Vimacs

```
[illegible] Searching and [illegible] Replace

inoremap <C-s> <C-o>:call <SID>VM_search_on('/')<CR><C-o>/
inoremap <C-r> <C-o>:call <SID>VM_search_on('?')<CR><C-o>?
inoremap <M-n> <C-o>n
" <M-n> not in Emacs: search again downward (n = search for Next)
inoremap <M-p> <C-o>N
" <M-p> not in Emacs: search again upward (p = search for Previous)
inoremap <C-M-s> <C-o>:call <SID>VM_search_on('/')<CR><C-o>/
inoremap <C-M-r> <C-o>:call <SID>VM_search_on('?')<CR><C-o>?
[illegible]p <M-[illegible]set invhls<CR>
[illegible]remap <M-%> [illegible]o>:c[illegible] <SID>VM_query_replace()<CR>
[illegible]oremap <C-M[illegible] <C-o>[illegible]l <SID>VM_query_replace_regexp()<CR>
Doesn't w[illegible] on Win32?
[illegible]oremap <[illegible]> <CR><C-o>?<Up>

[illegible]mmand! [illegible]ueryReplac[illegible] call [illegible]VM_qu[illegible]place(
[illegible]mman[illegible] QueryReplac[illegible]gexp [illegible]l <SID>[illegible]uery_re[illegible]e_re[illegible]p()

S[illegible]rching is a b[illegible] tricky [illegible]ause we [illegible] to emul[illegible] Emacs' [illegible]
[illegible]arching again [illegible] <C-s> [illegible] <C-r> i[illegible] pressed _i[illegible]de_ [illegible] search
commandline.  V[illegible] has no e[illegible]valent t[illegible] this, so w[illegible] must [illegible] a bit
on-the-fly rem[illegible] trickery [illegible]opular in [illegible]ke-style games [illegible] provi[illegible]
different func[illegible], [illegible]pending o[illegible]ther yo[illegible] in [illegible]
not.

We must map <C-g> and <CR> because we have to undo the map trickery that we
set up when we abort/finish the search.  All in all, it's not too complex
when you actually look at what the code does.

Note that <C-c> in Emacs is functionally the same as <CR>.

LetDefault g:VM_SearchRepeatHighlight 0

function! <SID>VM_search_on(search_dir)
  let s:incsearch_status = &incsearch
  [illegible]lazyredr[illegible]status = &lazyredraw
```

- Conrad Parker: “Dude, I gotta get you doing some serious hacking projects”
- Erik de Castro Lopo: “Oh, so you’re the insane guy that K was talking about”
- Wichert Akkerman: “Gross. Horror. Vim abuse. Mind if I include this as an example in the Debian vim packages? :)”
- Henry Gebhardt: “Vimacs, as ridiculously as it sounds, is a great idea”

Yep, it’s Emacs… in Vim.

André

Download Vimacs

The latest version of Vimacs is 0.95.

- Unix: vimacs-0.95.tar.gz
- Windows: vimacs-0.95.zip

Vimacs hasn't been updated in about four years, but hey, it still works. (Thank you Vim, for having a sane notion of backwards compatibility.)

The files are essentially the same, except that the installation instructions are a bit more tailored toward the platform that you're using.

What's Vimacs?

In a nutshell, Vimacs is Vim emulating Emacs. (If you've never heard of Vim nor Emacs, you're probably in the wrong place right now). Vimacs (**Vi**m-improved E**macs**) is a Vim plugin, which brings Emacs's extensive key bindings and modeless editing features to the Vim world, while retaining Vim's powerful moded editing style.

It's an emulation of Emacs using Vim: you get modeless editing inside moded editing. If you're familiar with Vim, don't worry: Emacs emulation only operates while you're in Vim's Insert mode. Vimacs is based on the keymaps of GNU Emacs 21 and XEmacs, so if you're familiar with them, you'll feel right at home in Vimacs.

Most (if not all) of the common key maps in Emacs are present in Vimacs, such as <C-Space>, <C-w> and <C-y> to mark, kill and yank regions, <C-x><C-f> to find a file, and <C-x>2 to split a window. However, you can now take advantage of the incredible power of Vim's Normal and Command modes with a touch of the <Esc> key.

Why?

If you're a long-time Vim user, you retain Vi's powerful moded editing paradigm while gaining all the benefits of Emacs's modeless editing. You have nothing to lose! Since Vimacs only emulates Emacs when you're in Insert mode, you may not even notice that it's there.

Of course, after a while, you may find that the <C-a> and <C-e> keys become second nature to you, just as you're used to the h, j, k, and l keys for movement. Some of the keys familiar to you in Insert mode have been changed, but you won't take long to get used to Emacs's keys: just like Vim, Emacs's initially obscure key layout will reward you later on for its efficiency. Emacs experts are just as fast as Vim experts in manipulating text, and as a bonus, you start becoming familiar with Emacs keybindings, which are gradually becoming more pervasive in Unix applications. (Even Mac OS X supports some Emacs keys in its dialog boxes!)

If you're an Emacs user, you'll find that Vimacs offers most of the comforts that you're used to in Emacs, with one significant advantage: blazing speed. Vim is optimised for efficient text editing, whereas Emacs is optimised for extensibility, and sacrifices speed because of it. In addition, I feel that Vim is much more straightforward to customise than Emacs, it has better documentation, and has better cross-platform support.

Of course, a major disadvantage of Vim compared to Emacs is that you don't have the enormous number of Emacs add-ons that have been integrated into the editor over the last few decades. However, you may be surprised at how many of those add-ons are incorporated into Vim itself, for speed. For example, dynamic abbreviation, parenthesis highlighting, and C indenting are all built into Vim. Even complex add-ons, such as Emacs' Grand Unified Debugger, have been implemented in Vim via add-on scripts. Vim does offer its own scripting language if you want to extend it, and if that isn't enough, Vim 6.1 supports Perl, Python, Tcl, OLE, and Ruby as scripting languages: far more choices than just LISP!

Ultimately, the differences between the two editors come down to different design goals: Emacs is aimed at being everything for everyone, and Vim is aimed at being a pure text editor. Vimacs opens up Vim to Emacs users (and everyone else) by allowing you to work with it without having to press <Esc> all the time, so you aren't forced to work with Vi's moded editing style. If you've shied away from Vim because of its Vi-style editing, give Vimacs a shot!

History

In classic open-source spirit, Vimacs was created to "scratch an itch". I used to use Emacs all the time; it's a fantastic tool for doing all kinds of things (even text editing…). However, it suffers from some problems which I feel are unsolvable without a drastic redesign:

- Speed. Emacs doesn't even begin to approach Vim for raw speed; Vim can load and fully syntax-colour a file before Emacs even displays its window. (Using gnuclient doesn't count!) One of the reasons for Emacs's slow speed compared to many other text editors is that it uses LISP for everything. Now, using LISP for everything has many advantages, including making Emacs is completely extensible. I'm all for extensibility, but in this case, Emacs is too slow (for me) because of it. In my eyes, you use an editor so much in the UNIX environment that you really want it to be as fast as possible. Even though Emacs is extensible++, it's slow enough that I never load it for quick editing jobs.

- Customisation is too hard. Emacs exposed so much of its complexity and internals to you that it would befuddle your average user who just wanted to change a few settings. I didn't feel like learning LISP just to customise the editor a bit; M-x customize made this much better, but it's still still unnecessarily hard to change something which isn't in that list of options. Try writing your own syntax file or colour scheme, for instance; other editors usually make this much easier. Of course, you can argue that learning the language of Emacs (i.e. LISP) leads to learning a powerful generic language which can be used in many places outside of Emacs, but hey, some of us want to get other work done, too.

Please don't get me wrong — I'm not flaming Emacs, and I'm definitely not saying that it's a bad editor. It's a fantastic editor; it's arguably one of the most powerful pieces of software on the planet. It's just not the kind of editor that I like to use on a daily basis. I don't need most of its ninja-fu abilities, but some of them are incredibly helpful. I can't live without things like dynamic abbreviation (the magical M-/ key), and having the editor understand the syntax of the file you're editing. Once you're used to those features, life is simply miserable without them.

Vimacs was created because I was looking for an alternative to Emacs, and I found none. I wanted a cross-platform editor, since I work a lot on both Win32 and UNIX systems, and I wanted it to be fast and easy to customise, yet still have most of the power that Emacs gave me when I was editing text files, or coding. I tried every single editor I found on Freshmeat and Ibiblio, but I still couldn't find one that I liked. Vim would have been perfect, but I could never get used to the moded style of editing. Pressing <Esc> all the time drove me nuts. (I'm not saying that moded editing sucks, mind you, I'm just saying that I personally hated being forced into that kind of editing style.)

After toying around with Vim's scripting language, I decided to do something to end my misery, and I figured that Vim's scripting capabilities were powerful enough that it could be used to emulate Emacs quite well. I wouldn't be able to emulate everything, of course, but I'd be able to emulate Emacs well enough that it'd have all the key bindings that I was used to. So, Vimacs was born. Vim by itself is a powerful editor; I loved everything about it except for the moded editing paradigm. Vimacs is simply something which makes Vim more accessible to the non-Vi crowd, and it does so in a way which impacts existing Vim users in a very small way.