefer the language and physics analogies) but other people do.

Although I don’t think of the *basic* Vim commands as programming I do see some aspects of it in other areas. For example: say I define a leader command in my ~/.vimrc file:

nnoremap <leader>1 yypVr=

I think of this as defining <leader>1 to be a function that performs the following actions:

* Yank/copy the current line.
* Paste it below (and move down to the pasted version).
* Select the copied line.
* Replace every character with =.

Another more obvious “programming” aspect of Vim is creating macros. Macros are like tiny little functions you create to help you when you’re editing a file. You can define them, execute them, and even edit them by pasting, editing and yanking them back.

Getting Started

When I switched back to Vim I did a few things that helped make my transition stick.

First: I bought [Learning the vi and Vim Editors](http://amzn.com/059652983X) and read it cover to cover. It helps quite a bit to know the history behind Vim, and the book also teaches you the basics of Vim’s editing commands.

I also looked through [the Vim subreddit](http://reddit.com/r/vim/) to find some good blog posts about switching. A lot of people have written about it and it was very helpful to read about their experiences.

And finally, the most important part: I switched cold-turkey. Vim runs everywhere, so there’s really no excuse for not doing this. Every decent OS has a version of Vim that makes insert mode “just work” for their OS (like gvim or MacVim), so the barrier to entry these days is pretty low.

You can get Vim and start using it right away by staying in insert mode most of the time. Then you can start learning its real features at your own pace. Trying to use two different editors at once just slows down the learning process and makes you less productive in both.

Making Vim More Useful

One of the flaws of Vim is that it takes quite a bit of configuration to make it behave in a decent way. The defaults are backwards-compatible with vi (i.e. older than most college students) and not very helpful.

Here are a few of the things I do to make Vim a bit more palatable.

**A quick side note:** even if you don’t use Vim, you *need* to [remap your capslock key](http://c2.com/cgi/wiki?RemapCapsLock) to something useful. Just do it, you’ll thank me.

Important .vimrc Lines

I won’t go through every line in [my ~/.vimrc file](http://bitbucket.org/sjl/dotfiles/src/tip/vim/.vimrc), but here are some of the lines that I simply could not live without.

First, a few lines that you absolutely must have:

filetype off

call pathogen#runtime\_append\_all\_bundles()

filetype plugin indent on

set nocompatible

set modelines=0

The filetype and call lines are for loading Pathogen, which is described in the bundles section. See [Pathogen’s docs](http://www.vim.org/scripts/script.php?script_id=2332) to learn about why the first filetypeline is there.

set nocompatible gets rid of all the crap that Vim does to be vi compatible. It’s 2010 — we don’t need to be compatible with vi at the expense of functionality any more.

The modelines bit prevents some [security exploits](http://lists.alioth.debian.org/pipermail/pkg-vim-maintainers/2007-June/004020.html) having to do with modelines in files. I never use modelines so I don’t miss any functionality here.

Next I set my tab settings:

set tabstop=4

set shiftwidth=4

set softtabstop=4

set expandtab

I like all tabs to expand to four spaces. Check out [this Vimcast](http://vimcasts.org/episodes/tabs-and-spaces/) to learn more about each of these options.

Next are a few options that just make things better:

set encoding=utf-8

set scrolloff=3

set autoindent

set showmode

set showcmd

set hidden

set wildmenu

set wildmode=list:longest

set visualbell

set cursorline

set ttyfast

set ruler

set backspace=indent,eol,start

set laststatus=2

set relativenumber

set undofile

Each of these lines are basically to make Vim behave in a sane manner. The two “interesting” ones are the last two, and both deal with features that are[new in Vim 7.3](http://groups.google.com/group/vim_announce/browse_thread/thread/66c02efd1523554b).

relativenumber changes Vim’s line number column to display how far away each line is from the current one, instead of showing the absolute line number.

I almost never care what numeric line I’m on in a file (and if I do I can see it in the status line), so I don’t miss the normal line numbers. I *do* care how far away a particular line might be, because it tells me what number I need to use with motion commands like d<NUMBER>d.

undofile tells Vim to create <FILENAME>.un~ files whenever you edit a file. These files contain undo information so you can undo previous actions even after you close and reopen a file.

Next I change the <leader> key:

let mapleader = ","

For me , is easier to type than \. I use leader commands constantly so it’s worth changing.

The next thing I do is tame searching/moving:

nnoremap / /\v

vnoremap / /\v

set ignorecase

set smartcase

set gdefault

set incsearch

set showmatch

set hlsearch

nnoremap <leader><space> :noh<cr>

nnoremap <tab> %

vnoremap <tab> %

The first two lines fix Vim’s horribly broken default regex “handling” by automatically inserting a \v before any string you search for. This turns off Vim’s crazy default regex characters and makes searches use normal regexes. I already know Perl/Python compatible regex formatting, why would I want to learn another scheme?

ignorecase and smartcase together make Vim deal with case-sensitive search intelligently. If you search for an all-lowercase string your search will be case-insensitive, but if one or more characters is uppercase the search will be case-sensitive. Most of the time this does what you want.

gdefault applies substitutions globally on lines. For example, instead of:%s/foo/bar/g you just type :%s/foo/bar/. This is almost always what you want (when was the last time you wanted to only replace the first occurrence of a word on a line?) and if you need the previous behavior you just tack on the gagain.

incsearch, showmatch and hlsearch work together to highlight search results (as you type). It’s really quite handy, as long as you have the next line as well.

The <leader><space> mapping makes it easy to clear out a search by typing,<space>. This gets rid of the distracting highlighting once I’ve found what I’m looking for.

The last two lines make the tab key match bracket pairs. I use this to move around all the time and <tab> is a hell of a lot easier to type than %.

The next section makes Vim handle long lines correctly:

set wrap

set textwidth=79

set formatoptions=qrn1

set colorcolumn=85

These lines manage my line wrapping settings and also show a colored column at 85 characters (so I can see when I write a too-long line of code).

See [:help fo-table](http://vimdoc.sourceforge.net/htmldoc/change.html#fo-table) and the Vimcasts on [soft wrapping](http://vimcasts.org/episodes/soft-wrapping-text/) and [hard wrapping](http://vimcasts.org/episodes/hard-wrapping-text/) for more information.

Next comes something other TextMate refugees may like:

set list

set listchars=tab:▸\ ,eol:¬

This makes Vim show invisible characters with the same characters that TextMate uses. You might need to adjust your color scheme so they’re not too distracting. [This Vimcast](http://vimcasts.org/episodes/show-invisibles/) has more information.

New Vim users will want the following lines to teach them to do things right:

nnoremap <up> <nop>

nnoremap <down> <nop>

nnoremap <left> <nop>

nnoremap <right> <nop>

inoremap <up> <nop>

inoremap <down> <nop>

inoremap <left> <nop>

inoremap <right> <nop>

nnoremap j gj

nnoremap k gk

This will disable the arrow keys while you’re in normal mode to help you learn to use hjkl. Trust me, you want to learn to use hjkl. Playing a lot of[Nethack](http://www.nethack.org/) also helps.

It also disables the arrow keys in insert mode to force you to get back into normal mode the instant you’re done inserting text, which is the “right way” to do things.

It also makes j and k work the way you expect instead of working in some archaic “movement by file line instead of screen line” fashion.

Next, get rid of that stupid goddamned help key that you will invaribly hit*constantly* while aiming for escape:

inoremap <F1> <ESC>

nnoremap <F1> <ESC>

vnoremap <F1> <ESC>

I also like to make ; do the same thing as : — it’s one less key to hit every time I want to save a file:

nnoremap ; :

I don’t remap : back to ; because it seems to break a bunch of plugins.

Finally, I really like TextMate’s “save on losing focus” feature. I can’t remember a time when I *didn’t* want to save a file after tabbing away from my editor (especially with version control and Vim’s persistent undo):

au FocusLost \* :wa

Those are the most important bits of my ~/.vimrc file. Next I’ll talk about the wonderful namespace of customization that is Vim’s <leader> key.

Using the Leader

Vim dedicates an entire keyboard key for user-specific customizations. This is called the “leader” and by default it’s mapped to \. As I mentioned in the previous section I prefer to use , instead.

Each person will find little things they type or execute often and want to create shortcuts for those things. The leader is a kind of “namespace” to keep those customizations separate and prevent them from shadowing default commands.

Here are a few of the things I use leader commands for. You’ll certainly have different ideas than I do, but this might give you an idea of what you can do.

I use ,W to mean “strip all trailing whitespace in the current file” so I can clean things up quickly:

nnoremap <leader>W :%s/\s\+$//<cr>:let @/=''<CR>

I use Ack a lot (described below), so I mapped a leader key for it:

nnoremap <leader>a :Ack

I work with HTML often, so I have ,ft mapped to a “fold tag” function:

nnoremap <leader>ft Vatzf

I also work with Nick Sergeant and he likes his CSS properties sorted, so here’s a ,S mapping that sorts them for me:

nnoremap <leader>S ?{<CR>jV/^\s\*\}?$<CR>k:sort<CR>:noh<CR>

This next mapping imitates TextMates Ctrl+Q function to re-hardwrap paragraphs of text:

nnoremap <leader>q gqip

I have a ,v mapping to reselect the text that was just pasted so I can perform commands (like indentation) on it:

nnoremap <leader>v V`]

This last mapping lets me quickly open up my ~/.vimrc file in a vertically split window so I can add new things to it on the fly.

nnoremap <leader>ev <C-w><C-v><C-l>:e $MYVIMRC<cr>

Quicker Escaping

One thing you’ll find yourself constantly doing in Vim is moving from insert mode to normal mode. The default way to do this is by hitting Escape, but that key is out of the way and hard to hit.

Another way is to use Ctrl+C or Ctrl+[, but I don’t like using a chord for something I press that often.

I personally use jj to exit back to normal mode. The only time I’ve ever actually tried to hit two j’s in a row is just now while writing this entry, so it doesn’t conflict with my normal typing at all:

inoremap jj <ESC>

Working With Split Windows

Being able to split my editor window and see more than one file at once is one of the main reasons I switched to Vim. I know I’m not alone here after reading various posts around the internet asking for this feature in TextMate.

Vim’s default commands for interacting with splits, however, are kind of clunky. I’ve added a few mappings to my ~/.vimrc that make things work a bit smoother.

This first mapping makes ,w open a new vertical split and switch over to it. Because really, how often do you split your window and not want to do something in the new split?

nnoremap <leader>w <C-w>v<C-w>l

You might notice that this mapping is for vertical splits. I almost *never* use horizontal splits. All of my screens are widescreen, so I can fit several files onscreen if they’re split vertically. Horizontal splits don’t let me see enough of the file. If I *really* want a horizontal split I can use <C-w>s to get one.

This next set of mappings maps <C-[h/j/k/l]> to the commands needed to move around your splits. If you remap your capslock key to Ctrl it makes for very easy navigation.

nnoremap <C-h> <C-w>h

nnoremap <C-j> <C-w>j

nnoremap <C-k> <C-w>k

nnoremap <C-l> <C-w>l

Aesthetics

I know some people say: “looks don’t matter, the only thing that’s important is functionality”. That’s completely wrong.

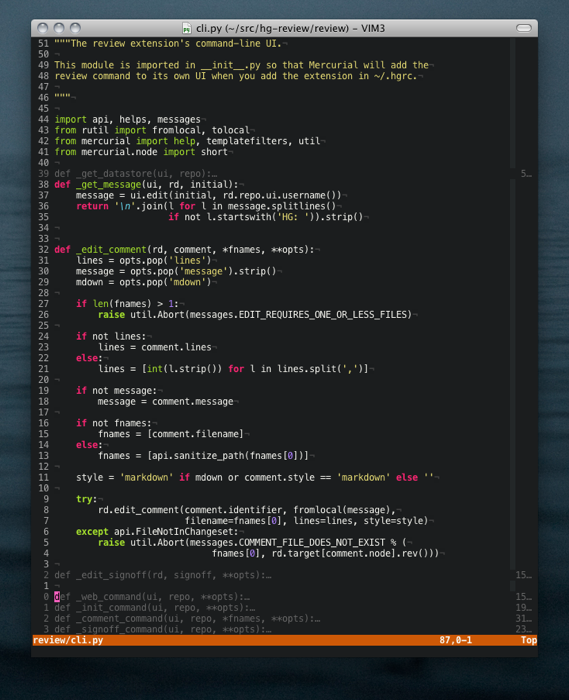
If I’m looking at something for multiple hours each day it had damn well better look pretty. There are three main steps to making Vim look pretty (for me, on OS X — your mileage may vary).

First: get [MacVim](http://code.google.com/p/macvim/). It makes Vim look (and behave) like a native Mac application. It helps it avoid standing out like a sore thumb among your other applications. Linux and Windows users will probably want gvim.

Next: pick a decent font. I prefer 12 point [Menlo](http://arstechnica.com/apple/news/2009/06/font-changes-coming-to-mac-os-x-snow-leopard.ars). You might have a different preference, but at least try out a few and find one that has a nice balance between line height (so you can get a good amount of lines on the screen) and readability (slashed zeros, please).

Finally: find a good color scheme. I’ve been using [a slightly modified version](http://bitbucket.org/sjl/dotfiles/src/tip/vim/colors/molokai.vim)of [Molokai](http://www.vim.org/scripts/script.php?script_id=2340) (a port of the TextMate [Monokai](http://www.monokai.nl/blog/2006/07/15/textmate-color-theme/) theme) since I came back to Vim, but recently I’ve also started flirting with [Mustang](http://hcalves.deviantart.com/art/Mustang-Vim-Colorscheme-98974484) and [Clouds Midnight](http://forr.st/~yZn). Spend an hour and find something that looks good to you.

Here’s what my current setup looks like:



Bundles I Use

Vim has been around for a *long* time, and many people have written extensions for it. Here are a few of the bundles I couldn’t live without in my day-to-day editing.

Pathogen

First of all: [Tim Pope](http://github.com/tpope) has created the wonderful [Pathogen](http://github.com/tpope/vim-pathogen) plugin that makes managing *other* Vim plugins painless. Instead of scattering their files throughout your ~/.vim/ folder you can keep their files inside a single folder in ~/.vim/bundles/.

Install it and keep your sanity.

PeepOpen

One of TextMate’s “killer features” is its Cmd+T key. It lets you open files quickly by typing fragments of their names.

There are a number of Vim plugins out there that try to emulate it. My favorite is [PeepOpen](http://peepcode.com/products/peepopen). Yes, it’s OS X only and costs money, but it’s worth it. It looks good and “just works”.

PeepOpen has some nifty little features, like showing Git metadata in the file list. I’ve [offered a patch](http://github.com/topfunky/PeepOpen-Issues/issues#issue/91) to add the same functionality for Mercurial but haven’t heard back from the developers.

If you don’t use OS X (or want a free alternative to PeepOpen) I hear the[Command-T](https://wincent.com/products/command-t) plugin is quite nice.

NERDTree

One plugin you’ll hear about in almost all of these “switching to Vim” blog posts is [NERDTree](http://github.com/scrooloose/nerdtree). It’s a little plugin for browsing files in your project, and it works great. There’s no better “TextMate file-drawer” plugin around.

NERDCommenter

Want to be able to comment and uncomment code with a few keypresses? You need [NERDCommenter](http://github.com/scrooloose/nerdcommenter). It’s surprising that Vim doesn’t have decent commenting functionality built in, but NERDCommenter fixes that.

I really only use this plugin for one single function: “toggle comment” with<leader>c<space>. That function alone is worth installing it.

Ack

If you’re a programmer and you don’t know about [Ack](http://betterthangrep.com/), you need to start using it now. It’s far, far better than grep.

The [Ack plugin](http://github.com/mileszs/ack.vim) for Vim integrates Ack with Vim’s quickfix window so you can easily search and jump to results.

As I mentioned in the section about leader mappings I’ve got ,a mapped to bring up Ack all ready to search.

Snipmate

Another feature of TextMate that was *amazing* when I first saw it was snippets. [SnipMate](http://github.com/msanders/snipmate.vim) is a Vim plugin that emulates TextMate snippets.

It also stores them in easily-version-controlled plain text files, which is a bonus over TextMate’s snippets.

Sparkup

[Sparkup](http://github.com/rstacruz/sparkup) is pretty much a great port of [Zen Coding](http://code.google.com/p/zen-coding/) for Vim. If you write HTML at all this plugin will save you a ton of typing.

In a nutshell it lets you type something like:

div.content>h1.post-title+p{Sample Content}

Press Cmd+E and it will expand to this:

<div class=*"content"*>

<h1 class=*"post-title"*></h1>

<p>Sample Content</p>

</div>

It’s far less typing and it adds up over time.

Yankring

Vim’s copying and pasting functionality (which uses registers) is very, very powerful, but it’s not exactly “user friendly”. The [YankRing](http://www.vim.org/scripts/script.php?script_id=1234) plugin adds a lot*more* power, but also adds a few features that make copying and pasting much more pleasant.

For example, after you paste some text you can replace that paste with the previous item you copied with Ctrl-P. You can cycle back further by just hitting Ctrl-P over and over.

YankRing also shares your yanked text between Vim windows, which makes things “just work” when you want to paste text from one window into another.

You can also show a list of all your previously yanked text with :YRShow. Mapping that command to a key is quite helpful:

nnoremap <silent> <F3> :YRShow<cr>

inoremap <silent> <F3> <ESC>:YRShow<cr>

YankRing offers a ton of other cool functionality but I haven’t had the time or motivation to really dig in and find out how to use it.

Surround (and Repeat)

Another of Time Pope’s plugins that I use all the time is the [Surround](http://github.com/tpope/vim-surround)plugin. It adds another layer to Vim’s physics: “surrounding items”.

For example, you can “change surrounding single quotes to double quotes” by using cs'". Or if you’re writing something in Markdown and want to italicize a word you can use ysiW\*.

All the nouns and verbs you already know can be used, which fits with Vim’s philosophy and makes it extremely powerful.

The [Repeat](http://github.com/tpope/vim-repeat) plugin is necessary to make these surround actions repeatable with ..

Slime

Emacs users often tout SLIME as one of Emacs’ “killer features”. When writing LISP code it’s unchallengably awesome.

Vim doesn’t have anything like SLIME. It’s not integrated with any language the way Emacs is coupled to LISP. You can, however, achieve *some* of the power of SLIME in Vim.

[Jonathan Palardy](http://technotales.wordpress.com/) has written a little Vim plugin called [Slime.vim](http://technotales.wordpress.com/2007/10/03/like-slime-for-vim/) that lets Vim easily communicate with a screen session, probably one running a REPL for some language like LISP, Python or Ruby.

The general idea is that you’ll fire up LISP in a screen session, then go over to Vim and use Ctrl+C Ctrl+C to put chunks of code in your Vim window into the screen session.

It works and is quite handy for when you want to experiment with the contents of a file. Add a mapping for shoving over the entire file and you’re all set to bash out and test some code quickly.

I personally use [a slightly modified version](http://bitbucket.org/sjl/dotfiles/src/tip/vim/plugin/slime.vim) that removes a bit of the flexibility I don’t need to make it quicker to use. Maybe some day I’ll expand on it and push it up to BitBucket and GitHub in a repo of its own.

Scratch

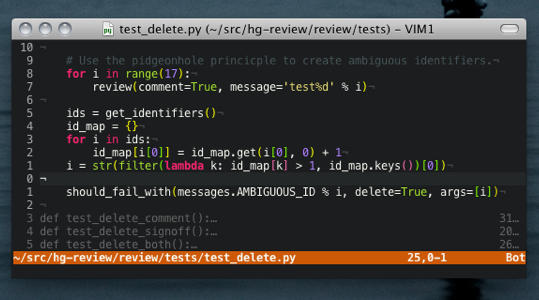
The [Scratch](http://www.vim.org/scripts/script.php?script_id=664) plugin adds a function to quickly open a “scratch” buffer that will never be saved.

I have it mapped to <leader><tab> and use it with slime to quickly stick a bit of code that I don’t want in my actual file into the REPL.

Rainbow Parentheses

[Rainbow Parentheses](http://www.vim.org/scripts/script.php?script_id=1561) is a Vim plugin that colorizes parentheses, square brackets, curly brackets and angle brackets according to their nesting.

It’s easiest to describe what this looks like with a screenshot:



I use [a slightly modified version](http://bitbucket.org/sjl/dotfiles/src/tip/vim/bundle/rainbow/). I have it mapped to <leader>R and off by default. When I’m dealing with a particularly hairy piece of code that has lots of nesting I simply hit <leader>R and get some color that helps me keep my place.

Things I Want

Although Vim is definitely my new favorite editor, this blog post wouldn’t be complete without a few complaints.

If anyone fixes/implements any of these I’ll gladly buy them several beers.

HTML Indentation that Doesn’t Suck

I write a lot of HTML. One thing that annoys the hell out of me is Vim’s “smart” indenting of HTML. I’ve tried every combination of cindent,smartindent and autoindent and I can’t seem to get Vim to behave sanely.

The problem is that sometimes when I press return after a tag that I’ve already created Vim will unindent the new line *and the previous line*. This is excruciatingly annoying.

All I want is the following:

* When I press return, create a new line at the same indentation level as the current one. Don’t try to be clever and adjust the indent of new line in any fashion.
* If I press tab at the beginning of a line, indent the *current line* by one tabstop.
* If I press backspace at the beginning of a line, delete one tabstop on that line only.

Please, someone make this happen and restore my sanity.

Python Support for Slime

Earlier I mentioned that I use the slime plugin. One language it doesn’t work very well with is Python, because it preserves whitespace when it moves the code over the REPL.

This is a problem with Python because if you try to move over an indented block of code you’ll get “unexpected indent” problems.

I’d love for someone to tweak slime so it intelligently unindents Python code. I have some ideas on how this could work but unfortunately I don’t have the time to implement them.

Gundo

A little-known fact about Vim is that it doesn’t keep a list of your undo history, it keeps a *tree*.

If I make 5 changes, undo 2 and then make 2 more Vim keeps track of *all* of them. You can use :undolist to see the list of leaves in this “undo tree”.

What I want is for someone to make an awesome “graphical undo” plugin that shows a graph of the undo tree, much like Mercurial’s graphlog extension shows a graph of the changesets in a repository.

This graph would actually be simpler than Mercurial’s graphlog because there are no merges to deal with.

For bonus points let me browse the tree with j and k, press p to preview a diff of what would happen if I went back to that version, and press <CR> to actually go back.

**UPDATE:** I got tired of waiting, so I [wrote it myself](http://bitbucket.org/sjl/gundo.vim/).

A Mercurial Version of Fugitive

The final thing I’d like is for someone to make a Mercurial plugin for Vim that is as awesome as Tim Pope’s [Fugitive](http://github.com/tpope/vim-fugitive).

There’s some Mercurial support for Vim out there, but for the most part it’s tied up in plugins that support multiple version control systems.

Any plugin that tries to support every system will fail at supporting one single system perfectly, so I’d love for someone to come along and make an*awesome* Vim plugin that lets you use Mercurial to its fullest while inside Vim.

I simply don’t know vimscript well enough to do this on my own, but if someone else is interested I can certainly help out on the Mercurial and user interface aspects.

Overall Thoughts

Transitioning back to Vim after using TextMate for a few years wasn’t trivial, but I feel that it was worth the effort. I have features like split windows now and there are plugins that can provide anything I miss from TextMate.

If TextMate 2 ever gets released it would have to do some pretty amazing things for me to want to switch back.

Overall I’m very happy with Vim. There are a lot of things that could be improved, especially with the default settings (who cares about vi compatibility in 2010?), but a text editor is worth a learning curve.

In my eyes Vim has two huge advantages over almost any other editor out there. The first is it’s *huge* ecosystem of plugins and syntax definitions, which is only rivaled by Emacs (and *possibly* TextMate, though I doubt it). The second is it’s “modal editing” philosophy, which is extremely powerful and hasn’t been adopted by any other mainstream editor (except for *very* recent versions of E).

If you’ve got questions or comments you should [find me on Twitter](http://twitter.com/stevelosh/) and let me know.

[*Bitbucket*](https://bitbucket.org/sjl/)*・ [GitHub](https://github.com/sjl/) ・*[*Twitter*](https://twitter.com/stevelosh/)*・ [Instagram](https://instagram.com/thirtytwobirds/)*