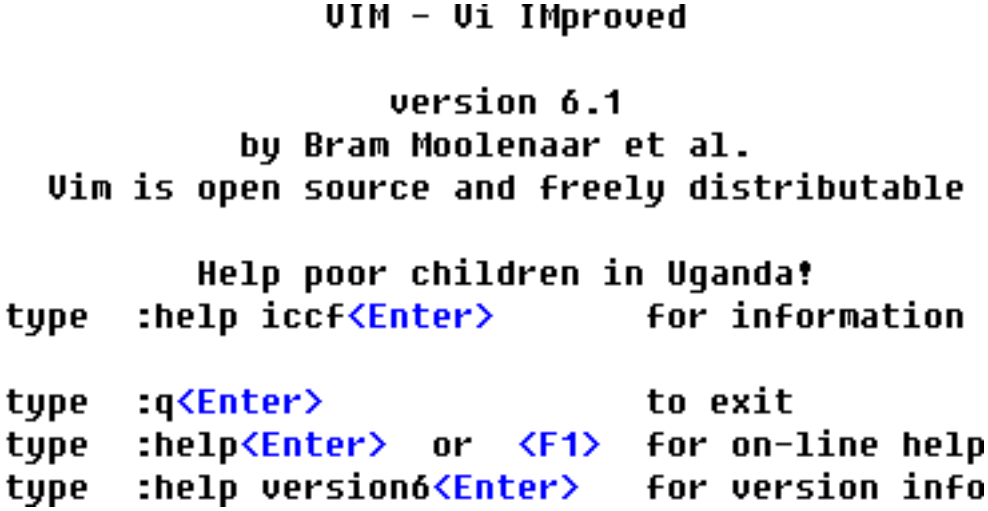
Editing with gVim

Ashley J.S Mills

<ug55axm@cs.bham.ac.uk>

Table of Contents

1. Introduction



Vi pronounced *vee-eye* was written by William Joy in around 1976 at The University of California, Berkeley. Vi was created by hacking various other editors around at the time and writing new code. Here is an interview with the creator: http://www.cs.pdx.edu/~kirkenda/joy84.html. Here is the original documenation for Vi: http://docs.freebsd.org/44doc/usd/12.vi/paper.html.

Vi is available on VMS and Unix systems, it is a small, fast, full-screen editor for terminals. Vi comes with most, if not all, distributions of Unix, Linux, and BSD. Vi is also present on other Unix-like operating systems.

VIM (VI IMproved) is a Vi clone created mainly by Bram Moolenaar, who owns the copyright. VIM is charityware, in that, it is freely distributable but the user is asked to make a donation to charity. VIM can operate in a text based environment but there is a version known as gVim which operates in a GUI environment. The latest version of gVim is available for many operating systems and has full mouse support, clipboard support is available where applicable. Here is the gVim homepage: http://www.vim.org/.

This tutorial will focus on using gVim because it is thought that the target audience will get most utility out of it compared to VIM and Vi. The basic commands are the same with VIM and Vi but Vi lacks certain functionality such as multiple levels of undo. Here are a few benefits of using gVim over Vim and Vi:

- It has menu support which is useful for begginers.
- It directly competes with xemacs.
- It has an "easy" mode for begginers.

Within the file editing community there is somewhat of a feud between users of gVim and users of (ughh!) Emacs. This rivalry is not to be taken too seriously since none of the editors are *THAT* good, see http://www.tuxedo.org/~esr/jargon/html/entry/holy-wars.html.

This document does not intend to cover the subject matter to any significant degree. Its intent is to provide a simple introduction to editing in gVim that is consistent in style with the rest of the tutorials in this tutorial collection. Only a handfull of the most useful commands are introduced. The user is referred to the gVim manual for further information. The gVim manual comes with the gVim installation and is located in the `doc` directory.

2. Installation

Installation of gVim is as easy as following the instructions on the download page: http://www.vim.org/download.php.

2.1. .vimrc

gVim is configured via a file called `.vimrc` on Unix machines and a file called `_vimrc` on Windows machines. Here is an example `.vimrc` which provides a basic working environment with syntax highlighting:

```
set nocompatible " Use gVim defaults

" set tw=80          tw to specify a default text width
set fo=tcrq         " fo to specify default formatoptions
                        " t auto-wraps text using textwidth
                        " c auto-wraps comments using textwidth
                        " r auto-inserts the current comment leader
                        " q allows formatting of comments

   " allow backspacing over everything in insert mode
   set backspace=2

set tabstop=1           " Each Tab has 1_spaces equivalent width
set shiftwidth=2        " Indentation width when using >> and << re-indentation
set nobackup
set expandtab           " Tabs are expanded to spaces

source $VIMRUNTIME/vimrc_example.vim
source $VIMRUNTIME/mswin.vim
behave mswin
```

Notice that the configuration file "sources" an example vimrc file that comes with the gVim distribution called `vimrc_example.vim`; this file defines the syntax highlighting colours. The last two lines are specific to Microsoft Windows machines and should be removed for other systems.

2.2. gVim - Easy

gVim-Easy, which is installed with gVim, has all the functionality of normal gVim but lacks modes. This is especially useful for begginers and people who do not want to, or do not have the time to, learn how to use gVim. Users can benefit from gVim's superior syntax highlighting and auto-indentation while not having to have to learn the, often deemed complex, command set of gVim in order to edit a simple document. It is recommended that readers of this tutorial at least try to learn how to use gVim in normal mode, the learning curve is steep, but, the benefits in speed and usability this confers is worth the investment.

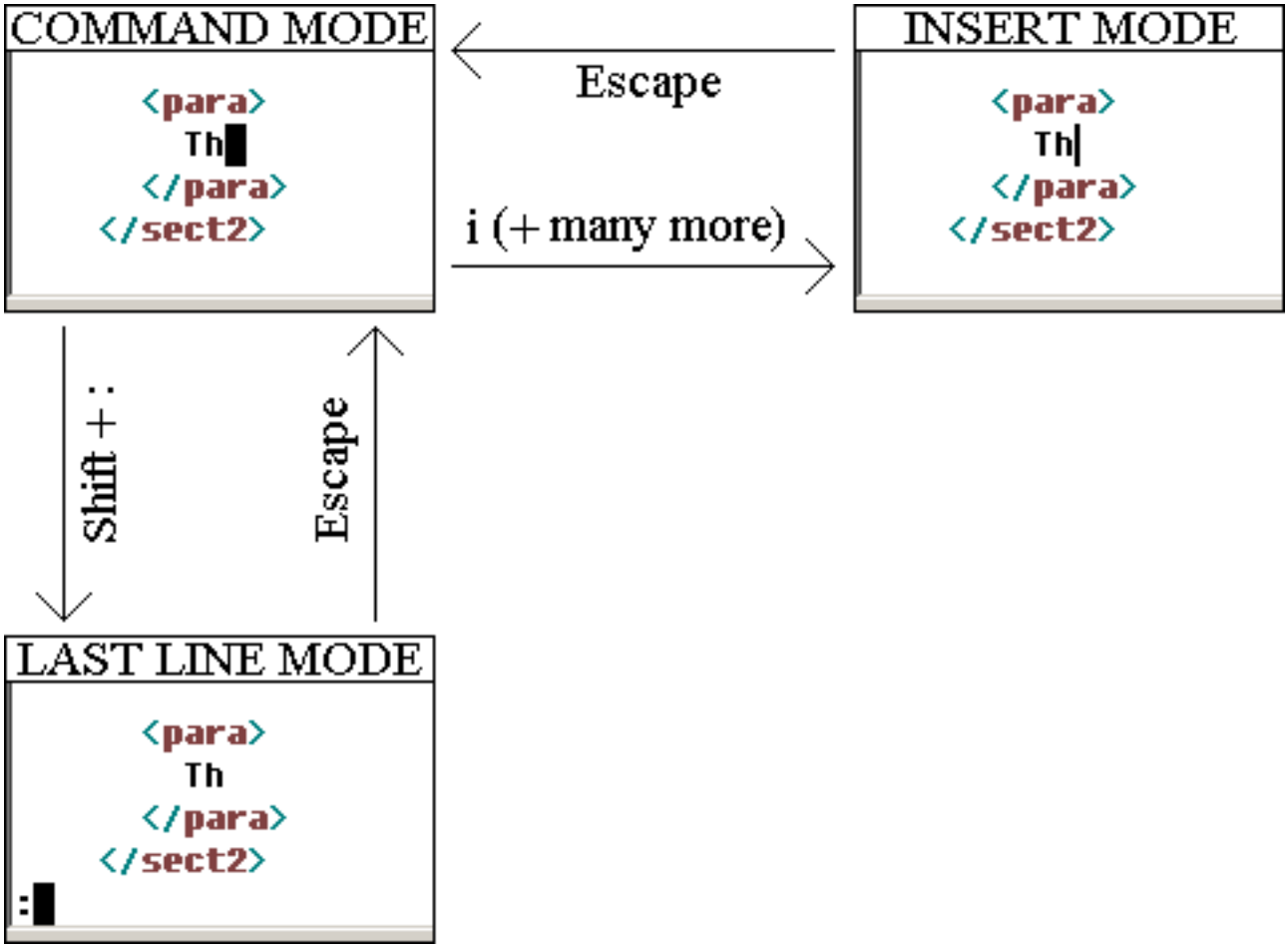
3. Core Commands

3.1. Modes

gVim is a *modal* editor. There is some discrepancy over how many modes gVim has, some consider gVim to have two modes; command mode, and input mode whereas some consider vi to have three modes; command mode, input mode, and 'last line' mode.

This tutorial will assume the latter.

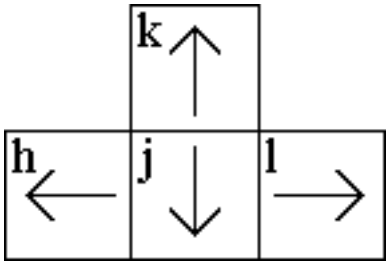
- *Input mode* is used to *input* text into the editing "buffer". This means that whatever is typed appears on the screen, verbatim, as it were. To get into input mode from command mode, press *escape* then press 'i'. To get into input mode from last line mode, press *escape* to enter command mode then follow the instructions for entering input mode from command mode. There are some other ways of entering input mode from command mode, these will be discussed later.
- *Command mode* is used to perform operations on text and to navigate quickly around the document being edited. To enter command mode from input mode *or* last line mode, press the *escape* key.
- *Last line mode* is used to toggle gVim options, perform system operations such as saving, and to perform abstract operations on text such as regular expression search-and-replace. To enter last line mode from input mode, first, press *escape* to enter command mode, then, press **Shift-:**. To enter last line mode from command mode, just press **Shift-:**.



3.2. Text Navigation

One of the most important skills needed to use gVim efficiently is the ability to navigate a document being edited quickly and proficiently. In insert mode, "standard" navigation applies, one may use backspace to delete characters, use the mouse to move the cursor position, and use the cursor keys to move through sentences. One may also use the cursor keys to position the cursor in command mode too.

The cursor keys are intuitive and familiar but their use requires the movement of the users hands away from the 'home' keys of the keyboard thus slowing the user down. It is thus preferable to use the command mode navigation keys:



It can be quite frustrating getting used to the keys but it is well worth the effort. There is a Unix game called hack where the aim of

the game is to navigate ones way through some dungeons down to below level 20 where it is rumored that the Amulet of Yendor lies. The aim of the game is to find the Amulet and get it back out. The keys used to navigate ones character in "hack", in the four primary directions, are the same as those used by gVim hence one could use this game as a fun way to learn the navigational keys of gVim. "hack" comes with the games package of the FreeBSD operating system and is present on decent Unix systems. There is a more recent version called Nethack, which is available for many operating systems, but it does not appear to support the same navigational methods, if I am wrong about this please inform me.

Table 1. gVim Navigation

Key sequence	Effect
k	move cursor up
j	move cursor down
h	move cursor left
l	move cursor right
b	move cursor to beggining of current word
e	move cursor to end of current word
Shift-h	move cursor to top of current screen
Shift-l	move cursor to bottom of current screen
Shift-]	move cursor to beggining of next paragraph
Shift-[	move cursor to beggining of previous paragraph

3.3. Inserting text

Besides basic navigation, there are other important concepts; entering insert mode from command mode can be accomplished in a number of different ways, depending on where one wants to begin inserting characters. The table below shows some of the ways to enter insert mode, the key sequences specified must be performed from *command mode*.

Table 2. Entering insert mode

Key sequence	Effect
i	insert text before cursor position
a	insert text after cursor position
Shift-i	insert text at beginning of line (before first blank)
Shift-a	insert text at end of line
o	insert text after current line (creates new line)
Shift-o	insert text before current line (creates new line)
g-i	insert text at last insert position

There are also a couple of special ways to insert text which don't really fit into the table above, these are shown in the table below:

Table 3. Entering insert mode

Key sequence	Effect
r-AnyCharacter	replace character at cursor position (stays in command mode)
x	deletes character at cursor position (also copies it to clipboard)
s	deletes character at cursor postion then enters insert mode at that position
Shift-r	enters *replace* mode where every character typed overwrites the character under the current cursor position

3.4. Cutting and pasting text

gVim has support for native cut and paste where applicable. For instance, on Windows, one may use **CTRL-X**, **CTRL-C**, and **CTRL-V** to cut, copy and paste text respectively. gVim also provides it's own commands for cutting and pasting areas of text.

Table 4. Cutting and pasting in gVim

key sequence	Effect
Shift-Y	'yank' current line (copies current line)
d-d	deletes current line (and also copies it to clipboard)
p	paste yanked line(s) after current line
Shift-p	paste yanked line(s) before current line
y-w	'yank' from cursor position to beggining of next word
d-w	delete from cursor position to beggining of next word
p	paste yanked word(s) after current cursor position
Shift-p	paste yanked word(s) before current cursor position

3.5. Performing the same operation more than once

Most of the gVim commands can be prefixed with an integer to specify that the command should be performed the number of times indicated by the integer, some examples are shown below.

Table 5. Examples of [count] prefixed command usage

Key sequence	Effect
5-j	move cursor down 5 lines
3-d-w	delete from cursor position to beggining of third word after current word
5-Shift-y	'yank' 5 lines from current line (current line counts as one of the 5)
5-p	paste the yanked text 5 times (in manner depending on whether lines or words were yanked)
3-s	delete the three characters from current cursor position and enter insert mode (substitute)

3.6. System Commands

gVim has menus for performing system operations such as opening files and saving files and so on, however, the system commands are usually quicker. Opening a file from the menu is preferable when one finds it quicker to find the file through the edit-file dialog box than to specify it at the command line. The commands below are all entered in last line mode hence the prefixed ':'.

Table 6. System commands

Key sequence	Effect
:**w**	write the current file
:**w***filename*	write the current file as *filename*
:**e***filename*	edit *filename*, file does not need to exist

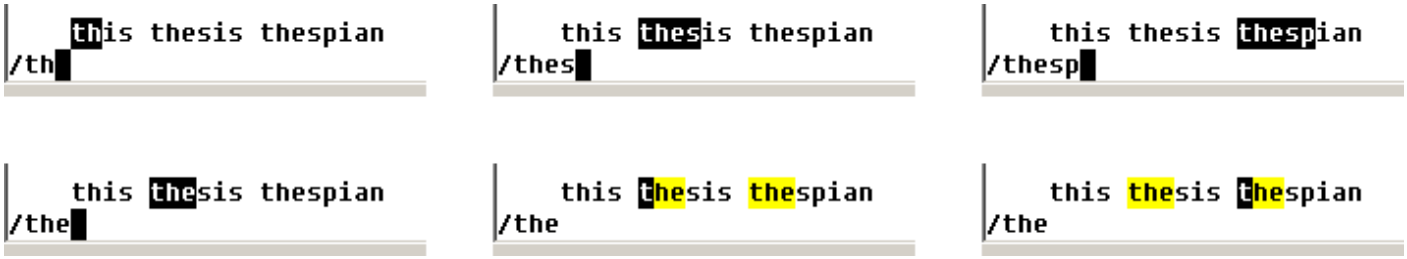
4. Searching

1. Enter search mode by typing **/** from command mode. '/' will appear at the bottom of the screen.

2. Enter a regular expression to search for. Any gVim supported regular expression may be entered, this includes normal words like "dog". Each time the user adds an extra character to the regular expression, gVim will highlight the first match it finds in the document from the character after the cursor position from which search mode was entered. Backspacing is allowed.

3. Press **Enter** to confirm expression to search for. All instances of the expression in the document will be highlighted or if no expression is found "Pattern not found: *pattern*" will be displayed.

4. Use **n** and **Shift-n** to cycle forward backwards through all matches found respectively.



Note

Thespian correctly begins with a capital 'T'.

In the image above, the first row shows gVim matching the first occurance of the expression as more characters are added. The first image of the second row shows the string "the" has been highlighted (backspace was pressed twice from the last image of the first row). The second image of the second row shows the effect of pressing enter to confirm the expression to search for; all the instances of the expression have been highlighted. The last image of the second row illustrates the pressing of **n** to cycle through the instances of the expression found.

5. gVim regular expressions - *Search and Replace*

Regular expressions are one of the most powerful tools available to the gVim user; they can be used to search and replace on sections of text, saving the user much time. The general form of a gVim *Search and Replace* expression is shown below:

:*start_point*,*end_points*/*search_pattern*/*replacement_pattern*/g

The start point can either be an absolute line number, or be relative to the current line. The same applies for the end point. Commands are entered from last line mode. The 'g' at the end is optional, if it is ommited only the first occurance of the pattern will be replaced.

- To represent the current line use '.'
- To represent an absolute line number just use the number.
- To represent *n* lines before the current line use *-n*.
- To represent *n* lines after the current line use *+n*.
- The character '%' can be used to indicate global replacement, it replaces "*start_point*,*end_point*".
- Special characters must be escaped with a backslash if their literal meaning is desired. For example, if one wants to replace "/" one must use "\/".

Table 7. gVim search and replace examples

Command	Effect
:-1,+1s/love/live/g	replaces every occurance of "love" on the line before the current line, the current line and the line after the current line with "live
:.s/pie/potato/	replaces the first occurance of "pie" on the current line with "potato"
:10,30s/</\</g	replaces every occurance of "<" between lines 10 and 30 inclusive with "<"
%s/Chapter5/Chapter6/g	replaces every occurance of "Chapter5" within the file with "Chapter6".
-1,.s/\/\/\/\/g	replaces every occurance of "/" on the line before the current line and the current line with "//"

For more complex regular expression examples, and mode general information about regular expressions, see: Regular Expressions - Ashley J.S Mills [../regexp/regexphome.html].

6. gVim and DocBook

6.1. Useful key mappings

One can write DocBook documents at an incredably faster rate if one maps element entry to key bindings. A directory called `xml` was created in the `ftplugins` directory of the gVim installation. Into this was placed a vim file that contained macros to map key combinations to element insertions. Comma preceeds each mapping, this is convenient because if the user types comma followed by space, nothing happens, but if the user types comma followed by one of the mapped key combinations an element is inserted. Most of the mappings are intuitive, for example, *ulink* is mapped onto **,-u-l**. The mappings are very easy to customise and the improvement in document creation speed is amazing.

I wanted to go through the file and add mappings for every single element, and make the mappings more intuitive since I felt that the previous mappings, on occasion, deviated from intution somewhat. I have not done this yet.

I am not the creator of the file, this information is specified in the file header. I modified some of the mappings from the original and added some new ones, here is the file: dbhelper.vim [files/dbhelper.vim]. It would be nice if somebody created a comprehensive version of this file with mappings for every tag installed, let me know if you do.

Download the file (on Windows) to `where/you/installed/gvim/vimXX/ftplugin/xml`, where XX is the version number with a dot removed, for example 61. On Unix variants, save the file to your home directory and add something like the following to your `.vimrc` file:

```
augroup BEGIN
   au! BufRead,BufNewFile *.xml source ~/dbhelper.vim
augroup END
```

Next time you use gvim to edit an XML file `dbhelper.vim` will be sourced and the mappings enabled. Of course, it may not always be desireable to have such mappings enabled for every XML file but different mappings for different types of XML could be sourced manually from gvim by going into lastline mode and typeing "**source** `fileName`". Or one could use the gvim macro language to write something which scans the XML doctype and loads the appropriate helper file.

The mappings are used in insert mode by typing ',' followed by the mapping chars, typing ",pp" causes the following to be inserted:

```
<para>
</para>
```

Usually the cursor will be appropriately positioned.

7. gVim and Java

7.1. Automatically generate getter and setter methods

7.1.1. Install

1. Download the script from http://www.vim.org/script.php?script_id=490.
2. Place in `ftplugins` directory of gVim installation. If there are already files that begins with *java*, create a directory called `java` in `ftplugins` and put all the files that begin with *java* in that directory.

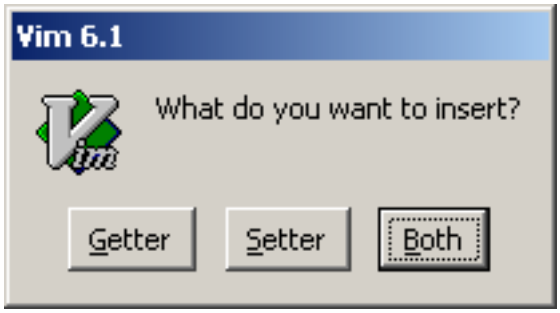
7.1.2. Use

1. Select the variables to create getter/setter methods for.
2. Press \-**p**

The getter and setter methods for the selected variables will be generated. For example, selection of the following:

```
private int age;
```

Followed by executing the keystroke sequence \-**p**, brings up the dialog box shown below:



Clicking on Both generates the two methods shown below:

```
/**
 * Get age.
 *
 * @return age as int.
 */
public int getAge() {
   return age;
}

/**
 * Set age.
 *
 * @param age the value to set.
 */
public void setAge(int age) {
   this.age = age;
}
```

One can modify the templates in the script to produce custom output, for example, the above output shows an indentation of 3 and has the opening bracket of a method on the same line as the method name.

8. References (and links you may find useful)

- http://octopus.cdut.edu.cn/~yf17/unix/unixcd/vi/index.htm

  Online version of *Learning the vi Editor* by Linda Lamb and Arnold Robbins.

- http://www.thomer.com/vi/vi.html

  Vi Lovers Home Page

- http://www.moesworld.com/vim/usr_toc.html

  Online version of the VIM user manual by Bram Moolenaar