Beginner Vimtorial

Welcome!

Please login to your Ducss or Netsoc account with putty or open your OS's terminal.

If you don't know how to do this let us know!

Format of this tutorial

1) Vim config
2) Introduction to Vim
3) Crash course of vim commands
4) Tips & Tricks
5) Config explained

Run the following in a terminal

- cd
- mv .vimrc .vimrc.bkp
- mv .vim .vim.bkp
- git clone https://github.com/mikesligo/.vim.git
- ln -s .vim/.vimrc
- cd .vim
- ./getPlugins.sh
- ./updatePlugins.sh (sometimes, though not always)

Introduction to Vim

What is Vim?

- Vim is a Text Editor (created 1991)
- It is based off the editor 'Vi' (created 1976) and is short for 'Vi improved'
- It is typically a command line text editor, using the keyboard to navigate and edit text
- It or Vi is included in almost every Linux/Unix machine running
- Widely considered to be the best text editor for programmers

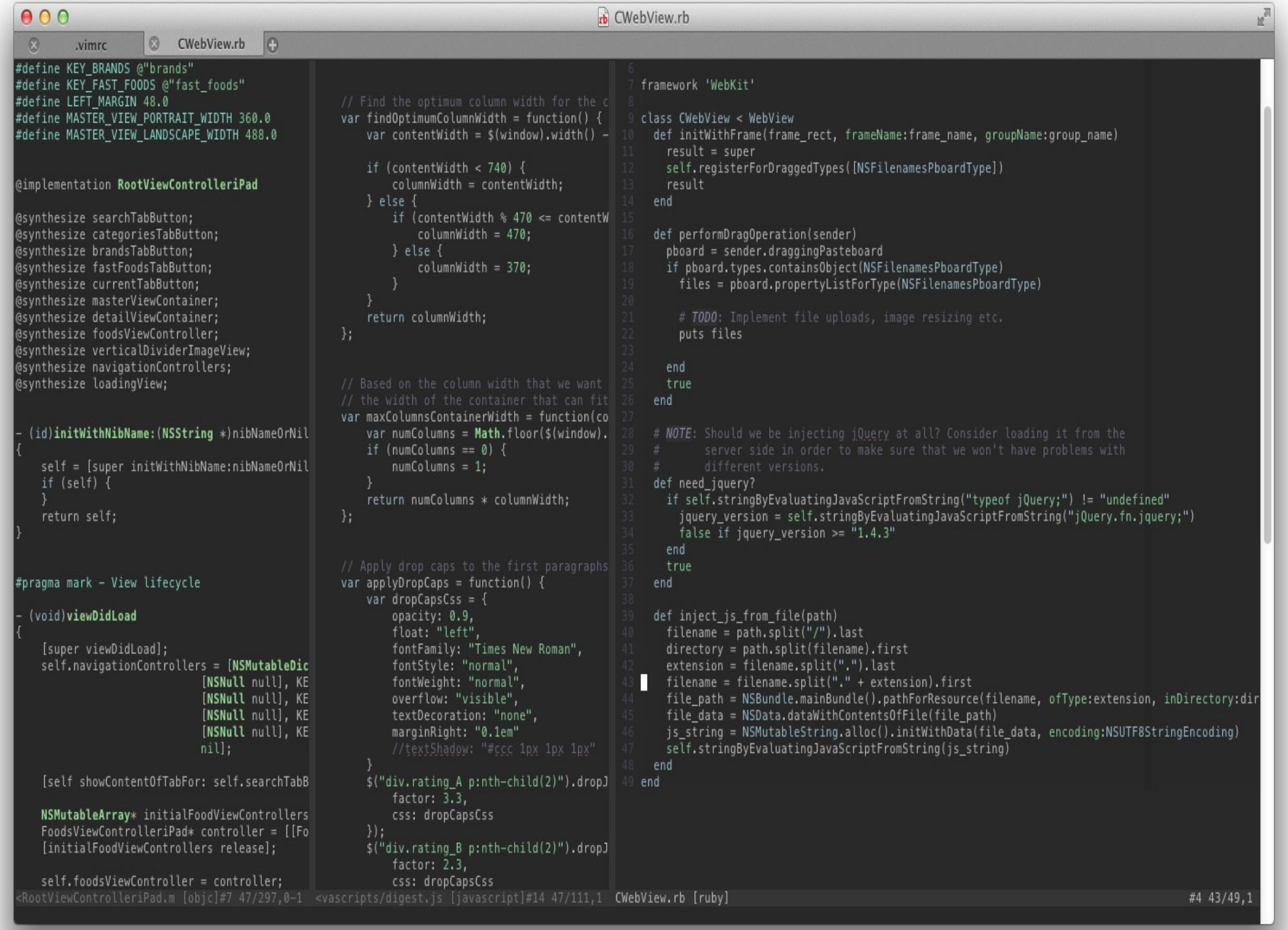
```
" Press ? for help

.. (up a dir)
/home/mike/src/gridlab-d/core/
▸ 90Houses/
▸ 9Houses/
▸ autotest/
▸ gridlab-d/
▸ gui/
▸ kill/
▸ odbc++/
▸ rt/
▸ simple/
▸ test/
  aggregate.c
  aggregate.h
  aggregate.o
  build.h
  class.c
~/src/gridlab-d/core
```

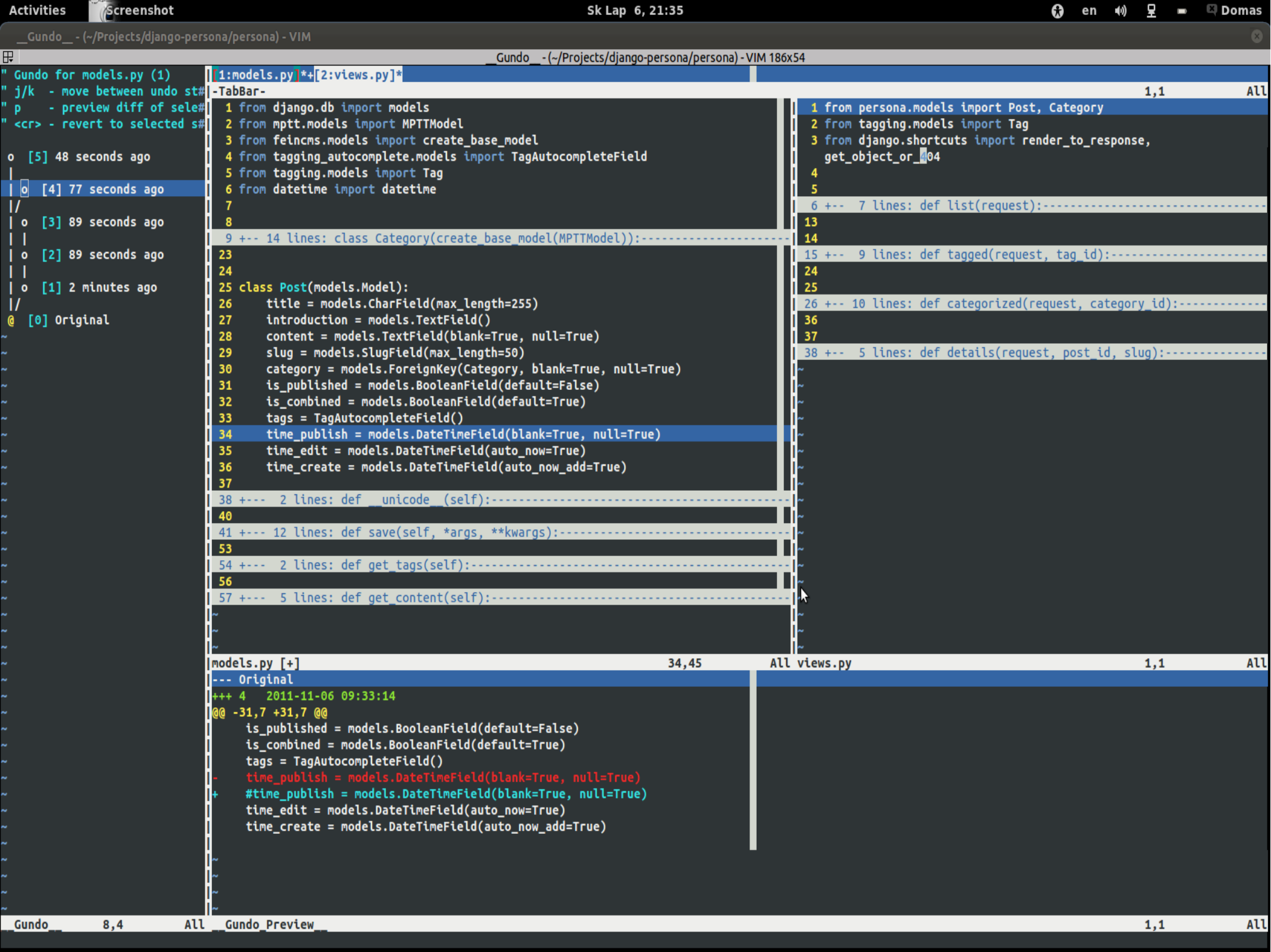
```
208     return oname;
209 }
210
211 /** Get the unit of an object, if any
212  **/
213 char *object_get_unit(OBJECT *obj, char *name){
214     static UNIT *dimless = NULL;
215     PROPERTY *prop = object_get_property(obj, name);
216
217     if(prop == NULL){
218         throw_exception("property '%s' not found in object '%s'",
    name, object_name(obj));
219         /* TROUBLESHOOT
220             The property for which the unit was requested does not
    exist.
221             Depending on where this occurs it's either a bug or an
    error in the model.
222             Try fixing your model and try again.  If the problem
    persists, report it.
```

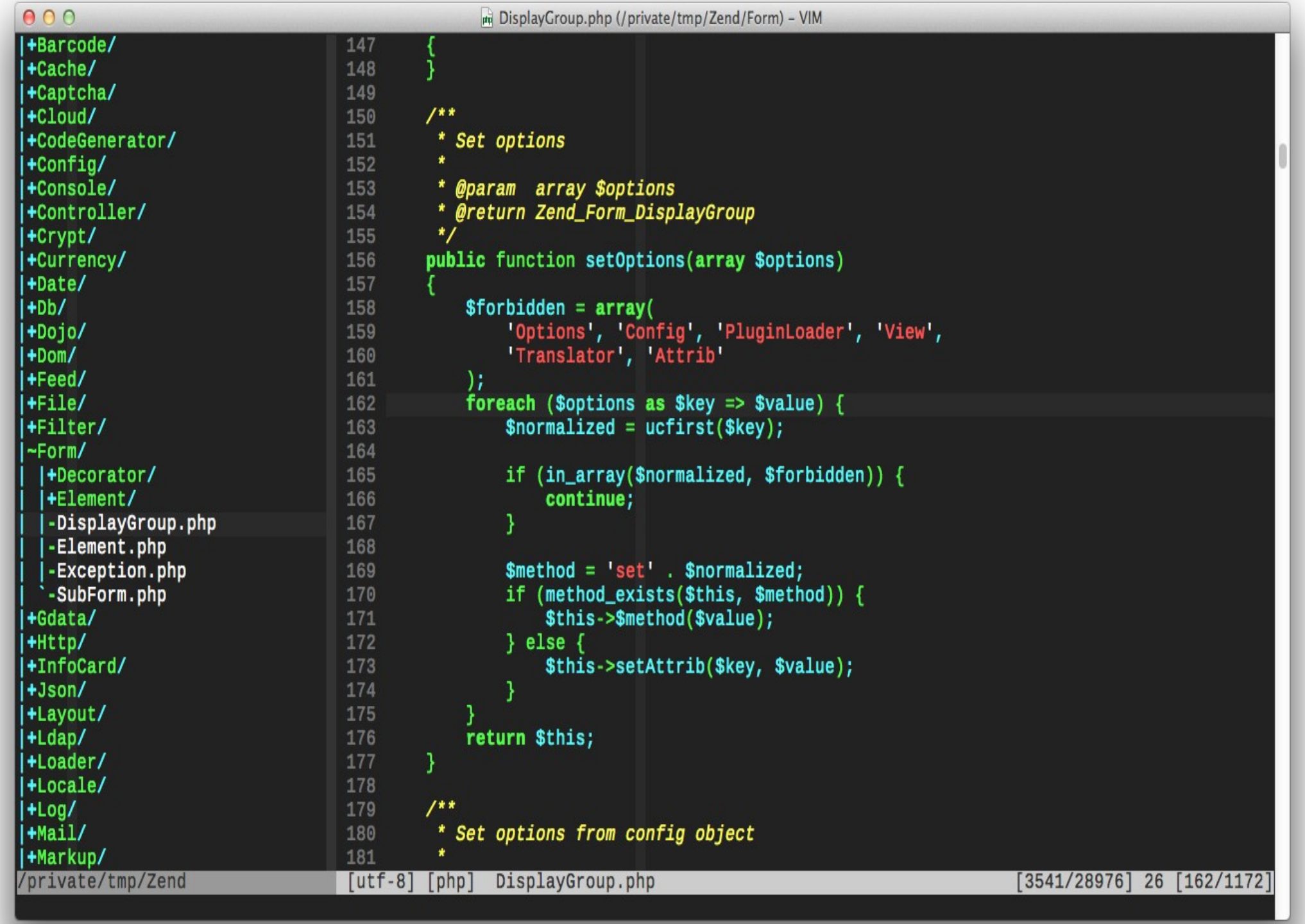
```
BR: master  object.c                    10% | LN 214:5
-- INSERT --
```

```
381             *out++ = hex(hi)*16 + hex(lo);
382         }
383         else
384             *out++ = *in;
385     }
386     *out='\0';
387     strcpy(buffer,result);
388 }
389
390 int http_xml_request(HTTP *http,char *uri)
391 {
392     char arg1[1024]="", arg2[1024]="";
393     int nargs = sscanf(uri,"%1023[^/=\r\n]/%1023[^\r\n=]",arg1,arg2);
394     char *value = strchr(uri,'=');
395     char buffer[1024]="";
396     OBJECT *obj=NULL;
397     char *id;
398
399     /* value */
400     if (value) *value++;
401
402     /* decode %.. */
403     http_decode(arg1);
404     http_decode(arg2);
405     if (value) http_decode(value);
406
407     /* process request */
408     switch (nargs) {
409
410         /* get global variable */
411         case 1:
412
413             /* find the variable */
414             if (global_getvar(arg1,buffer,sizeof(buffer))==NULL)
415             {
416                 output_error("global variable '%s' not found", arg1);
417                 return 0;
418             }
419
```

```
INSERT | BR: master | server.c    unix | utf-8 | c  46% | LN 400:24
```


CWebView.rb
.vimrc
CWebView.rb
#define KEY_BRANDS @"brands"
#define KEY_FAST_FOODS @"fast_foods"
#define LEFT_MARGIN 48.0
#define MASTER_VIEW_PORTRAIT_WIDTH 360.0
#define MASTER_VIEW_LANDSCAPE_WIDTH 488.0
@implementation RootViewControlleriPad
@synthesize searchTabButton;
@synthesize categoriesTabButton;
@synthesize brandsTabButton;
@synthesize fastFoodsTabButton;
@synthesize currentTabButton;
@synthesize masterViewContainer;
@synthesize detailViewContainer;
@synthesize foodsViewController;
@synthesize verticalDividerImageView;
@synthesize navigationControllers;
@synthesize loadingView;
- (id)initWithNibName:(NSString *)nibNameOrNil
{
self = [super initWithNibName:nibNameOrNil
if (self) {
}
return self;
}
#pragma mark - View lifecycle
- (void)viewDidLoad
{
[super viewDidLoad];
self.navigationControllers = [NSMutableDic
[NSNull null], KE
[NSNull null], KE
[NSNull null], KE
[NSNull null], KE
nil];
[self showContentOfTabFor: self.searchTabB
NSMutableArray* initialFoodViewControllers
FoodsViewControlleriPad* controller = [[Fo
[initialFoodViewControllers release];
self.foodsViewController = controller;
// Find the optimum column width for the c
var findOptimumColumnWidth = function() {
var contentWidth = $(window).width() -
if (contentWidth < 740) {
columnWidth = contentWidth;
} else {
if (contentWidth % 470 <= contentW
columnWidth = 470;
} else {
columnWidth = 370;
}
}
return columnWidth;
};
// Based on the column width that we want
// the width of the container that can fit
var maxColumnsContainerWidth = function(co
var numColumns = Math.floor($(window).
if (numColumns == 0) {
numColumns = 1;
}
return numColumns * columnWidth;
};
// Apply drop caps to the first paragraphs
var applyDropCaps = function() {
var dropCapsCss = {
opacity: 0.9,
float: "left",
fontFamily: "Times New Roman",
fontStyle: "normal",
fontWeight: "normal",
overflow: "visible",
textDecoration: "none",
marginRight: "0.1em"
//textShadow: "#ccc 1px 1px 1px"
}
$("div.rating_A p:nth-child(2)").dropJ
factor: 3.3,
css: dropCapsCss
});
$("div.rating_B p:nth-child(2)").dropJ
factor: 2.3,
css: dropCapsCss
framework 'WebKit'
class CWebView < WebView
def initWithFrame(frame_rect, frameName:frame_name, groupName:group_name)
result = super
self.registerForDraggedTypes([NSFilenamesPboardType])
result
end
def performDragOperation(sender)
pboard = sender.draggingPasteboard
if pboard.types.containsObject(NSFilenamesPboardType)
files = pboard.propertyListForType(NSFilenamesPboardType)
# TODO: Implement file uploads, image resizing etc.
puts files
end
true
end
NOTE: Should we be injecting jQuery at all? Consider loading it from the
server side in order to make sure that we won't have problems with
different versions.
def need_jquery?
if self.stringByEvaluatingJavaScriptFromString("typeof jQuery;") != "undefined"
jquery_version = self.stringByEvaluatingJavaScriptFromString("jQuery.fn.jquery;")
false if jquery_version >= "1.4.3"
end
true
end
def inject_js_from_file(path)
filename = path.split("/").last
directory = path.split(filename).first
extension = filename.split(".").last
filename = filename.split("." + extension).first
file_path = NSBundle.mainBundle().pathForResource(filename, ofType:extension, inDirectory:dir
file_data = NSData.dataWithContentsOfFile(file_path)
js_string = NSMutableString.alloc().initWithData(file_data, encoding:NSUTF8StringEncoding)
self.stringByEvaluatingJavaScriptFromString(js_string)
end
end
<RootViewControlleriPad.m [objc]#7 47/297,0-1
<vascripts/digest.js [javascript]#14 47/111,1
CWebView.rb [ruby]
#4 43/49,1


Activities
Screenshot
Sk Lap 6, 21:35
en
Domas
__Gundo__ - (~/Projects/django-persona/persona) - VIM
__Gundo__ - (~/Projects/django-persona/persona) - VIM 186x54
" Gundo for models.py (1)
" j/k - move between undo st#
" p - preview diff of sele#
" <cr> - revert to selected s#
o [5] 48 seconds ago
| o [4] 77 seconds ago
|/
| o [3] 89 seconds ago
| |
| o [2] 89 seconds ago
| |
| o [1] 2 minutes ago
|/
@ [0] Original
[1:models.py]*+[2:views.py]*
-TabBar-
1,1 All
1 from django.db import models
2 from mptt.models import MPTTModel
3 from feincms.models import create_base_model
4 from tagging_autocomplete.models import TagAutocompleteField
5 from tagging.models import Tag
6 from datetime import datetime
7
8
9 +-- 14 lines: class Category(create_base_model(MPTTModel)):----------
23
24
25 class Post(models.Model):
26 title = models.CharField(max_length=255)
27 introduction = models.TextField()
28 content = models.TextField(blank=True, null=True)
29 slug = models.SlugField(max_length=50)
30 category = models.ForeignKey(Category, blank=True, null=True)
31 is_published = models.BooleanField(default=False)
32 is_combined = models.BooleanField(default=True)
33 tags = TagAutocompleteField()
34 time_publish = models.DateTimeField(blank=True, null=True)
35 time_edit = models.DateTimeField(auto_now=True)
36 time_create = models.DateTimeField(auto_now_add=True)
37
38 +--- 2 lines: def __unicode__(self):----------
40
41 +--- 12 lines: def save(self, *args, **kwargs):----------
53
54 +--- 2 lines: def get_tags(self):----------
56
57 +--- 5 lines: def get_content(self):----------
1 from persona.models import Post, Category
2 from tagging.models import Tag
3 from django.shortcuts import render_to_response,
get_object_or_404
4
5
6 +-- 7 lines: def list(request):----------
13
14
15 +-- 9 lines: def tagged(request, tag_id):----------
24
25
26 +-- 10 lines: def categorized(request, category_id):----------
36
37
38 +-- 5 lines: def details(request, post_id, slug):----------
models.py [+] 34,45 All
views.py 1,1 All
--- Original
+++ 4 2011-11-06 09:33:14
@@ -31,7 +31,7 @@
is_published = models.BooleanField(default=False)
is_combined = models.BooleanField(default=True)
tags = TagAutocompleteField()
- time_publish = models.DateTimeField(blank=True, null=True)
+ #time_publish = models.DateTimeField(blank=True, null=True)
time_edit = models.DateTimeField(auto_now=True)
time_create = models.DateTimeField(auto_now_add=True)
__Gundo__ 8,4 All
__Gundo_Preview__ 1,1 All


DisplayGroup.php (/private/tmp/Zend/Form) – VIM
|+Barcode/
|+Cache/
|+Captcha/
|+Cloud/
|+CodeGenerator/
|+Config/
|+Console/
|+Controller/
|+Crypt/
|+Currency/
|+Date/
|+Db/
|+Dojo/
|+Dom/
|+Feed/
|+File/
|+Filter/
|~Form/
| |+Decorator/
| |+Element/
| |-DisplayGroup.php
| |-Element.php
| |-Exception.php
| `-SubForm.php
|+Gdata/
|+Http/
|+InfoCard/
|+Json/
|+Layout/
|+Ldap/
|+Loader/
|+Locale/
|+Log/
|+Mail/
|+Markup/
147 {
148 }
149
150 /**
151 * Set options
152 *
153 * @param array $options
154 * @return Zend_Form_DisplayGroup
155 */
156 public function setOptions(array $options)
157 {
158 $forbidden = array(
159 'Options', 'Config', 'PluginLoader', 'View',
160 'Translator', 'Attrib'
161);
162 foreach ($options as $key => $value) {
163 $normalized = ucfirst($key);
164
165 if (in_array($normalized, $forbidden)) {
166 continue;
167 }
168
169 $method = 'set' . $normalized;
170 if (method_exists($this, $method)) {
171 $this->$method($value);
172 } else {
173 $this->setAttrib($key, $value);
174 }
175 }
176 return $this;
177 }
178
179 /**
180 * Set options from config object
181 *
/private/tmp/Zend [utf-8] [php] DisplayGroup.php [3541/28976] 26 [162/1172]

What makes Vim different?

- It is a Modal text editor (explained later)
- When editing with vim you rarely need to take your hands away from the home keys
- Vim is incredibly powerful and includes a huge array of features
- Vim is very small, not resource heavy and starts very quickly
- Because it has been around for so long, there is a massive community of dedicated users

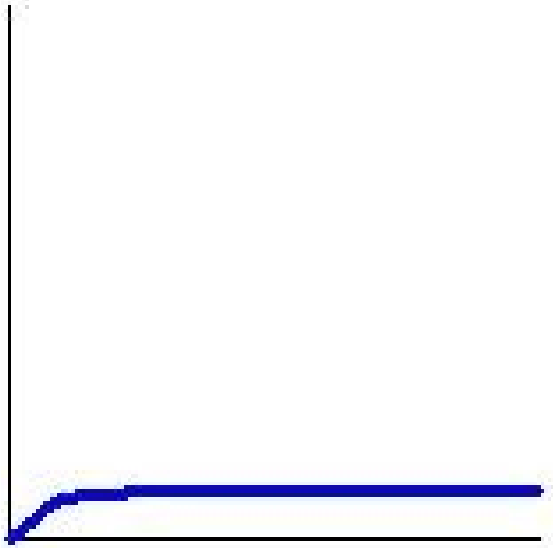
Vim has a steep learning curve!

- Getting to know vim is difficult, but it is very worth it
- Learning vim now will pay off over and over
- There's pretty much always something to learn, and very few could call themselves experts

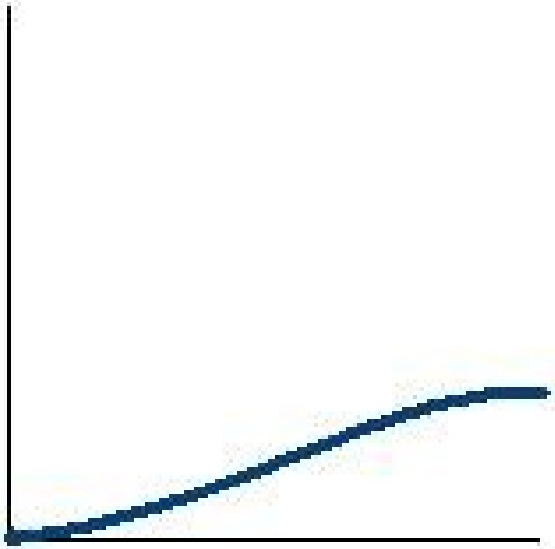
Classical learning curves for some common editors

11-17-04

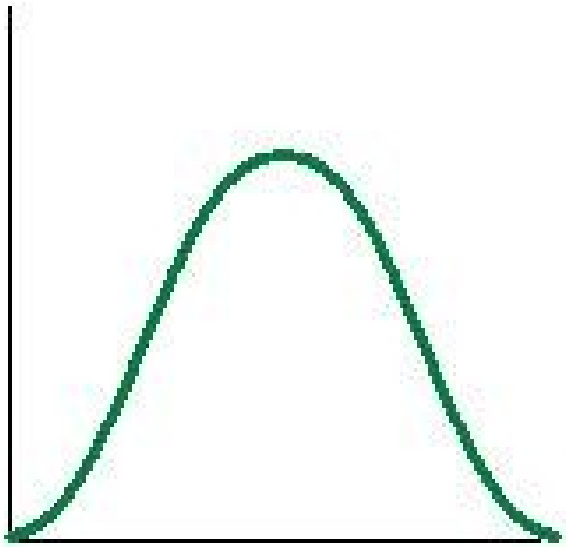
Notepad



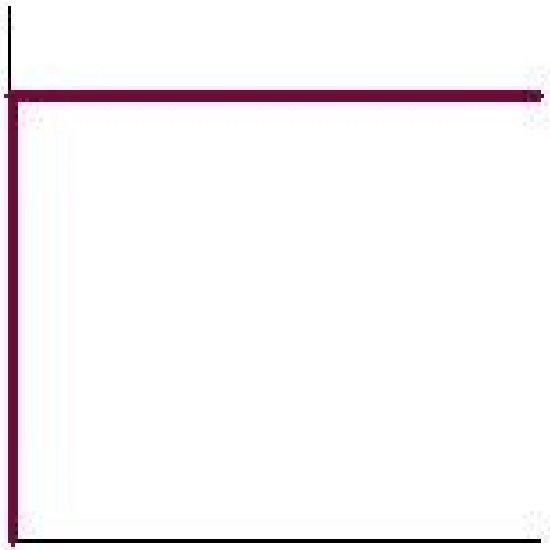
Pico



Visual Studio



vi



emacs

What is a Modal text editor

- Modal means using modes
- There are 6 modes in vim, and depending on which mode you're in vim will treat keyboard input differently
- Because of this you never need to move your hands from the home keys, as all the characters from [a..z] can be interpreted as commands in different modes
- Today we will focus on 2 modes to keep things simple

What are the modes

- **Normal (Command mode)**
- **Insert (Enter text)**
- Visual (Select things visually)
- Command Line (Run commands and scripts)
- Ex Mode (Used for optimisation)
- Select (rarely used – CUA complient)
- We will be focusing on Normal and Insert Mode

Normal Mode

- Normal mode is the most commonly used mode.
- It takes different keyboard inputs as commands to manipulate text.
- It is case sensitive
- Eg:
  p – paste
  u – undo
  i – enter insert mode
- You might find the name counter-intuitive as you don't enter text in this mode

Insert Mode

- Use this mode to insert text
- Keyboard values are taken literally
- Pressing 'esc' will bring you back to command mode
- Enter this mode from normal mode a number of ways, Eg.
  i – Insert text at the cursor
  A – Append text to the end of the line
  r – replace the current character

Command line mode

- Command-line mode is used to enter commands in a command line
- Typical commands include find & replace, write a file, quit, enable line numbers etc.
- You can enter this mode by pressing “**:**” in normal mode
- You can exit this mode by running the command or pressing **ctrl-c**

If something goes wrong

- The different modes and behaviours of vim can be confusing
- If something goes wrong, you can almost always go back to normal mode by pressing “**esc**” or “**ctrl-c**”
- From there you can undo/save/quit
- If things are randomly spazzing out, check that you don't have caps lock on

Introduction to common commands

Absolute minimum

- Open up a terminal and run “vim example.c”
- When you start vim you will automatically be in normal mode and can't enter text
- To enter text – press “**i**” and type in some code
- When you have written something, press “**esc**” to go back to normal mode
- To save the file, enter command line mode with “**:**” and type “**w**” and enter
- To quit type “**:q**” and enter
- So to write and quit, you can run “**:wq**” (remember to type these, and most commands, one at at time)

Summary of absolute minimum

- You opened vim
- You went into insert mode with “**i**”
- You typed some text
- You went back to normal mode with “**esc**”
- You wrote the file and quit with “**:wq**”
- ez

Saving/Quitting

- **:q** – Quit
- **:q!** - Quit without saving
- **:w** – Write
- **:w *filename*** – save as filename
- **:wq** – Write and quit
- **e!** - Undo all changes since the last save
- **:o *filename*** – Open filename

Vimtutor

- We're going to continue with the basics, trying a load of commands and recapping at the end
- Type “**vimtutor**” in your terminal
- Vimtutor lets you edit text and play around freely, and includes a basic vim tutorial
- Nb - It does not listen to your config files for some reason

Basic Movement

- To move around, do:
  h – left
  j – down
  k – up
  l – right
- Don't argue with this, it will become natural with time
- DON'T USE ARROW KEYS – as tempting as it is it just takes away the benifits you get from your hands being on the home keys

Basic Movement Cont'd

- **0** (that's a zero) – Go to beginning of line
- **^** - Go to the first non-whitespace character
- **$** - Go to the end of the line
- **%** - Move to matching parenthesis, when the cursor is over one of them
- **gg** – Go to the top of the file
- **G** – Go to the bottom of the file
- **54gg** – Go to line 54
- And remember, **HJKL** – Left, Down, Up, Right

More practical movement

- **w** – Move to the beginning of the next **w**ord
- **b** – Move **b**ack to the beginning of a word
- **e** – Move to the **e**nd of the current word
- **}** - Move to start of next paragraph
- **{** - Move to start of previous paragraph
- **Ctrl-u** – Scroll half a page **u**p
- **Ctrl-d** – Scroll half a page **d**own
- **zz** – center around your cursor (very handy)

Basic Editing

- **i** – **i**nsert text at the cursor
- **a** – **a**ppend text after the cursor
- **I** – **I**nsert text at the beginning of the line
- **A** – **A**ppend text to the end of the linee
- **u** - **u**ndo
- **r** – **r**eplace the current character with another
- After editing text, always go back to Normal Mode with ESC, make this a habit!

Basic Editing Cont'd

- Generally when you delete something in vim, it is saved to a buffer and can then be pasted – Tf. most delete commands act the same as 'cut'
- **x** – delete the character under the cursor
- **dd** – **d**elete a line
- **cc** – **c**hange a whole line
- **d$** - delete from the cursor to the end of the line
- **yw** – **y**ank (copy) a **w**ord
- **yy** – **y**anks (copies) the line
- **P –** paste before cursor
- **p** – paste after cursor

More practical editing

- **dw – d**elete **w**ord
- **3dd** – **d**elete 3 lines
- **cw** – **c**hange **w**ord, change the text from the cursor to the end of the word
- **ctrl-r** – **r**edo an undo
- . (that's a full stop) – Do the last command again
- **s** – **s**ubstitute (try S)
- **o** – Start writing in next line (O for previous line)

Visual Mode

- Visual mode lets you select things visually and perform commands on selections
- To enter visual mode, press v and move the cursor
- You can then perform commands like **y** to yank, **d** to delete, **c** to change
- **V** (capital **v**) – select visually by line

Search

- To search, hit **/** (forward slash) and type in your query and enter
- To find the **n**ext instance of the query, hit **n**
- To find the previous instance of a query, hit **N**
- To search backwards use **?** instead of **/**
- For reference, the config option “set hlsearch” will make search terms hilighted in the text, see http://vim.wikia.com/wiki/Highlight_all_search_pattern_matches

Find/Replace

- When you hit **:** (colon) you enter vims command line
- To replace foo with bar in your text, run the following **:%s/foo/bar/g**
- Explained:
- the **:** enters command line mode
- the **%** indicates go through every line in the file
- the **s** is for **s**ubstitute and takes the paramters foo and bar
- the **g** means replace all instances

Find/Replace cont'd

- Selective find/replace
  Hilight the text you want with **v** or **V**
  Press **:** and it will have specified the range of that text automatically
  Then do **s/foo/bar/g**
- **:.s/foo/bar/g** – Replace only on current line
- **:8,22s/foo/bar/g** – Only replace between line 8 and 22

Tips and tricks

Useful Commands

- **gg=G** – Formats all of your code
  Explained:
  gg – Go to top of file
  = - Format
  G – until end of file
- You can run your systems shell commands with ! in the command line, eg:
- **:!screen -dr –** Open screen, check irc
- **:!ls** – List files

Useful Commands Cont'd

-) - Navigate text by sentence (forward)
- (- Navigate text by sentence (backward)
- **>>** - Indent the line by 1 block
- **<<** - The opposite of the above
- **"** (double apostrophe) - Jump between a search result and the last mark
- **~** - Change the case of the character
- **ci”** - Change all the text between the parenthesis that the cursor is in
- **ci}** – Same thing except for paragraphs

Some more useful commands

- **ctrl-a** – increment number under the cursor
- **ctrl-x** – decrement number under the cursor
- **guu** – Lowercase a line (wheras ~ toggles)
- **gUU –** Uppercase a line
- **w !sudo tee % >/dev/null<CR>:e!<CR><CR>** - If you open a file owned by root in vim, and make changes, this will write it as root
- In the config, the above is mapped to **w!!**
- **:g/foo/#** - See all instances of foo and their line number

Split windows

- You can split windows with **:vs** for **v**ertical **s**plit, or **:sp** for horizontal **sp**lit
- You can then **o**pen files with :**o** *filename*
- You can swap between this windows with **ctrl-w** followed by **h** or **l**
- You can also swap to the next with **ctrl-w ctrl-w**
- You can open the file explorer window in a split window with **:Sex**
  Nb. To open it in current window run with Ex

Block indenting

- Easy, you may be able to figure it out based off previous example
- Hilight the lines with **V** and press right arrow for right indent, left arrow for left indent
- Changing the indent spacing can be done with **:set shitftwidth=4** (abbreviated to **:set sw=4**)

Block Commenting

- Vim doesn't handle this fantastically by default, although there are plugins for it
- **ctrl-v } I // esc esc** – This will comment the paragraph
- **ctrl-v** – Block commenting, one char space at a time
- **}** - From the cursor to the end of the paragraph
- **I** - In block select mode tells VIM to switch to insert mode with the cursor before the first character in the first line of the block. Note the capital I.
- **//** - Input commented chars
- **esc** – Leave insert mode
- **esc** – Indicate that you're not strining more commands

Folding

- You can hide text by folding it to make your code neater
- To do this, hilight your text in visual mode, and do **:fold**
- **zo –** Open the fold
- **zc** – Close the fold
- You can also set vim to fold based on indentation and syntax and others, aswell as save views

Random

- gq} – ??
- doing :set *command* followed by a question mark will tell you the setting, eg.
  :set ff?

Vim Config Explained

Config Binds

- **<f2>** - Paste toggle, for when you're pasting from your systems buffer
- **<f3>** - Tab completion
- **<f4>** - Swaps between header and source
- Need to run ctags -R for the next 2 to work
- **<f5>** - Opens definition in a new tab
- **<f6>** - Opens definition in a vsplit
- **<f7>** - Toggle taglbar
- **<f8>** - Go to definition
- **<f10>** - Nerdtree file explorer

Conclusion

Things that weren't discussed

- Ex mode (see :help holy-grail)
- Marks, Tags
- Tabs
- Buffers
- Mapping
- Advanced Register use
- How to use the help system
- The more advanced plugins
- VimScript
- Emacs