Vi 201

By Vikram Vaswani

Table of Contents

Introduction

Oh, my...

I must say, I didn't expect quite such a large turnout for this class − it's been years since I had such an attentive audience for one of my little discourses. And so many unfamiliar faces, too − obviously, you've heard the stories of my rapier wit and overpowering charm, and decided to drop in to see the legend in action for yourself.

Anyway, for those of you who don't know me, let me introduce myself. I am Professor Elias Flootburger, newly appointed Head of this university's Computer Science department, and easily one of the most interesting men on campus. I speak seven languages, run four miles a day, and am currently worth fourteen million dollars, thanks to a very intelligent investment I made in a little biotech company a few years ago. As you might imagine, I rarely have trouble finding a date on Saturday night.

Now that we've established my credentials to teach this class, let's get down to work. If you remember, in last time's class, I compared the vi text editor to a Ferrari, and showed you a few basic tricks designed to get you from point A to point B. But no one in their right mind uses a Ferrari just to go shopping for groceries − and so, in today's class, we're going to throw away the rule book, push the pedal to the floor, and indulge in some really dangerous driving.

God, I'm really good with analogies...

Liar, Liar...

The last time we met, I told you that vi operates in either one of two modes − "command mode" and "insert mode".

Well, I lied.

Don't look so horrified − I hate to burst your bubble, but people lie all the time. And just so you know − Santa Claus doesn't exist, and the tooth fairy decided to get out of the business a few years ago; she's now in Las Vegas earning millions as a showgirl...

Anyway, getting back to vi − the editor also comes with a very powerful "visual mode", which allows you to select blocks of text and perform actions − cutting, copying, deleting − on them, in a manner similar to Windows−based editors like Microsoft Word.

Entering visual mode is simplicity itself − simply hit

```
v
```

while in command mode. Vi will display a visual notification of the mode change at the bottom left corner of your screen. Now try using the various motion keys to move around − you'll notice that instead of moving you around the document, those keys now control a selection block which can be used to highlight sections of the text.

In addition to visual mode, there's also a "visual line" mode, which allows you to perform line−by−line selection and is activated by hitting

```
V [that's Shift−V]
```

and a "visual block" mode, which allows you to select vertical, rather than horizontal, blocks of text − hit

```
^−V [that's Ctrl−V]
```

to activate this mode. In all these different visual modes, the usual motion keys can be used to control the size of the selection block.

Now that you've got your text selected, it's time to do something with it − and vi comes with some built−in actions that allow you to interact with your selection. For example, type

```
d
```

to delete your entire selection, or

```
y
```

to copy it. You can then paste it by moving to the appropriate point in your document and hitting

```
p
```

The "change" command, activated in visual mode by typing

```
c
```

will delete the selected block and enter insert mode at the first character position of the deleted block.

If you're a programmer,

```
<
```

and

```
>
```

can be used to indent blocks of code left and right, respectively. And

```
~
```

can be used to switch the case of the selected text, from upper−case to lower−case and vice−versa.

A couple of other interesting commands in visual mode − type

```
o
```

to move the cursor between the upper left corner and lower right corner of your selection block, or

```
gv
```

to repeat the previous selection.

To exit visual mode, simply tap the

```
<ESC>
```

key, and you'll be returned to command mode.

As you'll see after you play with it for a while, visual mode is extremely powerful. It comes in particularly handy when you're dealing with large text files, and need to transpose sections of text from one place to another − instead of counting the number of lines to be copied, and then doing a

```
10dd
```

or a

```
21yy
```

you can simply select the lines of text with the motion keys, and use the operators above to transfer them to their new home. With power like this, who needs Santa Claus?

Re−thinking Relativity

According to Einstein, time is relative. According to Flootburger, time is money − especially when you're billing by the hour. And so, for all you smarmy consultants−to−be, here are some tips and tricks that should help you in bleeding the last cent from your eight−hour day.

First on the list: abbreviations. Vi allows you to define abbreviations for commonly−used words or phrases, and insert the complete word or phrase simply by typing its abbreviation. All this happens via the "abbreviate" command, which looks like this:

```
:abbreviate <abbreviation> <replacement−text>
```

or, for convenience,

```
:ab <abbreviation> <replacement−text>
```

From my own experience, I can tell you that this command is a boon to all those of us blessed with a somewhat longer−than−usual name. In my case, for example, I often use this abbreviation:

```
:abbreviate god Professor Elias Flootburger The Third
```

Go on − try it out yourself. Each time you type the word "god" in your document and follow it with either a space or a carriage return, vi will automatically replace it with the complete, unabridged version.

And once you get tired of seeing my name all over your document − how could you?! − you can remove the abbreviation with the

```
:unabbreviate god
```

or

```
:unab god
```

command.

To remove *all* abbreviations from memory, the command to use is

```
:abclear
```

Abbreviations are particularly useful for inserting frequently−used snippets of text − email addresses, telephone numbers and the like − into your documents. As a standard rule of thumb, if you find yourself typing the same piece of text more than five times in the same document, make it an abbreviation and use the money you save on RSI treatments to buy yourself a lifetime's worth of chocolate icecream − or simply donate it to the Flootburger Foundation For The Empowerment Of Semi−Retired University Professors. All donations are tax−deductible.

Vi also lets you enter symbols and accented characters, which cannot normally be entered with a regular keyboard. A digraphs table contains a list of different symbols, together with the key codes necessary to enter them; all you need to do is locate your symbol in the table, find the two characters that are mapped to create it, and enter them by typing

```
^−K <first−character> <second−character> [that's Ctrl−K]
```

For example, let's suppose you need to enter the copyright symbol ©

The first thing you need to do is look up the digraphs table to find out which combination of characters represents this symbol. You can access the digraphs table with the

```
:digraphs
```

command.

As you'll see, the digraphs table consists of a series of columns − the first two columns contain the characters which, when combined, create the symbol in the third column. The fourth column contains the decimal representation of the symbol. Our copyright symbol, for example, is represented by the characters "A" and "~", and the decimal code 169.

Now, in order to enter this symbol into your document, move to the appropriate location and enter insert mode by typing

```
i
```

Next, type

```
^−K <first−character> <second−character> [that's Ctrl−K]
```

In this case,

```
^−K A ~ [that's Ctrl−K, followed by A ~]
```

and your symbol will be entered into the document. Alternatively, if you'd prefer to use the decimal code, try

```
^−V <decimal code> [that's Ctrl−V]
```

In this case,

```
^−V 169 [that's Ctrl−V, followed by 169]
```

Sweet Revenge

In addition to abbreviations, vi also allows you map specific keys, or combinations of keys, to actions with its "map" and "map!" commands. The general format of a key map command [for normal and visual mode] is

```
:map <key−combo> <action>
```

while the format [for insert and command mode] is

```
:map! <key−combo> <action>
```

At its simplest, a key map is like an abbreviation. For example, here's one I use frequently when grading papers:

```
:map! gradef Terrible! This document has no merit whatsoever.
I'm awarding
you the lowest grade possible in the hope that it will
discourage you from
attending my class − God grant that I be so lucky!
```

Ah, Miss Schweppinger − I see you you're familiar with those words. I'm glad they left such a permanent impression on you − and I do applaud your courage in attending today's class. The Kleenex box is in the cupboard behind you.

You can also use the "map!" command to attach commands to specific keys, as in the following example, where I've mapped the Ctrl−H key combination to display help:

```
:map <C−H> :help<CR>
```

Now here's something a little more useful for all you Web developers. I like to map the Ctrl−N key combination to create an empty HTML document, according to a template I've previously defined. Here's what my mapping looks like:

```
:map <C−N> :read ~/templates/blank.html<CR>
```

where the file "blank.html" contains a standard HTML template.

A complete list of all current mappings can be obtained by typing

```
:map
```

or

```
:map!
```

while a mapped key or key combination can be deleted with the

```
:unmap <key−combo>
```

or

```
:unmap! <key−combo>
```

commands

You can use key mappings to turn vi into a truly powerful HTML editor, as Sven Guckes has done on his very interesting page at http://www.math.fu−berlin.de/~guckes/vim/source/html.vim − I suggest you take a look at it sometime.

And now for something truly evil:

```
:map! a b
:map! e f
:map! i j
:map! o p
:map! u v
```

I'm sure you see the potential for mayhem − especially if you're a system administrator with lots of enemies...

Vi, Robot!

If automation is your thing, vi is a dream come true; it allows you to record and play back the words you type, and even set or change configuration parameters when pre−defined events take place.

Let's talk about the record and playback functions first. To begin recording, make sure you're in command mode and then type

```
qa
```

where "a" is the name of the register to which all keystrokes will be written [you can use any of the 26 letters of the alphabet as a register identifier]. You'll notice a message at the lower left corner of your screen, indicating that all keystrokes are now being recorded.

Now type this in:

```
This is a recording. Please leave your message after the beep.
```

End the recording by typing

```
q
```

Now, in order to play it back, position the cursor on a new line, type

```
@a
```

where "a" is the register to be accessed, and watch as vi repeats your keystrokes for you.

You can precede the playback command with an integer, indicating the number of times it should be executed. So, in the example above, if you used

```
67@a
```

you'd get 67 lines, all saying the same thing − similar to a VCR on magic mushrooms, but not quite as visually entertaining.

Now, what about autocommands? In vi−lingo, an "autocommand" is a command that is executed when creating or editing a new file, reading an existing file, saving a file, or exiting the editor. Autocommands are defined with the "autocmd" command, which usually looks like this:

```
:autocmd <event> <file-pattern> <command>
```

or, in accordance with the time−is−money philosophy,

```
:au <event> <file-pattern> <command>
```

Vi comes with a whole list of <events> − for example, "FileReadPost" refers to what happens after a file is read into the editor with the "read" command, while "BufNewFile" is the event that is triggered when a new file is created. A complete list of events can be obtained from the documentation that comes with vi.

The <file−pattern> parameter is simply a wildcard specifying the types of files to run the command on − for example, use *.c for all C source code, or *.html for all HTML documents.

The <command> is the vi command to be automatically executed once both event and file conditions are satisfied. For example,

```
:au FileReadPost *.txt set ruler
```

would turn the ruler on each time I read a file with the .txt extension into the editor.

Or how about this:

```
:au BufNewFile *.html read ~/template.html
```

This would ensure that all new files created with a .html extension would automatically open with the default HTML template.

To display all autocommands in memory, try:

```
:autocmd
```

while

```
:autocmd!
```

will delete all existing autocommands from memory, leaving you with a clean slate for subsequent experiments.

The Shell Game

The guys who created vi built in all kinds of creature comforts designed to reduce your interaction with the shell to a minimum. One of the coolest ideas they came up with was a "shell filter", which allows you to replace shell commands with their output without needing to leave the editor at all.

Try this − open up a blank document and type

```
Today's date is date
```

Now exit insert mode, position the cursor over the second "date" in that line, and type

```
!!bash
```

The line should now read

```
Today's date is Tue May 2 15:54:13 IST 2000
```

or whatever the current date and time happens to be on your system. Vi passes the word under the cursor to the interpreter specified by the user − the "bash" shell, in this case − as a command, and then replaces the word with the output of that command.

Despite these time−saving features, there are times when you'll want to execute a shell command directly. In vi, you can do this with

```
:! <shell−command>
```

So, if you needed a quick directory listing of the /home directory, you could type

```
:! ls −l /home
```

and vi would pass the command to the shell, and display the output to you.

Finally, vi also allows you to spawn a new shell with the

```
:shell
```

command. You can then execute shell commands and run other programs − even a new instance of vi, although only the truly warped among you would find this entertaining.

Once you're done, simply log out of the spawned shell by typing

```
^−D
```

and you'll be returned to your original vi session.

First Aid 101

Unlike some of its better−known brethren, vi comes with some powerful crash recovery features. For example, let's suppose you're editing a file named "cart.php3", and your system suddenly crashes because Joe in the other room mistook the server's power button for the microwave's on/off switch.

Hey, some people are just born stupid−after all, only a moron would keep the microwave next to the server!

Well, once you get the system back up, recovering your file is as simple as typing

```
vi −r cart.php3
```

For every file that you edit, vi creates a temporary swap file by the same name and the file extension .swp in the same directory. When you add the −r parameter to the file name, vi will attempt to "recover" the file by reading the temporary backup and combining it with bits of the original file to get you back to where you were before the crash.

The temporary swap file is usually updated every four seconds or every 200 characters − although you can change this with the "updatetime" and "updatecount" commands.

Once you've confirmed that the restored version of the file is acceptable, you should delete the swap file to avoid error messages.

In addition to swap files, another very important file is the vi configuration file, usually found lurking in your home directory under the name ~/.vimrc . This configuration file is read every time you start the editor, and the commands and options in it executed automatically.

All the "colon" commands you've learnt thus far can be placed in the configuration file for automatic execution. There's one simple rule of thumb − if you execute the command while the editor is running, you usually need to precede it with a colon [as in the examples above]; however, the same command, when placed in the configuration file, will *not* be preceded by a colon. So, for example, though I might type

```
:abbreviate god Professor Elias Flootburger The Third
```

in the editor, my configuration file would only contain

```
abbreviate god Professor Elias Flootburger The Third
```

without the initial colon [:]

A simple way to create your .vimrc configuration file is to set the editor up the way you want, and then execute the command

```
:mkvimrc
```

which will create a configuration file for you automatically. And also take a look at Sven Guckes' heavily commented configuration file at http://www.math.fu−berlin.de/~guckes/vim/vimrc.forall for more information on the various options.

An Indent In Time...

Finally, if you're a programmer, vi comes with a number of features that you should find useful. Here are some tips that might help you the next time you sit down to code The Next Big Thing.

1. Let vi indent your code for you. Try

```
:set autoindent
```

and watch as vi automatically indents nested blocks of code like loops and conditional statements. C programmers might also like to take a look at

```
:set cindent
```

which offers some additional indenting styles.

2. In case you forget to turn auto−indenting on, all is not lost. Simply select the lines of code you wish to indent, and hit

```
=
```

Vi will do its best to indent the code correctly for you.

To illustrate the power of this trick, here's a "before" picture:

```
if(time == 13)
{
lunch();
}
else if (time == 22)
{
if (day == "Friday")
{
echo "Apple pie for dessert!":
}
else
{
echo "Oh no, not lasagna again!";
}
}
```

and here's the "after" picture:

```
if(time == 13)
{
lunch();
}
else if (time == 22)
{
if (day == "Friday")
{
echo "Apple pie for dessert!":
}
else
{
echo "Oh no, not lasagna again!";
}
}
```

3. Newer versions of vi comes with "syntax highlighting" − the ability to highlight program code in different colours. To turn this feature on, simply type

```
:syntax on
```

while

```
:syntax off
```

will turn it off.

Vi currently supports most popular programming languages − your distribution should come with syntax files for C, C++, HTML, Java, Perl, PHP and other common languages.

4. Vi also comes with the very useful "ctags" program, which can be used to create an index of classes, function definitions, variable declarations and methods for C, C++ and Java source code. This index, also known as a "tag" list, can be used by vi and other text editors to assist you in quickly locating sections of your source code while programming.

To use vi's tag features, it's first necessary to create a tag file − simply change to the directory containing your source code and run the "ctags" program on the files you wish to index, like this:

```
$ ctags *.c
```

You should now have a file named "tags" in that directory.

Now open one of your C files, and scroll down until you locate a function call. Position the cursor over that function call, and hit

```
^-] [that's Ctrl-square-brace]
```

Vi will automatically load the file containing the function definition, and display it to you. To go back to the original file, use

```
^-T [that's Ctrl-T]
```

This comes in very handy when your code is scattered across multiple files, and you just can't remember what the function popeye_loves_spinach() does.

Exit Flootburger

Well, I'm afraid that's about all we have time for today. As always, it's been a pleasure teaching such an enthusiastic class. We'll be having a pop quiz sometime next week − do make sure that you're better prepared this time, Mr. Higgleswipe.

Those of you that have "Economics 301" with Ms. Boodlekeg may exit through the back. The rest of you − please remain seated for the first session in my new course, "Daytrading 101", in which I'll be telling you all about how the stock market is actually a conspiracy between the government and an alien race − and then showing you how to beat them at their own game, and make a few million in the process. Go on − get your popcorn, and settle down!

Notes:

1. All examples/illustrations in this article have been verified on Linux/i586, kernel version 2.2.6, with vim 5.3. YMMV!

2. Microsoft Windows and Microsoft Word are trademarks of Microsoft Corporation. All other trademarks copyright their respective owners.