Beginner's guide to Vi Improved (vim)

Jesse Goerz

jwgoerz@users.sourceforge.net

1. Introduction

This document will cover the console version of vim from Debian stable. Although much of what is discussed here will also apply to gvim, it may or may not work with gvim.

Learning vim can be a bit of a trial but it is well worth it in the long run. The most important thing you can do is read the section Vim's built in help. Once you figure out how to use vim's built-in help effectively, you'll learn vim much more comfortably.

No introduction to vim would be complete without a warning. Vim is not a difficult editor to learn but it won't be like any other editor you've used. In the beginning you'll find that the movement and editing commands can be quite irritating. However, with a bit of patience and perseverance you will become quite adept at using it.

If it is so irritating why would you want to learn how to use it? Good question. My primary motivation to learn vim was watching someone who was good with vim edit a configuration file. Without moving his hands from the keyboard he cut large blocks of text and moved them around quite rapidly. This simple textual explanation just doesn't do justice to what I witnessed that day. Vim offers you the ability to rapidly search, edit, copy, and "read in" files in far less time then your average editor. Vim can also do things you just can't do in other editors. Once you get good enough, vim saves you a lot of time over conventional editors.

Before we move on to the next section I would like to pass on a couple quick tips.

Beware the caps-lock key!

Because vim uses the keyboard as a primary means of navigation, you need to make sure the caps-lock key is *not* depressed. If you're trying to move around in vim and strange things are happening, check to make sure the caps-lock key is *not* depressed.

Tapping the Escape key twice in quick succession will clear any commands you have begun to type incorrectly.

2. Installing vim

To install vim on Debian simply become the root user and type this at the command line:

```
bash# apt-get install vim vim-rt
```

The package vim contains the executable, and the vim-rt package contains all the files necessary to get syntax highlighting to work.

If you want to install the graphical version of vim install the packages `vim-gtk` and `vim-rt`.

Programming

If you're going to be doing programming, you'll also want to install the package `exuberant-ctags`.

3. Vim's built in help

Learning the way the help system system works in any application you use is always a good idea. In vim, it isn't just a good idea, it's essential. When I first started using vim I put it off for the longest time. I wish I wouldn't have done that!

3.1. Tutor me!

Thanks to vim's hard working developers vim has a very extensive built-in help system. As part of their work they developed a tutor which will walk you through the basics.

To start the tutor simply type this command at the console:

```
bash$ vimtutor
```

Vim should start and look something like this:

```
===============================================================================
=    W e l c o m e   t o   t h e   V I M   T u t o r    -    Version 1.4      =
===============================================================================

Vim is a very powerful editor that has many commands, too many to
explain in a tutor such as this.  This tutor is designed to describe
enough of the commands that you will be able to easily use Vim as
an all-purpose editor.

The approximate time required to complete the tutor is 25-30 minutes,
depending upon how much time is spent with experimentation.

The commands in the lessons will modify the text.  Make a copy of this
file to practice on (if you started "vimtutor" this is already a copy).

It is important to remember that this tutor is set up to teach by
use.  That means that you need to execute the commands to learn them
properly.  If you only read the text, you will forget the commands!

Now, make sure that your Shift-Lock key is NOT depressed and press
the   j   key enough times to move the cursor so that Lesson 1.1
completely fills the screen.
```

Simply follow along with the tutor. Below is the outline of what the tutor covers in case you are already familiar with vim. I recommend you continue using the tutor until you can complete it "out of sequence." In other words, you ought to be able to jump around throughout the tutor, doing sections at random, before continuing with the rest of this document.

1. Moving the cursor

3.2. Navigating vim's help

Vim has an extensive help system. The most basic command is:

:help

This command will take you to vim's help text. Throughout the help files you will see tags like this: |bars|. By positioning your cursor over these tags and hitting **CTRL-]** you can jump directly to the subject. To get back to your original position simply type **CTRL-t**.

You can quit the help window by typing **:q**.

A variation of the above command is this:

:help *subject*

Where *subject* is the information you are looking for. One thing to note about the above command is that it takes you to the *subject* reference. What this means is that this type of help is aimed at users who are already familiar with vim's help syntax. The vimtutor command in the Tutor me! section covers this in Lesson 2 of the vimtutor. You did do the vimtutor didn't you?

You can also access vim's help by pressing the F1 function key.

Vim's help also contains some quick reference links which you can locate by typing:

:help quickref

If you're looking for less terse explanations a good place to start is with the vim introduction. In addition to the introduction you can find more detailed help with this command:

:help doc-file-list

4. Modeful editing

If you haven't noticed by now, vim is a modeful editor. Vim has 6 different *basic* modes of operation. The modes are Normal, Visual, Select, Insert, Command line, and Ex. Each mode is suited to a particular purpose. The following sections will cover the basics of each of these modes.

If you are not sure which mode you are in simply tap the Escape key twice and you will be put into Normal mode.

To get more information on modes type:

```
:help vim-modes
```

4.1. Normal mode

Normal mode is the mode vim normally starts in. Normal mode is also known as command mode. Be careful not to confuse this with Command line mode.

Normal mode is where you use most of the built in commands and is primarily used to move the cursor around.

To get some basic information about normal mode type:

```
:help normal-mode
```

4.2. Visual mode

Visual mode is a nice way to select large blocks of text so that you can perform other operations on them. You enter visual mode by pressing **v** (lowercase v), **V** (capital V), or **CTRL-V**. The **v** command highlights the block of text by character, the **V** command highlights text by the line, and the **CTRL-V** highlights the text by block.

Once you enter visual mode you can use the movement keys (h, k, j, l, w ...) to change which text is highlighted. You can then enter a sequence of Normal commands to manipulate the text. You can also enter Command line mode by typing the : (colon) and then entering a command.

If you enter command line mode from visual mode you'll notice this in the "status bar":

```
:'<,'>
```

This is just vim's notation for selecting that block of text. Don't backspace over it. It will be part of the command you enter.

As an example, open vim and create a new document, or simply use vimtutor and manipulate that file. Type several sentences on several different lines. Press **V** to enter "linewise" visual mode. Use your cursor/movement keys to select the first line and type: **gU**. This should change the entire selected text to all uppercase letters. Once the text is highlighted you can also use the **(y)**ank and **(d)**elete commands to "copy" or "cut" the text so you can paste it elsewhere with the **(p)**aste command. Experiment with it a little. You'll use these commands a lot.

TODO. Need some useful examples of visual mode. Anyone want to coauthor?

To get more information on what's possible type:

```
:help visual-mode
```

4.3. Select mode

TODO. Need some useful examples of select mode. Anyone want to coauthor?

To get more information on what's possible type:

```
:help select-mode
```

4.4. Insert mode

Insert mode is the mode in which the letters you type are input into the document. This mode resembles a regular editor. There are other options which will allow you to do additional things as well. For instance, if you use vim with languages which have accented characters that do not appear on your keyboard you can use the *digraph* option to type those characters.

To use digraphs you need to issue the command:

```
:set digraph
```

If you get an error after using this command your version of vim was not compiled with digraph support. The default version of vim in Debian has digraph support.

Digraphs are used by typing a character, the **Backspace** key, and then another character. These combinations of characters are called maps. You can see which digraph maps are available to you by typing:

```
:digraph
```

Here is the output of that command:

```
~! ¡ 161   c| ¢ 162   $$ £ 163   ox ¤ 164   e= ¤ 164   Y- ¥ 165   || ¦ 166   pa § 167
"" ¨ 168   cO © 169   a- ª 170   << « 171   -, ¬ 172   --   173   rO ® 174   -= ¯ 175
~o ° 176   +- ± 177   22 ² 178   33 ³ 179   ″ ´ 180   ju textmu 181   pp ¶ 182   ~. · 183
„ ¸ 184   11 ¹ 185   o- º 186   >> » 187   14 ¼ 188   12 ½ 189   34 ¾ 190   ~? ¿ 191
```

```
A` À 192   A' Á 193   A^ Â 194   A~ Ã 195   A" Ä 196   A@ Å 197   AA Å 197   AE Æ 198
C, Ç 199   E` È 200   E' É 201   E^ Ê 202   E" Ë 203   I` Ì 204   I' Í 205   I^ Î 206
I" Ï 207   D- Ð 208   N~ Ñ 209   O` Ò 210   O' Ó 211   O^ Ô 212   O~ Õ 213   O" Ö 214
/\ × 215   OE × 215   O/ Ø 216   U` Ù 217   U' Ú 218   U^ Û 219   U" Ü 220   Y' Ý 221
Ip Þ 222   ss ß 223   a` à 224   a' á 225   a^ â 226   a~ ã 227   a" ä 228   a@ å 229
aa å 229   ae æ 230   c, ç 231   e` è 232   e' é 233   e^ ê 234   e" ë 235   i` ì 236
i' í 237   i^ î 238   i" ï 239   d- ð 240   n~ ñ 241   o` ò 242   o' ó 243   o^ ô 244
o~ õ 245   o" ö 246   :- ÷ 247   oe ÷ 247   o/ ø 248   u` ù 249   u' ú 250   u^ û 251
u" ü 252   y' ý 253   ip þ 254
```

The columns are arranged so that the first character is typed followed by a backspace and then the second character is typed. For example, to get the ¡ (an upside down exclamation point) character I would simply type **~-Backspace-!**.

These mappings seem to change between systems and versions of vim. Make sure you double check what your maps are so you get what you expect.

If you use these characters frequently you may wish to explore the *langmap* option as well. For more information type the command **:help** langmap.

To get more information on what's possible with digraphs type:

:help digraphs

To get more information on what's possible with insert mode type:

:help insert-mode

4.5. Command line mode

Command line mode is entered by typing **:** (the colon). Once typed the "status bar" along the bottom of the console shows the colon and whatever you type. All the previous **:help** commands you have typed are examples.

Command line mode is entered from Normal mode. If you type the colon and can’t seem to get into command line mode try tapping the escape key twice and then type the colon.

Command line mode contains a "history" much like the shell does. By typing the beginning of your command and then using the arrow keys you can scroll through a list of the previous commands you entered.

Command line mode also has "tab-completion" much like the shell does. By typing the beginning of your command and then the **TAB** key the command line will complete the available commands for you.

To get more information on what’s possible with command line mode type:

```
:help cmdline-mode
```

4.6. Ex mode

TODO. Anyone want to coauthor?

TODO. Need some useful examples of ex mode.

5. Common commands

The most common commands used in vim are covered in the Tutor me! section. You did the tutor right? The following sections contain some additional commands.

5.1. Movement commands

Movement commands

G

Move to the end of the file.

gg

Move to the beginning of the file.

CTRL-b

Move back a page.

CTRL-f

Move forward a page.

CTRL-u

Scroll up.

CTRL-d

Scroll down.

If you don't like the way the scrolling works you can adjust it with the scroll, scrolloff, and scrolljump options. Type **:help** scroll for more information. The vimrc files section explains how to make these adjustments permanent.

CTRL-e

Expose another line at the top. Useful for scrolling the screen without moving your cursor from its position.

CTRL-y

Expose another line at the bottom. Useful for scrolling the screen without moving your cursor from its position.

}

Move forward a paragraph. Useful for moving through documents which have blank lines separating paragraphs or blocks. An easy way to distinguish between this and { is to remember it points in the direction you wish to go.

{

Move backward a paragraph. Useful for moving through documents which have blank lines separating paragraphs or blocks.

)

Move forward a sentence. An easy way to distinguish between this and (is to remember it points in the direction you wish to go

(

Move backward a sentence.

:number

Where *number* is a decimal number indicating the line number you wish to jump/move to.

'0

Vim creates a "bookmark" each time you exit Vim. The last "bookmark" is **'0**. So if you have a file you haven't edited in a long time and can't remember where you left off simply type this command

and it will take you to the place you last edited. You can find out what marks are set by typing **:marks**.

Reference **:help** navigation and **:help** motion for more information on motion commands.

5.2. Search commands

Search commands

/text

Search forward in the document for *text*. If found, vim will move the cursor to that position in the file. Once **/** is typed, *text* will show up in the status bar as you type it.

?text

Search backward in the document for *text*. If found, vim will move the cursor to that position in the file. Once **?** is typed, *text* will show up in the status bar as you type it.

Normally the search commands stop at the end or beginning of the file depending on which direction you are searching. You can force the search to "wrap around" the file using the wrapscan option. Type **:help** wrapscan for more information.

*****

This is a search command which will find the next occurance (forward) of the word the cursor is currently on.

n

Find the *(n)*ext occurance of the previous search. It will use the same direction (i.e. forward vs. backwards) of the previous search as well.

There are several options which allow you to enhance your searching. The hlsearch option highlights the text of the search. The incsearch option highlights and shows possible matches as you type the text. You can get more information on these options by typing **:help** hlsearch and **:help** incsearch. The vimrc files section explains how to make these adjustments permanent.

5.3. Simple editing/combination commands

Deleting/selecting parts of a line

dfx

Delete forward until you find *x* (inclusive). If *x* is found, vim will delete everything from the current cursor position to and including *x*.

This one and its variations **dFx**, **dtx**, and **dTx** are very useful when editing mark-up languages like html and sgml. It is especially useful in deleting and changing quoted attributes within mark-up tags.

This is a combination of the (d)elete command and the (f)ind motion command. You may also substitute other commands for the **d**elete portion of that command, such as **v**isual by character, or **y**ank.

dFx

Delete backward until you find *x* (inclusive). If *x* is found, vim will delete everything from the current cursor position to and including *x*.

dtx

Delete forward until you find *x* (exclusive). If *x* is found, vim will delete everything from the current cursor position to, but not including *x*.

dTx

Delete backward until you find *x* (exclusive). If *x* is found, vim will delete everything from the current cursor position to, but not including *x*.

Deleting/selecting spanning multiple lines

d/x

Delete forward until you find *x* (exclusive). If *x* is found, vim will delete everything from the current cursor position to, but not including *x*, spanning multiple lines if necessary.

x doesn't have to be one character. It's most likely to be a word or several characters. If you're editing your email, you could use this to delete all text up to your signature by executing **d/--**.

This is a combination of the (d)elete command and the (/) search forward motion command. You may also substitute other commands for the **d**elete portion of that command, such as **v**isual by character, **V**isual by line, or **y**ank.

d?x

Delete backward until you find *x* (exclusive). If *x* is found, vim will delete everything from the current cursor position to, but not including *x*, spanning multiple lines if necessary.

Cut, copy, and paste

V-any_motion_command

V selects the text by the line. Simply use **j**, **k**, or the paragraph motion commands (**{**, **}**) to select the block of text. Now type **d** to delete (cut) it, **y** to yank (copy) it, and **p** to paste it.

Spell checking

TODO

TODO.

Find and replace

This topic is covered extremely well by Sven Guckes. You can read the document here: http://www.math.fu-berlin.de/~guckes/vi/subst.html (http://www.math.fu-berlin.de/~guckes/vi/subst.html)

Do you have a simple editing option you use frequently? Please send it to us at newbiedoc-discuss@lists.sourceforge.net (mailto:newbiedoc-discuss@lists.sourceforge.net) so we can add it to this document.

6. Personalizing vim

6.1. vimrc files

Vim is a very customizable editor. Many of the options you can set require you to type in a command like this:

```
:set autoindent
```

This is the option to set autoindent which keeps the indentation level you are currently at when you hit the enter key. Rather than retype this command every time you want this option enabled, vim has a resource file it checks to see which options you would like set when vim starts. Technically, there are two such files, /etc/vimrc, and ~/.vimrc. Within these files you can set standard options which will automatically be set when you start vim.

The /etc/vimrc is the system wide configuration file and shouldn't really be edited unless you know what you are doing or you're the administrator for that machine. The ~/.vimrc file is located in your home directory (that's what the ~/ means, in case you were wondering) and is meant specifically for the purpose of customization.

Before editing a vimrc file it's a good idea to back it up just in case you mess it up beyond repair.

You can see a list of available options that can be set by typing the following command:

:set all

The output of that command looks something like this:

```
autoindent        noincsearch           scroll=18           textwidth=0
noautowrite         noinfercase         noscrollbind        notildeop
background=dark   noinsertmode          scrolljump=1        timeout
backspace=2         isprint=@,161-255   scrolloff=0         timeoutlen=1000
nobackup              joinspaces        nosecure            notitle
backupext=~         key=                selectmode=         titlelen=85
nobinary              keymodel=           shell=/bin/bash     titlestring=
nocindent             keywordprg=man      shellcmdflag=-c   nottimeout
cinoptions=         langmap=            shellquote=         ttimeoutlen=-1
cmdheight=1         laststatus=1        shellxquote=        ttybuiltin
columns=98        nolazyredraw        noshiftround          ttyfast
nocompatible        nolinebreak           shiftwidth=4        ttymouse=xterm
noconfirm             lines=38          noshortname           ttyscroll=999
cpoptions=aABceFs nolisp                showbreak=          ttytype=xterm
dictionary=       nolist              noshowcmd             undolevels=1000
nodigraph             listchars=eol:$   noshowfulltag         updatecount=200
display=            magic               showmatch           updatetime=4000
noedcompatible        makeprg=make        showmode            verbose=0
endofline           matchtime=5         sidescroll=0        viminfo='20,"50
equalalways         maxfuncdepth=100  nosmartcase           visualbell
equalprg=           maxmapdepth=1000  nosmartindent         warn
noerrorbells          maxmem=5120       nosmarttab          noweirdinvert
noexpandtab           modelines=5         startofline         wildcharm=^@
noexrc              nomodified            statusline=         wildignore=
fileencoding=ansi   more                swapfile          nowildmenu
fileformat=unix     mouse=              swapsync=fsync      wildmode=full
filetype=           mousemodel=extend   switchbuf=          winheight=1
formatoptions=tcq   mousetime=500       syntax=             winminheight=1
formatprg=        nonumber              tabstop=4           wrap
nogdefault          nopaste               tagbsearch          wrapmargin=0
grepprg=grep -n     pastetoggle=        taglength=0         wrapscan
helpheight=20       patchmode=          tagrelative         write
```

As you can see there are a lot of options to set/unset. To get more information on what an option does simply type **:help name_of_option** replacing name_of_option with the option you're interested in.

Once you decide how you want an option to be set you can add it to your ~/.vimrc file. Using the autoindent example above, simply add this line to your vimrc file:

```
set autoindent "this sets the autoindent option
" set noautoindent
```

Comments are created in vimrc files using the " (quote) character. As you can see I put a comment next to the autoindent option and also added a "commented out" option which disables the previous noautoindent option I had set. Most of the options follow this convention; if you wish to disable it, simply put no in front of the option.

For more information about vimrc files type:

```
:help vimrc
```

6.2. Color Schemes

TODO. Need some useful examples of color schemes. Anyone want to coauthor?

For more information about color schemes type:

```
:help
```

6.3. Syntax highlighting

If you installed the vim-rt package described in Installing vim, all you have to do to enable syntax highlighting is add **syntax on** to your .vimrc file. The highlighting syntax is chosen based on filename extension so it happens pretty much automatically.

You can change the default colors of a syntax highlight by using the **highlight** command. For example, to change all comments to the color green type:

```
:highlight Comment ctermfg=green
```

The second argument to that command is the group name. To make the setting permanent simply add it to your .vimrc.

In order for the color change to work the **highlight** command should come after the **syntax on** command in your .vimrc.

To see what group names are available to be changed type:

```
:help group-name
```

The available colors can be found by typing:

```
:help cterm-colors
```

Some colors may not be available. Reference **:help** xterm-colors and **:help** xiterm for more information.

TODO. Need some useful examples of customized syntax. Anyone want to coauthor?

For more information about syntax highlighting type:

```
:help syntax
```

6.4. Abbreviations

Abbreviations are just shortcuts you type which expand to longer versions of themselves. You can create an abbreviation simply by typing:

```
:ab dmv Department of Motor Vehicles
```

Abbreviations take the form:

```
:ab abbreviation Full text, spaces are ok
```

What happens is when you type the abbreviation in vim, as soon as you hit the space bar it is expanded to the full text. The abbreviation dmv in the prior example would expand to "Department of Motor Vehicles." If you close vim these settings are lost unless you commit them to your .vimrc file.

Unless you are only using the abbreviation on a small document, abbreviations are usually added to your personal .vimrc file. This is accomplished by adding the abbreviation like this:

```
ab abbreviation Everything after abbreviation will be input
```

To find out which abbreviations you currently have defined, just type **:ab** and hit enter and the abbreviation definition will show up in the "status bar."

For more information about abbreviations type:

```
:help ab
```

7. Mapping Keys

Mapping keys is one of the most useful and productive operations that vim performs. Mapping keys is just a method to map a complex command to a particular keystroke or sequence of keystrokes. If you find yourself routinely performing the same commands to edit a document there is a good possibility you could create a map to make your editing easier.

For instance, if you write html, xml, or sgml documents in a text editor you know it can become quite tedious typing out all those tags all the time. Markup language tags may also include attributes which you also have to type in or cut and paste from other locations. This can slow your work down quite a bit. There are gui programs which sort of solve this but many of them can't always give you the control you need and you end up resorting to hand editing the file anyway. The simple solution in vim is to create a map.

There are several different types of maps you can create but the two we will concentrate on are Normal mode and Insert mode maps. Normal mode maps begin with the command **map** and look something like this:

```
map keyboard_key command_to_execute
```

Insert mode maps begin with the command **map!** and look something like this:

```
map! keyboard_key command_to_execute
```

Avoid re-mapping the <F1> key as this is always mapped to vim's help.

The complexity of the map is pretty much limited to your imagination. Here is a simple map you might use to open a letter:

```
map <F2> iTo whom it may concern, <CR><CR><TAB>
```

This is a Normal mode map set to use the <F2> key. It won't work when you're in insert mode. When you press the <F2> key, vim executes the **i**nsert command (because it's still in Normal mode), it then types all the letters as it sees them (because now it's in Insert mode) until it hits <CR>. This is how you type a Carriage Return in maps. So vim executes a carriage return. It then executes the next carriage return and then indents that line with one tab. At this point the map is finished. If you would like to test this map go ahead and copy the map into vim by typing **:** and then pasting the map into the status bar. Press Return. Now try pressing the <F2> key and see what it does. One thing to notice is the map exits while in insert mode. This allows you to start typing immediately.

If you forgot what maps you have or just want to see what maps you have defined you can simply type: **:map** or **:map!** and hit enter. The defined maps will show up in the "status bar."

To make maps permanent add them to your .vimrc file.

Earlier I talked about how maps can make working with markup languages easier. Here are several example Insert mode maps which can be used to create headings in html:

```
map! ,h1 <H1></H1><ESC>2ba
map! ,h2 <H2></H2><ESC>2ba
map! ,h3 <H3></H3><ESC>2ba
```

We will focus on the first map. This map is executed while you are typing in insert mode. After you type **,h1** vim types <H1></H1> for you. It then sees the escape command which just like any other time you're in insert mode takes you back to Normal mode. Vim then sees the command to move back 2 words (**2b**) which puts the cursor on the ">" of the first <H1> tag. Vim then sees the **a** command so it moves one space to the right and enters insert mode. At this point the map is finished, your tags have been written, you are in insert mode and ready to type the heading. The beauty of maps like this is you never leave insert mode (at least from your perspective) and you type only 3 characters and get many more. Once you get used to your personal maps it's easy to forget about all the tags and concentrate more on your content.

If you would like to try these maps out simply copy and paste them into vim like the previous example.

There are several things you should notice about the previous examples. Insert mode maps are generally used to enter portions of text which are repetitive or time consuming. They work extremely well for inserting markup text and then positioning your cursor where you would logically type next. They take care of "closing" tags for you which can cut down on nesting errors and ill formed documents.

When creating maps, selection of your assigned keystrokes is pretty much up to you. I would recommend you avoid re-mapping standard commands as this makes it difficult for you to transition to someone else's machine unless you carry around a copy of your .vimrc on a floppy (which I've often thought about :)). When you choose a sequence of commands for Insert mode choose a sequence which is unlikely to be typed. In the previous example I chose the "," because in almost every type of document it is always followed by a space. The reason for this is vim has to pause for a second whenever the first character of one of its maps is pressed. It does this to see if you are typing a map or just a sequence of characters that just happens to start with the mapped character. By choosing the comma, every time I type a comma and then the following space, vim knows almost immediately that I don't want a map.

For more complex map examples see the section Example maps and vimrc files.

For more information about mapping keys type:

```
:help key-mapping
```

8. Writing Functions

TODO. Need some useful examples. Anyone want to coauthor?

For more information about built-in functions type:

```
:help functions
```

For more information about user defined functions type:

```
:help user-functions
```

9. Programming with vim

This subject is covered very well by a document at the Linux Documentation Project. Please reference the C-editing-with-VIM-HOWTO (http://www.linuxdoc.org/HOWTO/C-editing-with-VIM-HOWTO/index.html).

10. Example maps and vimrc files

Following is my "system" using a combination of vimrc files and maps.

What I did was create a basic .vimrc file in my home directory and map the function keys (except F1) to source each of the custom vimrc files. Here is my example ~/.vimrc.

The custom vimrc files were placed in the ~/.vim_custom directory. Currently those files consist of vimrc files for editing shell scripts, php scripts, C/C++ programming, and sgml source files.

- Here is my example ~/.vim_custom/bash-vimrc.
- Here is my example ~/.vim_custom/php-vimrc.
- Here is my example ~/.vim_custom/c-vimrc.
- Here is my example ~/.vim_custom/sgml-vimrc.

If you're going to copy and paste these into files please beware of the **CTRL**-**D** character. That character has to be typed into your file with this key combination: **CTRL**-**V** and then **CTRL**-**D**. It will display in the file like this: ^D. A quick way to convert all these is to do this:

```
:%s/\^D/CTRL-V CTRL-D/g
```

Everything in that command is typed as printed on this document except **CTRL-V CTRL-D** which is typed as explained above.

Each vimrc file calls a function which unmaps the function keys except for the F2 function key which is always mapped to re-source the base ~/.vimrc file. It then maps the function keys and any other custom maps specific to its purpose.

I chose to do it this way for several reasons. The first reason was modularity. If I find someone else's vimrc file which includes some neat functions or mappings it's easy to incorporate them into my existing "system" without messing with what's already there. Second, I don't use the function keys for anything else so it seemed the logical thing to do. Third, it seemed to follow a sort of "menu" approach to doing things.

If you have an approach to doing this which you like and would like to include it in this document please send in your vimrc/functions/mappings to newbiedoc-discuss@lists.sourceforge.net (mailto:newbiedoc-discuss@lists.sourceforge.net) so we can add it to this document.

11. References and links

Vi Lovers home page (http://www.thomer.com/thomer/vi/vi.html)

http://www.vim.org

C-editing-with-VIM-HOWTO (http://www.linuxdoc.org/HOWTO/C-editing-with-VIM-HOWTO/index.html)

Vi Substitute Guide (HOWTO use the ":s" command) (http://www.math.fu-berlin.de/~guckes/vi/subst.html)

12. Contributors

These folks have contributed in one way or another, I've tried to include what is was but chances are I screwed it up. If I did, my apologies, and thanks!

- Allan M. Wind (d/x and d?x)
- Romain Lerallut (*)

A. My examples

Example A-1. My ~/.vimrc

```
" Global variable which holds the path to my customization files
```

```
" You'll need to edit this so it matches your system
let g:VIM_CUSTOM = "/home/jesse/.vim_custom/"

set visualbell
set background=dark
set tabstop=4
set showmatch
set showcmd
set autowrite

""""""""""""""""""""""""""""""""""""""""""""""""""
"                 Color scheme                  "
""""""""""""""""""""""""""""""""""""""""""""""""""

" Some custom color modifications.  reference :help highlight and :help cterm
highlight ModeMsg cterm=bold ctermfg=2 ctermbg=black " set mode message ( --INSERT-- ) to gr
highlight StatusLine ctermfg=7 ctermbg=9 " set the active statusline to black on white
highlight StatusLineNC ctermfg=8 ctermbg=9 " set inactive statusline to black on grey
syntax on

""""""""""""""""""""""""""""""""""""""""""""""""""
"              Function Key maps                 "
""""""""""""""""""""""""""""""""""""""""""""""""""

" Re-source the default vimrc
map <F2> :execute Clean_up()<CR> :source $HOME/.vimrc<CR>

" C/C++ Programming
map <F3> :execute Clean_up()<CR> :execute Re_source("c-vimrc")<CR>

" shell programming
map <F4> :execute Clean_up()<CR> :execute Re_source("bash-vimrc")<CR>

" php programming
map <F5> :execute Clean_up()<CR> :execute Re_source("php-vimrc")<CR>

" sgml editing
map <F6> :execute Clean_up()<CR> :execute Re_source("sgml-vimrc")<CR>

" Once you invoke this you need to delete rows and type in the # you wish
" to process
map <F11> :execute Dump_extra_whitespace(rows)

" Reverse the background color
map <F12> :execute ReverseBackground()<CR>

""""""""""""""""""""""""""""""""""""""""""""""""""
"              Custom functions                  "
""""""""""""""""""""""""""""""""""""""""""""""""""

" Re-source the rc files
:function! Re_source(file_name)
: let path_file_name = g:VIM_CUSTOM . a:file_name
```

```
:  if filereadable(path_file_name)
:   execute 'source ' . path_file_name
:   echo path_file_name . " Loaded sucessfully"
:  else
:   echo path_file_name . " does NOT exist"
:   return 0
:  endif
:endfunction

" This function allows me to quickly remove extra tabs and whitespace
" from the beginning of lines.  This seems to be a problem when I cut
" and paste or when people don't use resizeable tabs.
" TODO The only problem with this is after you execute it it jumps to the
" beginning of the file.  I need to figure out how to fix that.
:function! Dump_extra_whitespace(rows)
: let com = ".,+" . a:rows . "s/^[  ]*//g"
: execute com
:endfunction

" This function was created by Dillon Jones (much better than my first attempt)
" it reverses the background color for switching between vim/gvim which have
" different defaults.
" TODO The only problem with this is after you execute it it jumps to the
" beginning of the file.  I need to figure out how to fix that.
:function! ReverseBackground()
: let Mysyn=&syntax
: if &bg=="light"
: se bg=dark
: else
: se bg=light
: endif
: syn on
: exe "set syntax=" . Mysyn
":   echo "now syntax is "&syntax
:endfunction


" Cleanup
:function! Clean_up()
:set visualbell&
:set background&
:set tabstop&
:set showmatch&
:set showcmd&
:set autowrite&
:endfunction
```

Example A-2. My ~/.vim_custom/bash-vimrc

```
:set visualbell " Silence the bell, use a flash instead
:set formatoptions=tcqor " t=text, c=comments, q=format with gq command, o,r=autoinsert comm
:set shiftwidth=4 " set shiftwidth to 4 spaces
:set tabstop=4 " set tab to 4 spaces
:set showmatch " Show matching brackets/braces/parantheses.
:set background=dark  " set background to dark
:set showcmd " Show (partial) command in status line.
:set autowrite " Automatically save before commands like :next and :make
:set textwidth=98 " My terminal is 98 characters wide
:set ai " set autoindent
:syntax on
"
" Shell Programming
:map <F2> :execute Clean_up()<CR> :source $HOME/.vimrc<CR>
:map <F3> gg:execute License_notice("bash_gpl_notice")<CR>dd18jO<ESC>
:map <F4> ofor var in $list<ESC>odo<ESC>odone<ESC>koecho $var<ESC>I
:map! <F4> for var in $list<ESC>odo<ESC>odone<ESC>koecho $var<ESC>I
:map <F5> ofor((i=0;; i++)); <ESC>odo<ESC>odone<ESC>2kI<ESC>f;a<SPACE>
:map! <F5> for((i=0;; i++)); <ESC>odo<ESC>odone<ESC>2kI<ESC>f;a<SPACE>
:map <F6> owhile ;<ESC>odo<ESC>odone<ESC>2kI<ESC>f;i
:map! <F6> while ;<ESC>odo<ESC>odone<ESC>2kI<ESC>f;i
:map <F7> oif [ ]; then<ESC>ofi<ESC>kf[a<SPACE>
:map! <F7> if [ ]; then<ESC>ofi<ESC>kf[a<SPACE>
:map <F8> ocase $list in<CR>);;<CR>);;<CR>*);;<CR>^Desac<ESC>3kf)i
:map! <F8> case $list in<CR>);;<CR>);;<CR>*);;<CR>^Desac<ESC>3kf)i
:map <F9> oPS3='Prompt: '<ESC>oselect choice in $list<ESC>odo<ESC>oecho "Here's your choice:
:map! <F9> PS3='Prompt: '<ESC>oselect choice in $list<ESC>odo<ESC>oecho "Here's your choice:


:function! License_notice(file_name)
: let path_file_name = g:VIM_CUSTOM . a:file_name
: execute 'r ' . path_file_name
:endfunction

" Cleanup
:function! Clean_up()
:set visualbell&
:set formatoptions&
:set shiftwidth&
:set tabstop&
:set showmatch&
:set background&
:set showcmd&
:set autowrite&
:set textwidth&
:set ai&
:unmap <F3>
:unmap! <F3>
:unmap <F4>
:unmap! <F4>
:unmap <F5>
```

```
:unmap! <F5>
:unmap <F6>
:unmap! <F6>
:unmap <F7>
:unmap! <F7>
:unmap <F8>
:unmap! <F8>
:unmap <F9>
:unmap! <F9>
:endfunction
```

Example A-3. My ~/.vim_custom/php-vimrc

```
set visualbell " Silence the bell, use a flash instead
set cinoptions=:.5s,>1s,p0,t0,(0,g2 " :.5s = indent case statements 1/2 shiftwidth
" >1s = indent 1 shiftwidth
" p0 = indent function definitions 0 spaces
" t0 = indent function return type 0 spaces
" (0 = indent from unclosed parantheses
" g2 = indent C++ scope resolution 2 spaces

set cinwords=if,else,while,do,for,switch,case " Which keywords should indent

set formatoptions=tcqor " t=text, c=comments, q=format with "gq", o,r=autoinsert comment lea
set cindent " indent on cinwords
set shiftwidth=4 " set shiftwidth to 4 spaces
set tabstop=4 " set tab to 4 spaces
set showmatch " Show matching brackets/braces/parantheses.
set background=dark  " set background to dark
set showcmd " Show (partial) command in status line.
set autowrite " Automatically save before commands like :next and :make
set textwidth=98 " My terminal is 98 characters wide
syntax on

" PHP Programming
map <F2> :execute Clean_up()<CR> :source $HOME/.vimrc<CR>
map <F3> gg:execute License_notice("c_gpl_notice")<CR>dd18jO<ESC>
map <F4> ofor($i;; $i++) {<ESC>o}<ESC>kI<ESC>f;a<SPACE>
map! <F4> for($i;; $i++) {<ESC>o}<ESC>kI<ESC>f;a<SPACE>
map <F5> oforeach($array as $index=>$value) {<ESC>o}<ESC>kI<ESC>f$l
map! <F5> foreach($array as $index=>$value) {<ESC>o}<ESC>kI<ESC>f$l
map <F6> owhile() {<ESC>o}<ESC>kI<ESC>f(a
map! <F6> while() {<ESC>o}<ESC>kI<ESC>f(a
map <F7> oif() {<ESC>o}<ESC>kf(a
map! <F7> if() {<ESC>o}<ESC>kf(a
map <F8> oswitch() {<ESC>ocase:<CR>break;<CR>case:<CR>break;<CR>default:<CR>}<ESC>6kf(a
map! <F8> switch() {<ESC>ocase:<CR>break;<CR>case:<CR>break;<CR>default:<CR>}<ESC>6kf(a


:function! License_notice(file_name)
: let path_file_name = g:VIM_CUSTOM . a:file_name
: :execute 'r ' . path_file_name
```

```
:endfunction

" Cleanup
:function! Clean_up()
: set visualbell&
: set cinoptions&
: set cinwords&
: set formatoptions&
: set cindent&
: set shiftwidth&
: set tabstop&
: set showmatch&
: set background&
: set showcmd&
: set autowrite&
: set textwidth&
: unmap <F3>
: unmap! <F3>
: unmap <F4>
: unmap! <F4>
: unmap <F5>
: unmap! <F5>
: unmap <F6>
: unmap! <F6>
: unmap <F7>
: unmap! <F7>
: unmap <F8>
: unmap! <F8>
:endfunction
```

Example A-4. My ~/.vim_custom/c-vimrc

```
set visualbell " Silence the bell, use a flash instead
set cinoptions=:.5s,>1s,p0,t0,(0,g2 " :.5s = indent case statements 1/2 shiftwidth
" >1s = indent 1 shiftwidth
" p0 = indent function definitions 0 spaces
" t0 = indent function return type 0 spaces
" (0 = indent from unclosed parantheses
" g2 = indent C++ scope resolution 2 spaces

set cinwords=if,else,while,do,for,switch,case " Which keywords should indent

set formatoptions=tcqor " t=text, c=comments, q=format with "gq", o,r=autoinsert comment lea
set cindent " indent on cinwords
set shiftwidth=4 " set shiftwidth to 4 spaces
set tabstop=4 " set tab to 4 spaces
set showmatch " Show matching brackets/braces/parantheses.
set background=dark  " set background to dark
set showcmd " Show (partial) command in status line.
set autowrite " Automatically save before commands like :next and :make
set textwidth=98 " My terminal is 98 characters wide
syntax on
```

```
" C Programming
map <F2> :execute Clean_up()<CR> :source $HOME/.vimrc<CR>
map <F3> gg:execute License_notice("c_gpl_notice")<CR>dd18jO<ESC>
map <F4> ofor(i=0;; i++) {<ESC>o}<ESC>kI<ESC>f;a<SPACE>
map! <F4> for(i;; i++) {<ESC>o}<ESC>kI<ESC>f;a<SPACE>
map <F5> owhile() {<ESC>o}<ESC>kI<ESC>f(a
map! <F5> while() {<ESC>o}<ESC>kI<ESC>f(a
map <F6> oif() {<ESC>o}<ESC>kf(a
map! <F6> if() {<ESC>o}<ESC>kf(a
map <F7> oswitch() {<ESC>ocase:<CR>break;<CR>case:<CR>break;<CR>default:<CR>}<ESC>6kf(a
map! <F7> switch() {<ESC>ocase:<CR>break;<CR>case:<CR>break;<CR>default:<CR>}<ESC>6kf(a


:function! License_notice(file_name)
: let path_file_name = g:VIM_CUSTOM . a:file_name
:  :execute 'r ' . path_file_name
:endfunction

" Cleanup
function! Clean_up()
set visualbell&
set cinoptions&
set cinwords&
set formatoptions&
set cindent&
set shiftwidth&
set tabstop&
set showmatch&
set background&
set showcmd&
set autowrite&
set textwidth&
unmap <F3>
unmap <F4>
unmap <F5>
unmap <F6>
unmap <F7>
endfunction
```

Example A-5. My ~/.vim_custom/sgml-vimrc

```
" These mappings are primarily based on Docbook 3.1 which is probably pretty old
" now.  If you upgrade this to a newer version please pass me what you have and
" what version you based it on.  Maybe we can set up version specific maps.

set ai " set autoindent
set visualbell
set shiftwidth=4 " set shiftwidth to 4 spaces
set tabstop=4 " set tab to 4 spaces
set showmatch " Show matching brackets/braces/parantheses.
set background=dark  " set background to dark
```

```
set showcmd " Show (partial) command in status line.
set autowrite " Automatically save before commands like :next and :make
set showmatch " Show matching brackets

"syntax keyword sgmlTODO contained TODO FIXME NEED
"highlight TODO ctermfg=0 ctermbg=3
syntax on

" Re-source the default vimrc
map <F2> :execute Clean_up()<CR> :source $HOME/.vimrc<CR>
map <F3> gg:execute License_notice("sgml_gfdl_comment_notice")<CR>dd13jO<ESC>

" TODO This map works but after it reads it in it takes you to the beginning of the
" file.  Need to fix this.
map <F4> :execute License_notice("sgml_gfdl_notice")<CR>

" Insert a new section header
map! ,h1 <sect1 id="" xreflabel=""><CR><title></title><CR><CR></sect1><ESC>3kf"a
map! ,h2 <sect2 id="" xreflabel=""><CR><title></title><CR><CR></sect2><ESC>3kf"a
map! ,h3 <sect3 id="" xreflabel=""><CR><title></title><CR><CR></sect3><ESC>3kf"a
map! ,h4 <sect4 id="" xreflabel=""><CR><title></title><CR><CR></sect4><ESC>3kf"a
map! ,h5 <sect5 id="" xreflabel=""><CR><title></title><CR><CR></sect5><ESC>3kf"a

" paragraphs
map! ,p <para><CR></para><ESC>O
map! ,P <formalpara><title></title><CR><TAB><para><CR></para><CR>^D</formalpara><ESC>3k$2ba

" Moves to end of next tag (this keeps you in insert mode and typing!)
" THIS IS BY FAR the most useful map, Thanks to the authors of "Learning Vi"
" from O'Reilly!
map! ,e <ESC>f>a
map ,e f>


" markup affecting words
map! ,b <emphasis></emphasis><ESC>2ba
map! ,B <emphasis role="bold"></emphasis><ESC>2bla
map! ,f <filename></filename><ESC>2ba
map! ,u <ulink url=""></ulink><ESC>2bla
map! ,le <link linkend=""></link><ESC>2bla
map! ,x <xref linkend="<ESC>a
map! ,gt <glossterm></glossterm><ESC>2ba
map! ,Gt <glossterm baseform=""></glossterm><ESC>2bla
map! ,r <citation></citation><ESC>2ba
map! ,k <keycombo><keycap></keycap><keycap></keycap></keycombo><ESC>8ba


" regular list and then numbered list (arabic)
map! ,li <itemizedlist><CR><TAB><listitem><CR><TAB><para><CR></para><CR>^D</listitem><CR><C
map! ,ln <orderedlist numeration="Arabic"><CR><TAB><listitem><CR><TAB><para><CR></para><CR>
map! ,lt <listitem><CR><TAB><para><CR></para><CR>^D</listitem><ESC>kO

" Admonitions
```

```
map! ,n <note><CR><TAB><para><CR></para><CR>^D</note><ESC>kO
map! ,N <note><title></title><CR><TAB><para><CR></para><CR>^D</note><ESC>3k$2ba

map! ,t <tip><CR><TAB><para><CR></para><CR>^D</tip><ESC>kO
map! ,T <tip><title></title><CR><TAB><para><CR></para><CR>^D</tip><ESC>3k$2ba

map! ,i <important><CR><TAB><para><CR></para><CR>^D</important><ESC>kO
map! ,I <important><title></title><CR><TAB><para><CR></para><CR>^D</important><ESC>3k$2ba

map! ,c <caution><CR><TAB><para><CR></para><CR>^D</caution><ESC>kO
map! ,C <caution><title></title><CR><TAB><para><CR></para><CR>^D</caution><ESC>3k$2ba

map! ,w <warning><CR><TAB><para><CR></para><CR>^D</warning><ESC>kO
map! ,W <warning><title></title><CR><TAB><para><CR></para><CR>^D</warning><ESC>3k$2ba



" For quick console command examples
map! ,s <ESC>I<screen><CR><prompt>bash$</prompt> <command></command><CR></screen><ESC>k$2ba

" This one is for when commands contain characters that the
" parser wants to parse.
map! ,S <ESC>I<screen><![ CDATA [ <CR>]]<CR></screen><ESC>kO

" TODO test the examp to make sure it works correctly
map! ,E <ESC>o<example id=""><title></title><CR></example><ESC>k^f"a
map! ,q <ESC>I<programlisting><![ CDATA [ <CR>]]><CR></programlisting><ESC>kO
map! ,Q <ESC>I<programlisting><![ CDATA [ <CR>]]><CR></programlisting><ESC>kO

" appendix
map! ,a <appendix xreflabel=""><title></title><CR></appendix><ESC>kf"a

" For inserting graphics
map! ,gr <mediaobject><CR><TAB><imageobject><CR><TAB><imagedata fileref="" format="gif"><CR>

:function! License_notice(file_name)
: let path_file_name = g:VIM_CUSTOM . a:file_name
:  execute 'r ' . path_file_name
:endfunction

" Cleanup
:function! Clean_up()
:set visualbell&
:set background&
:set tabstop&
:set showmatch&
:set autowrite&
:unmap <F3>
:unmap <F4>
:unmap! ,h1
:unmap! ,h2
:unmap! ,h3
:unmap! ,h4
```

```
:unmap! ,h5
:unmap! ,p
:unmap! ,P
:unmap! ,e
:unmap ,e
:unmap! ,b
:unmap! ,B
:unmap! ,k
:unmap! ,f
:unmap! ,u
:unmap! ,le
:unmap! ,x
:unmap! ,gt
:unmap! ,Gt
:unmap! ,r
:unmap! ,li
:unmap! ,ln
:unmap! ,lt
:unmap! ,n
:unmap! ,N
:unmap! ,t
:unmap! ,T
:unmap! ,i
:unmap! ,I
:unmap! ,c
:unmap! ,C
:unmap! ,w
:unmap! ,W
:unmap! ,s
:unmap! ,S
:unmap! ,E
:unmap! ,q
:unmap! ,Q
:unmap! ,gr
:endfunction
```