

INTRODUCTION TO VIM

John Kerl

University of Arizona

Department of Mathematics

Software Interest Group

September 7, 2005

Overview

- Why
- What
- How

Why use a power editor such as vim or emacs?

- Notepad, gedit, et al. provide easy point-and-click use, with very low learning curve. These are great tools, and they suffice for many people much of the time.
- Much of our time in math, sciences, and engineering (as well as many other fields!) is spent editing text files: C/C++ source code, Matlab .m files, .tex files (such as this one!), .html/.css/.php files, etc.
- If you don't enjoy these tasks, make it efficient so that you can spend more time doing what you want to do.
- If you do enjoy these tasks, make it efficient so that you can be more productive.
- A power editor can increase your productivity many times over. For example, the more comfortable and efficient you are at creating \LaTeX files, the more likely you will be to create easy-to-read papers, slides, exams, vita, etc. This benefits your career.

What power editors do

Features:

- Commands can be executed using keystrokes, instead of or in addition to drop-down menus. Keeps you from moving your hands back and forth between the mouse and keyboard.
- Customizable key binding, to abbreviate your most-used commands.
- Syntax highlighting. Not just decoration — helps detect typos early, keeping you from having to decipher cryptic error messages from $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$.
- Macros and scripting to automate repetitive tasks.
- Ability to pipe part or all of a file out to an external program.

What power editors do, continued

- Parenthesis matching. Useful for many programming languages, but especially LISP!
- Tags files for C/C++ programs:
 - Run the ctags program.
 - Edit your C/C++ source code.
 - Push a key (control-] in vim) to move from a function or variable to its definition.
 - Push another key (control-t in vim) to move back.
 - These may be nested — there is a “tag stack”.
 - This makes code archaeology much easier.

vi

- History: vi, short for visual editor, was written by Bill Joy in 1976. It was one of the first full-screen editors (i.e. not a line editor).
- It was designed for use over a 300-baud modem, so it minimized keystrokes. Slow modems are a thing of the past, but minimal number of keystrokes is still desirable.
- vi is present on all Unix-like systems.

vim

- vim, short for vi improved, was written by Bram Moolenaar in the early 90s.
- vim does everything vi does, and much more: multiple-level undo, multiple windows and buffers, macros, block operators, on-line help, syntax highlighting, mouse and GUI support, etc.
- Standard component of all major Linux distributions.

How to use vim

- Fundamental concept: vim is a **modal editor**.
- In most editors, including Emacs, the letter keys are always used for inserting text. This means commands must be entered using physically awkward combinations of control, alt (meta), and/or letter keys — or, mouse menus.
- In vim, you are either in **command mode** or **editing mode**. E.g. the letter j is the same as down arrow when you are in command mode.
- Result: you keep your hands on the keyboard and your eyes on the screen. You accomplish more, in less time.

Getting in and out

- `vim (filename)` — Start editing *(filename)* by typing the vim (or gvim) command at the command prompt.
- `:w (new name)` — save as.
- `:w! (new name)` — save as with replace.
- `:w` — save.
- `:q` — quit.
- `:wq` — save and quit.
- `:q!` — quit, discarding changes.

Moving around

- **h, l, k, j** — left, right, up, down arrow. (You can use the mouse and/or arrow keys too.)
- **control-b, control-f** — page up/down. (You can also use the PageUp and PageDown keys on the keyboard.)
- **0, \$** — move to start/end of line.
- **b, w** — move backward/forward one word.
- **{, }** — move up/down a paragraph.
- **%** — Move to matching bracket, if any. Works for **()**, **[]**, **{}**.
- Etc.

Getting into and out of editing mode

- **i** — Insert at the current position.
- **I** — Insert at the start of the line.
- **A** — Insert at the end of the line.
- **o** — Insert below the current line.
- **O** — Insert above the current line.
- **cw** — Change current word.
- **ESC** — Out of editing mode.

More commands

- **x, X** — Delete current/previous character.
- **r** (*key*) — Replace current character
- **~** — invert case of current character.
- **u** — undo.
- **control-r** — redo.
- **/** (*pattern*)— Forward search.
- **?** (*pattern*)— Reverse search.
- **n, N** — Next/previous match.
- **%s/old word/new word/g** — Global search and replace.

Copying and pasting

- **yy** — Copy (yank) current line.
- **3yy** — Copy current line and the next two.
- **3y}** — Copy the next three paragraphs.
- **dd** — Cut (delete) current line.
- **3dd** — Cut current line and the next two.
- **3d}** — Cut the next three paragraphs.
- **yw** — Copy current word.
- **dw** — Cut current word.
- In general, **y** or **d** followed by any cursor-motion command.
For example, **d\$**.
- **p** — Paste after.
- **P** — Paste before.

Learning curve

I was once a die-hard Emacs user, but found myself at a company where vi was on all workstations, but Emacs was not.

Learning vi, I found that I did not try to memorize all possible commands. Rather, I learned the most useful ones. Later, after I became accustomed to them, I learned some more commands as a way of saving keystrokes.

Different people have different crossover points for productivity vs. amount of things to memorize. If you are learning vi or Emacs, I would suggest the approach I used: memorize what you need. **Memorize more when you need it.** Like playing the piano or learning to type by touch, your fingers will learn what they use.

Also, **gvim** has a nice GUI. If you wish, you can use familiar point-and-click commands, gradually replacing them with more efficient keystrokes as meets your needs.

.vimrc file

The `~/.vimrc` file contains any particular settings you prefer. Some of these look messy, but you only type them in once and forget about the details. Example lines:

- **set vb t_vb=** — turn beep off.
- **set ts=4** — set tabstop to 4.
- **syntax on** — enable syntax highlighting.
- **map ; : —** Do `:wq`, etc. without needing to hold down the shift key.
- **map - :.=^M** — Show current line number. (Note: To enter the last character into your `.vimrc` file, type control-v followed by control-m.)

.vimrc file, continued

- **map \t O%% control-v ESC64A-control-v ESC** — Insert a commented out line of dashes such as the following:

```
%% _____
```

Helps make your .tex files easier to navigate through.

- **map \w :w^M:!latex %^M** — Save current file and run it through latex, using just two keystrokes.

Tip: Leave “xdvi filename.dvi &” running, then alt-tab to bring xdvi forward and back. The xdvi program automatically reloads the .dvi every time it is foregrounded. This enables rapid, iterative development of .tex files. Then, pdflatex when you are ready to print.

Resources

- Type **vimtutor** at the command prompt. The tutorial takes about half an hour.

- The file you are reading is at

<http://math.arizona.edu/~kerl/doc/vimswig/vimswig.pdf>

It was written in L^AT_EX using vim:

<http://math.arizona.edu/~kerl/doc/vimswig/vimswig.tex>

It was typeset using the command “`pdflatex vimswig.tex`”.

- <http://math.arizona.edu/~swig>
- <http://www.vim.org> for on-line documentation and FAQ.
- Fly like the wind!