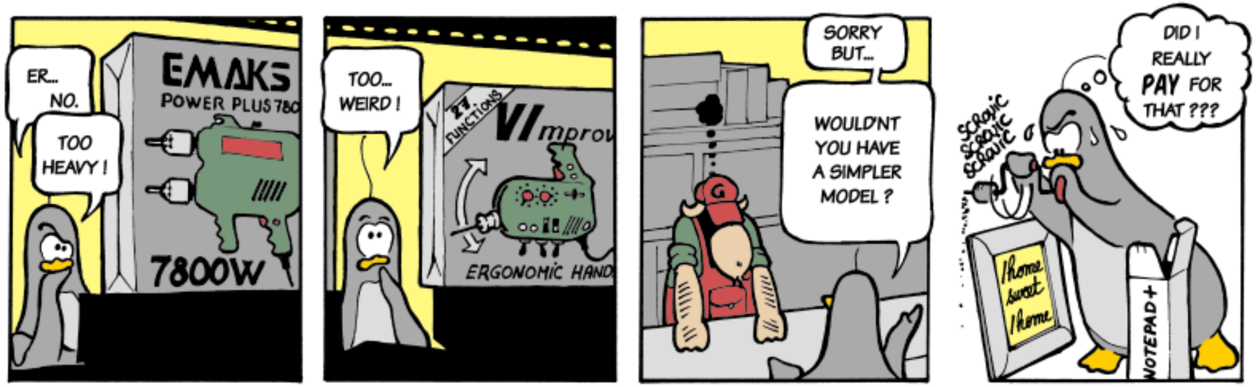
Easy Introduction to Vim with Riku

with some examples and demonstrations



Background

I’ve spent about a week (if not a couple of weeks) trying to find the right text editor (and how to configure it) for me.

This process was time consuming, confusing, complicated, and difficult.

I’ve decided to bring this up as a topic for SYSKA so that no one else will have to repeat what I did

Why use text editors?

So you can code quickly and efficiently **WITHOUT** using the mouse

- The time it takes for you to take your hands off the keyboard to move the mouse is wasteful
    - Probably the reason why we have shortcuts in the first place (ctrl-c, ctrl-v, etc.)

What are my options?

There are many text editors:

- gedit
- notepad/notepad++
- emacs
- visual studio
- vi/vim
- etc., etc., etc.

So which text editor is the best?

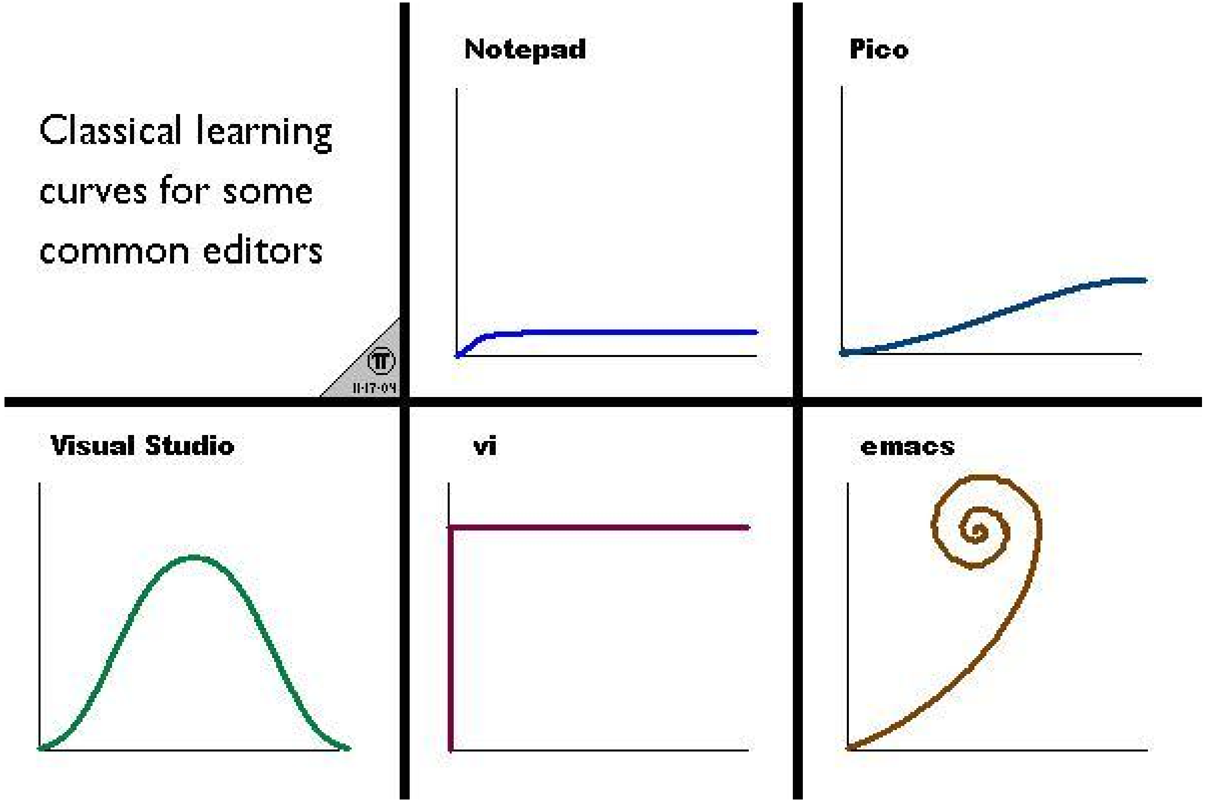
Emacs and vim seems to be the two most recommended text editors for programmers (i. e. us).

Both emacs and vim are:

- customizable (e.g. plugins)
- efficient (once you know how to use it)

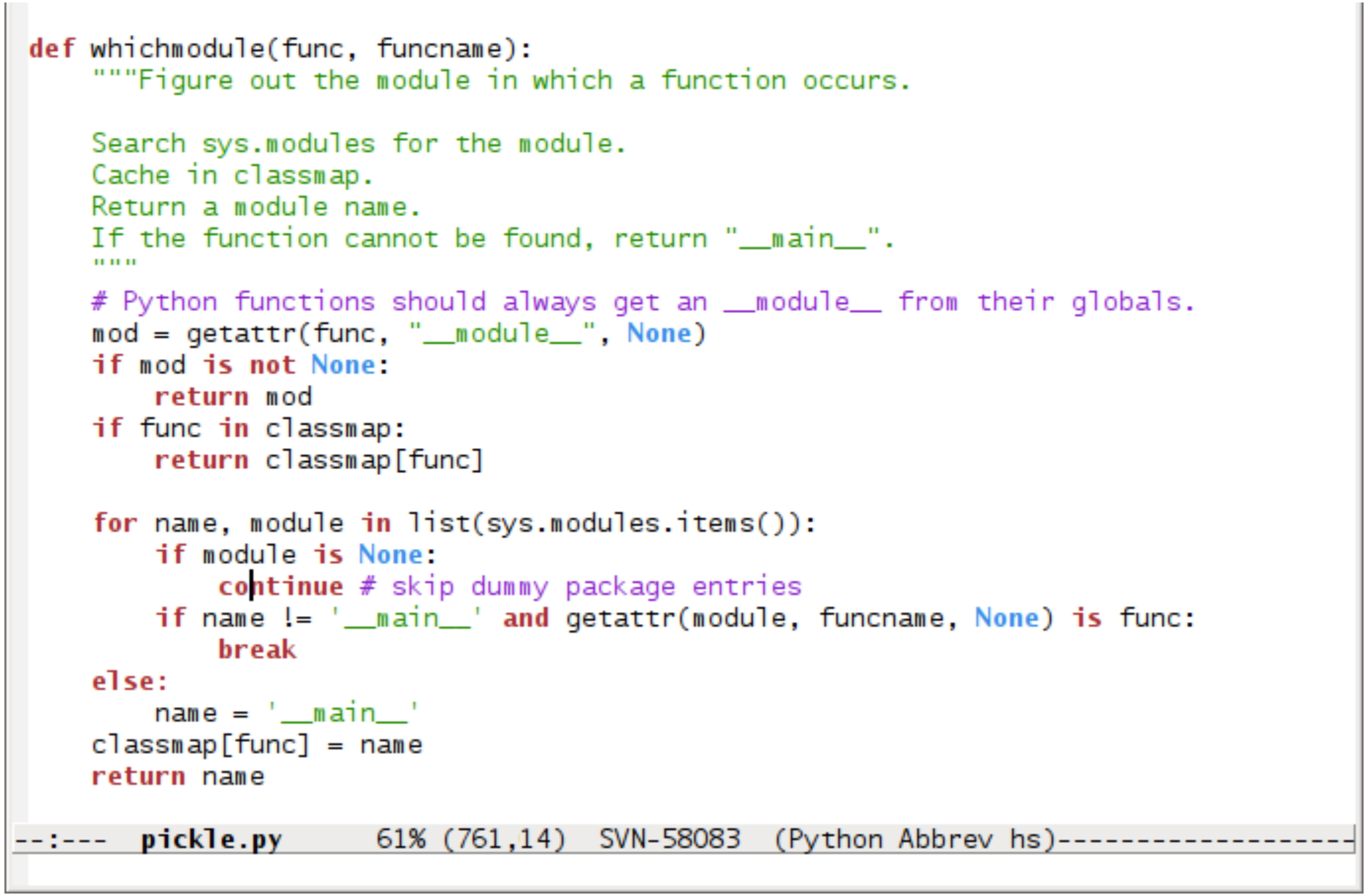
The problem with most people (including me) is that they don't know how to use it properly

Classical learning curves for some common editors



Emacs

- Very easy to use - it's just like a notepad
- Syntax highlighting
- Auto indentation
- Many shortcuts for efficiency
- Looks pretty
- No ‘modes’ like in vim



```
def whichmodule(func, funcname):
    """Figure out the module in which a function occurs.

    Search sys.modules for the module.
    Cache in classmap.
    Return a module name.
    If the function cannot be found, return "__main__".
    """
    # Python functions should always get an __module__ from their globals.
    mod = getattr(func, "__module__", None)
    if mod is not None:
        return mod
    if func in classmap:
        return classmap[func]

    for name, module in list(sys.modules.items()):
        if module is None:
            continue # skip dummy package entries
        if name != '__main__' and getattr(module, funcname, None) is func:
            break
    else:
        name = '__main__'
    classmap[func] = name
    return name

--:---  pickle.py      61% (761,14)   SVN-58083  (Python Abbrev hs)------------------
```

Why I switched from emacs to vim

Emacs was very easy to use, but there were

TOO MANY SHORTCUTS!!

Actually, having too many shortcuts wasn’t really much of a problem, but the combinations for the shortcuts were

- too many different (and quite hard to remember) combinations of letter keys with or without ctrl, alt, and/or shift

GNU Emacs Reference Card

(for version 22)

Starting Emacs

To enter GNU Emacs 22, just type its name: `emacs`

Leaving Emacs

suspend Emacs (or iconify it under X)	`C-z`
exit Emacs permanently	`C-x C-c`

Files

read a file into Emacs	`C-x C-f`
save a file back to disk	`C-x C-s`
save **all** files	`C-x s`
insert contents of another file into this buffer	`C-x i`
replace this file with the file you really want	`C-x C-v`
write buffer to a specified file	`C-x C-w`
toggle read-only status of buffer	`C-x C-q`

Getting Help

The help system is simple. Type `C-h` (or `F1`) and follow the directions. If you are a first-time user, type `C-h t` for a **tutorial**.

remove help window	`C-x 1`
scroll help window	`C-M-v`
apropos: show commands matching a string	`C-h a`
describe the function a key runs	`C-h k`
describe a function	`C-h f`
get mode-specific information	`C-h m`

Error Recovery

abort partially typed or executing command	`C-g`
recover files lost by a system crash	`M-x recover-session`
undo an unwanted change	`C-x u`, `C-_` or `C-/`
restore a buffer to its original contents	`M-x revert-buffer`
redraw garbaged screen	`C-l`

Incremental Search

search forward	`C-s`
search backward	`C-r`
regular expression search	`C-M-s`
reverse regular expression search	`C-M-r`
select previous search string	`M-p`
select next later search string	`M-n`
exit incremental search	`RET`
undo effect of last character	`DEL`
abort current search	`C-g`

Use `C-s` or `C-r` again to repeat the search in either direction. If Emacs is still searching, `C-g` cancels only the part not done.

Motion

entity to move over	**backward**	**forward**
character	`C-b`	`C-f`
word	`M-b`	`M-f`
line	`C-p`	`C-n`
go to line beginning (or end)	`C-a`	`C-e`
sentence	`M-a`	`M-e`
paragraph	`M-{`	`M-}`
page	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
function	`C-M-a`	`C-M-e`
go to buffer beginning (or end)	`M-<`	`M->`

scroll to next screen	`C-v`
scroll to previous screen	`M-v`
scroll left	`C-x <`
scroll right	`C-x >`
scroll current line to center of screen	`C-u C-l`

Killing and Deleting

entity to kill	**backward**	**forward**
character (delete, not kill)	`DEL`	`C-d`
word	`M-DEL`	`M-d`
line (to end of)	`M-0 C-k`	`C-k`
sentence	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill **region**	`C-w`
copy region to kill ring	`M-w`
kill through next occurrence of *char*	`M-z` *char*
yank back last thing killed	`C-y`
replace last yank with previous kill	`M-y`

Marking

set mark here	`C-@` or `C-SPC`
exchange point and mark	`C-x C-x`
set mark *arg* **words** away	`M-@`
mark **paragraph**	`M-h`
mark **page**	`C-x C-p`
mark **sexp**	`C-M-@`
mark **function**	`C-M-h`
mark entire **buffer**	`C-x h`

Query Replace

interactively replace a text string	`M-%`
using regular expressions	`M-x query-replace-regexp`

Valid responses in query-replace mode are

replace this one, go on to next	`SPC`
replace this one, don't move	`,`
skip to next without replacing	`DEL`
replace all remaining matches	`!`
back up to the previous match	`^`
exit query-replace	`RET`
enter recursive edit (`C-M-c` to exit)	`C-r`

Multiple Windows

When two commands are shown, the second is a similar command for a frame instead of a window.

delete all other windows	`C-x 1`	`C-x 5 1`
split window, above and below	`C-x 2`	`C-x 5 2`
delete this window	`C-x 0`	`C-x 5 0`
split window, side by side	`C-x 3`	
scroll other window	`C-M-v`	
switch cursor to another window	`C-x o`	`C-x 5 o`
select buffer in other window	`C-x 4 b`	`C-x 5 b`
display buffer in other window	`C-x 4 C-o`	`C-x 5 C-o`
find file in other window	`C-x 4 f`	`C-x 5 f`
find file read-only in other window	`C-x 4 r`	`C-x 5 r`
run Dired in other window	`C-x 4 d`	`C-x 5 d`
find tag in other window	`C-x 4 .`	`C-x 5 .`
grow window taller	`C-x ^`	
shrink window narrower	`C-x {`	
grow window wider	`C-x }`	

Formatting

indent current **line** (mode-dependent)	`TAB`
indent **region** (mode-dependent)	`C-M-\`
indent **sexp** (mode-dependent)	`C-M-q`
indent region rigidly *arg* columns	`C-x TAB`
insert newline after point	`C-o`
move rest of line vertically down	`C-M-o`
delete blank lines around point	`C-x C-o`
join line with previous (with arg, next)	`M-^`
delete all white space around point	`M-\`
put exactly one space at point	`M-SPC`
fill paragraph	`M-q`
set fill column	`C-x f`
set prefix each line starts with	`C-x .`
set face	`M-o`

Case Change

uppercase word	`M-u`
lowercase word	`M-l`
capitalize word	`M-c`
uppercase region	`C-x C-u`
lowercase region	`C-x C-l`

The Minibuffer

The following keys are defined in the minibuffer.

complete as much as possible	`TAB`
complete up to one word	`SPC`
complete and execute	`RET`
show possible completions	`?`
fetch previous minibuffer input	`M-p`
fetch later minibuffer input or default	`M-n`
regexp search backward through history	`M-r`
regexp search forward through history	`M-s`
abort command	`C-g`

Type `C-x ESC ESC` to edit and repeat the last command that used the minibuffer. Type `F10` to activate the menu bar using the minibuffer.

Vi Improved (vim)

- VERY steep learning curve
- **‘Modal User Interface’**
- Very effective at keeping the user away from the mouse
- Once you master it, you’ll be able to **“edit text at the speed of thought”**

Before we dive into vim

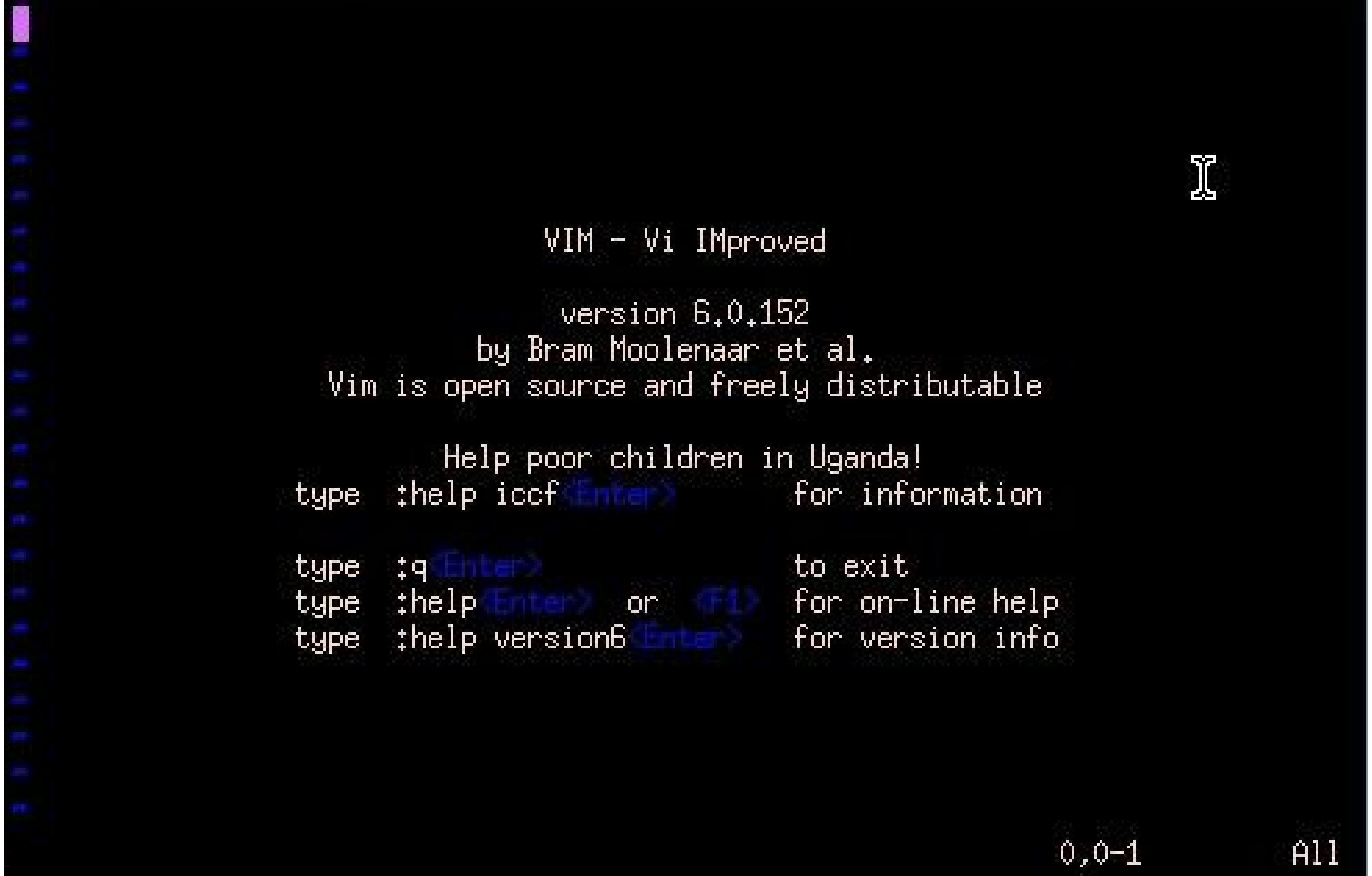
Vim has 3 main modes:

1. **Normal** - editing, cutting/pasting, moving around, etc.
2. **Insert** - for typing text
3. **Visual** - select text to edit/copy/cut/paste

Every key on the keyboard acts differently in different modes (but some keys overlap)

This is all you need to know for now

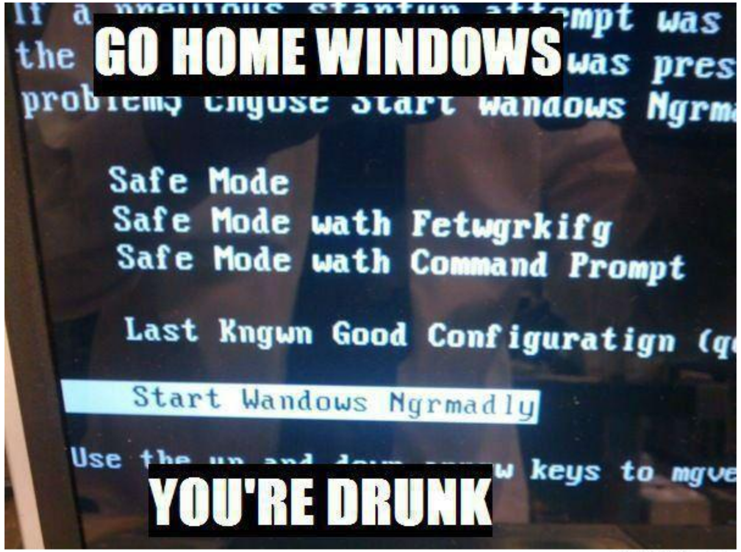
Are you ready?



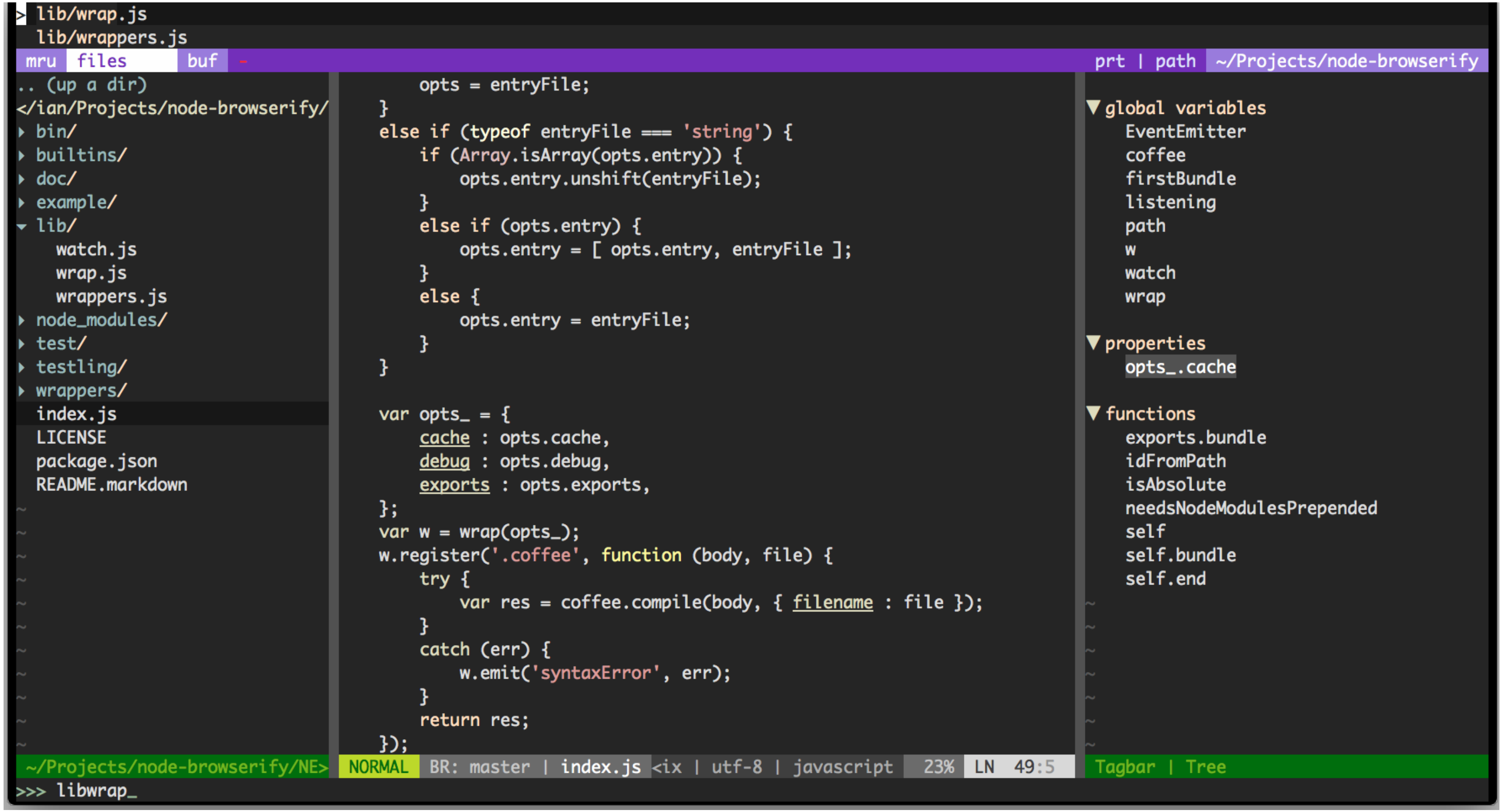
```
                VIM - Vi IMproved

                version 6.0.152
            by Bram Moolenaar et al.
    Vim is open source and freely distributable

           Help poor children in Uganda!
  type  :help iccf<Enter>       for information

  type  :q<Enter>               to exit
  type  :help<Enter>  or  <F1>  for on-line help
  type  :help version6<Enter>   for version info

                                          0,0-1         All
```

First Impression - Not Good

WTF, this thing is impossible to use, it looks crap, everything is dark, shit colour schemes and themes, no tab complete, hard to understand, why did I even open up vim, I want to go back to emacs and actually do some work.


GO HOME WINDOWS
Safe Mode
Safe Mode wath Fetwgrkifg
Safe Mode wath Command Prompt
Last Kngwn Good Configuratign
Start Wandows Ngrmadly
YOU'RE DRUNK
WeKnowMemes

But then I found this...



```
> lib/wrap.js
  lib/wrappers.js
 mru  files  buf  -                                              prt | path  ~/Projects/node-browserify
```

```
.. (up a dir)
</ian/Projects/node-browserify/
▸ bin/
▸ builtins/
▸ doc/
▸ example/
▾ lib/
    watch.js
    wrap.js
    wrappers.js
▸ node_modules/
▸ test/
▸ testling/
▸ wrappers/
  index.js
  LICENSE
  package.json
  README.markdown
~
~
~
~
~
~
~
~
~
~
~
```

```
        opts = entryFile;
    }
    else if (typeof entryFile === 'string') {
        if (Array.isArray(opts.entry)) {
            opts.entry.unshift(entryFile);
        }
        else if (opts.entry) {
            opts.entry = [ opts.entry, entryFile ];
        }
        else {
            opts.entry = entryFile;
        }
    }

    var opts_ = {
        cache : opts.cache,
        debug : opts.debug,
        exports : opts.exports,
    };
    var w = wrap(opts_);
    w.register('.coffee', function (body, file) {
        try {
            var res = coffee.compile(body, { filename : file });
        }
        catch (err) {
            w.emit('syntaxError', err);
        }
        return res;
    });
```

```
▼ global variables
    EventEmitter
    coffee
    firstBundle
    listening
    path
    w
    watch
    wrap

▼ properties
    opts_.cache

▼ functions
    exports.bundle
    idFromPath
    isAbsolute
    needsNodeModulesPrepended
    self
    self.bundle
    self.end
~
~
~
~
~
~
~
```

```
~/Projects/node-browserify/NE>  NORMAL  BR: master | index.js <ix | utf-8 | javascript   23%  LN  49:5   Tagbar | Tree
>>> libwrap_
```

Wait… But… What… HOW???

When vim starts up, it reads in an ‘invisible’ file called .
vimrc

- .vimrc is where you put all your custom configuration details

If vim can’t do it by default, there are many plugins for vim which you can install and improve vim’s functionality

Okay then, let’s install some plugins and configure the .
vimrc

Too many plugins!

Yep, there are way too many plugins

Also, some plugins are quite hard and complicated to install and configure

- installing Vundle or Pathogen
- creating .vim file, then putting everything in different directories

.vimrc is quite hard to configure if you don’t know what you’re doing

IT’S JUST TOO MUCH WORK

But then I found this...

spf13-vim: ‘The Ultimate Vim Distribution’

A ‘package’ of .vimrc with all of the highly recommended and useful plugins to make your vim experience better

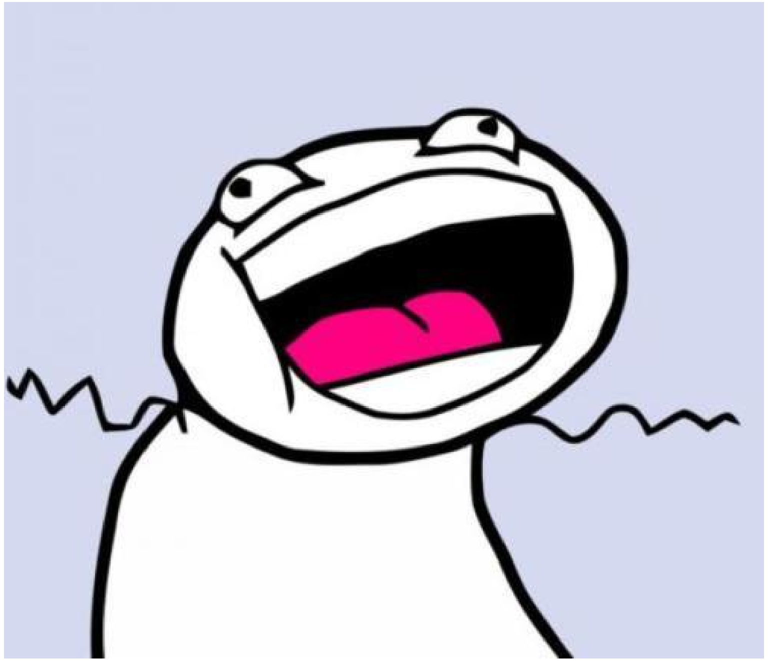
Download it from: https://github.com/spf13/spf13-vim

Run the shell script and you’re ready to go!!!

```
# spf13-vim : Steve Francia's Vim Distribution

spf13-vim is a distribution of vim plugins and resources for Vim, Gvim #

It is a good starting point for anyone intending to use VIM for develop#

The distribution is completely customisable using a `~/.vimrc.local` an#

Unlike traditional VIM plugin structure, which similar to UNIX throws a#

Great care has been taken to ensure that each plugin plays nicely with #

Lastly (and perhaps, most importantly) It is completely cross platform.#

 N  BR: 3.0 | README.markdown  unix | utf-8 | markdown   2%  LN  10
"README.markdown" 419L, 15789C written
```

What plugins does it have?

- NERDTree
- NERDCommenter
- Syntastic
- Tabularize
- neocomplcache
- Surround
- Airline
- ctrl-p
- easymotion
- etc., etc., etc…



And a huge variety of colour schemes to choose from!!

DEMONSTRATION

Extra comments

What if I don’t like spf13-vim’s configuration?

- there is a .vimrc.local file where you can add your configuration (this should override the spf13-vim’s .vimrc without ruining the plugins)

What if I don’t like some of the spf13-vim plugins and/or add my own?

- there’s a detailed description on the github website, but basically, you put all your personal plugins into .vimrc.bundles.local directory

Unfortunately, I don’t think there is an R syntax thing (well, none that’s easy to install)

How to vim

I’ve given you a great tool to make your vim experience enjoyable, but a tool that you can’t use properly is useless…

Recommended to do BEFORE you even think about typing ‘vim’ into the terminal

- Interactive vim tutorial http://www.openvim.com/
  - recommended for absolute beginners to get a feel for what vim is like
- Another interactive vim tutorial for linux/unix users - type in ‘vimtutor’ in terminal
  - pretty good - you’ll actually learn the basics of vim with this

After doing the above 2 points (takes less than an hour), you may now think about typing ‘vim’ into the terminal.

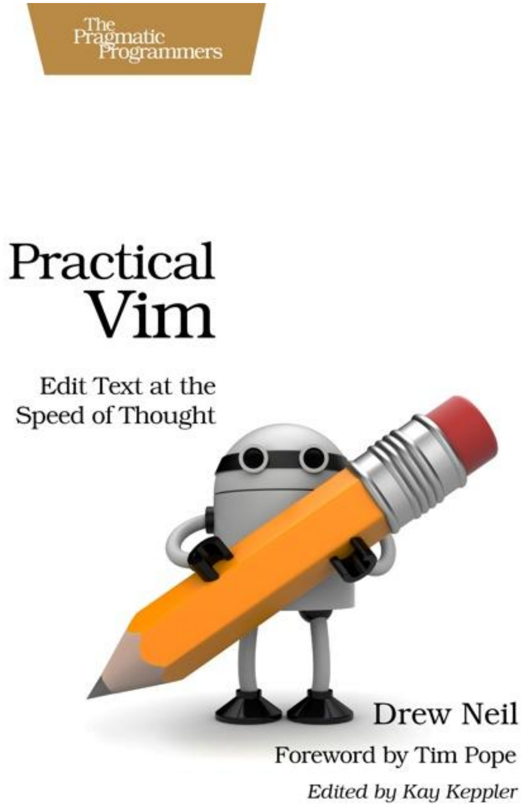
Highly Recommended

"Practical Vim - edit text at the speed of thought" written by Drew Neil

Rated 5 stars by over 90% of reviewers (i.e people who give a shit about vim) on Amazon

Recommended for everyone - from beginners to expert

Full of helpful tips that most vim users don't know, but should know



Hopefully, you’ll all end up like this