Vi 101

By Vikram Vaswani

Table of Contents

Introduction

New: Be sure to see Vi 201 when you're finished here!

Quiet, please! Quiet!

Ah, wonderful! It's nice to have such an obliging class for once; I wish all my students were this kind...

Allow me to introduce myself. I am Professor Elias Flootburger, a member of this august institution's Computer Science department, and the author of several well−received monographs on alien visitations, the occult, and the virtues of banana oil as a cheap substitute for carbon−based petroleum fuels.

I'm also the resident vi expert around here, which is why Principal Kinshott asked me to take this introductory class in vi basics. Looking at the blank expressions around me, I can see that none of you have any idea who or what vi is, let alone any inkling of how it's going to change your life...

Allow me to enlighten you.

Vi? V who?

Vi is a "visual editor" − an editor which lets you view a document, in its entirety, as you create and edit it. Those of you from the Microsoft Word generation might find that a little redundant − surely, you're thinking, all editors let you do that! Well, not exactly − in the early days of UNIX, line editors like edlin and ex were pretty standard, and creating a document in one of those babies involved using arcane commands to add, remove and retrieve text from an invisible buffer.

Times change, and almost all Unices today ship with some variant of vi − it's easily one of the most powerful editors out there, with a whole bunch of useful and interesting features. In my lighter moments − and they're few and far between, mind you! − I've been known to call it the Ferrari of text editors; it's fast, it's stylish, and once you know how to drive it, you'll never have any trouble finding a date on Saturday night...

Ah, I see you liked that analogy. Good, good...I have a feeling we're going to get along just fine!

The variant we're going to be dealing with throughout this course is called vim, which stands for "Vi IMproved"; since I'm going to be giving you a lot of homework, I suggest that you get yourself a copy of the latest version from http://www.vim.org as soon as possible. You might also like to take a look at the numerous other variants: elvis, at ftp://ftp.cs.pdx.edu/pub/elvis/; and vile, at http://www.clark.net/pub/dickey/vile/vile.html, are two of the better ones. While these clones are, by and large, true to the spirit of the original vi, many of them offer additional improvements and enhancements as well.

Oh yes − although we'll be using vim throughout this course, I'll be referring to it as vi. This quirk is just one of the many reasons women find me so attractive.

Start Me Up!

Starting a vi session is ridiculously simple; type

```
$ vi
```

at your UNIX shell prompt, followed by a carriage return, and vi will pop open an empty file for you to begin work. Alternatively, you could specify the file to open on the command line itself, like this:

```
$ vi myfile
```

Don't worry about the ~ signs in the editor − they're simply visual indicators of the bottom of your file, and don't actually appear in your document.

The first − and most confusing thing − about vi is the various different modes the editor can operate in. For our purposes, I'll split them into two broad categories: "insert mode", and "command mode". "Insert mode" is where you'll be spending most of your time − it allows you to add, or insert, text into your document. "Command mode," on the other hand, lets you move around the document, delete or replace blocks of text, and access useful vi functions like buffers and ranges.

When vi first starts up, it's usually in command mode. To enter insert mode, type

```
i
```

vi will display a visual notification of the mode change in the bottom left corner. You can now enter text into the document in the normal manner; when you're done, hit

```
<ESC>
```

to return to command mode.

To begin inserting text on a new line, try

```
o
```

to enter insert mode with the cursor already positioned on the next line, or

```
O
```

to enter it with the cursor on the previous line.

In addition to inserting text, vi also allows you to overwrite existing text − this is known as "replace mode", and can be entered by typing

```
R
```

from command mode. Everything you type will now "write over" existing text. If your intention was to replace a single character − for example, correcting a spelling error − use

```
r
```

to replace only a single character; the editor will automatically return you to command mode after the character has been replaced.

Obviously, at some point, you're going to need to save your work. In vi, this is accomplished via the "write" command, abbreviated to

```
:w
```

To quit, use the handy "quit" command, known to family and friends as

```
:q
```

vi also lets you combine frequently−used commands together − to really impress the pretty girls, not to mention add a few points to your geek quotient, try

```
:wq
```

for powerful save−and−exit functionality. Alternatively, if you've decided to bury whatever you've just written, try

```
:q!
```

which exits vi without saving any changes to the file being edited.

Obviously, all these "colon" commands need to be executed in command mode − attempting them while in insert mode will result in them appearing as part of your document, and no professor likes to receive assignments interspersed with :wqs and :q!s. Well, actually, some of them do − but that's mostly the younger crowd, and what can you expect from them? Those punks actually think rock 'n' roll is dead...

Let's Tango!

Now that you know how to insert text, it's time to teach you the second most confusing thing about vi − moving around within the document. In order to gain the maximum benefit from this lesson, I suggest you either open an existing document, or create a new one and enter a few lines of text into it. Make sure that you're in command mode by hitting

```
<ESC>
```

a few times when you're done.

Vi's movement keys are

h = left
j = down
k = up
l = right

On some UNIX consoles, it may also be possible to use the cursor keys to accomplish movement within the document; however, I suggest that you take a few moments to memorize the list above instead, as cursor key movements can sometimes get lost over telnet connections, while the standard vi keys above will work in any situation.

Care to move between words? Try

```
w
```

to move one word forward, or

```
b
```

to move one back.

To move between paragraphs, try

```
{
```

and

```
}
```

while moving between sentences requires careful use of

(

and

)

If you need to move to a specific line in your docume nt − line 568, for example − try this

:568

To move to the beginning of the file, type

gg

while the end of the file is just a

G [that's shift−G]

away. And vi also has an equivalent for the "Page−Up" and "Page−Down" commands common in other text editors; try

^F [that's Ctrl−F]

to move one page forward, and

^B [that's Ctrl−B]

to move one page back.

To find out your exact position in the document [kinda like GPS, but not as cool!], try

^G [that's Ctrl−G]

Of course, these aren't the only motion keys − there are quite a few more, but I'm afraid that you'll only hear about them in "Vi 202". For the moment, these should be sufficient for our next exercise, in which you'll be doing the tango with your keyboard. Come on, you know how this goes − two steps left, one step right, three steps back, and shake it all about...

Lather, Rinse, Repeat...

Next up, deleting and copying text. To delete a single character, position the cursor under it, make sure you're in command mode, and type

```
x
```

to delete it. To delete a word, hit

```
dw
```

which stands for "delete word". Nuking an entire line is accomplished with

```
dd
```

while novelists with writers block will appreciate the thoughtfulness of the

```
dgg
```

and

```
dG
```

commands, which delete everything from the current cursor position to the beginning and end of the file respectively.

Oops − you didn't just try that last one, did you? Not to worry − vi also comes with a very handy "undo" command, which can be accessed by hitting

```
u
```

whenever you feel the urge to undo your previous mistakes. And since undo and redo go together like strawberries and cream, you might want to check out the "redo" command, which allows you to repeat past actions by typing

```
. [yup, that's a period]
```

When you delete something in vi, the deleted text usually finds its way to a temporary buffer in memory, where it resides until it is replaced. This deleted text can be re−inserted into your document with the "put" command; if you've been following along, you already know that the "put" command is accessed with

```
p
```

Go on − try it. Delete a line, move down to a new paragraph, and put it back in a different place. Simple − and equivalent to the cut−paste functionality available in other editors.

In case you'd like to insert the deleted text *above* the current cursor position, simply use

```
P
```

instead of

```
p
```

What about copy−paste? Also pretty easy − vi uses the "yank" command to copy text in much the same way as the delete command is used to delete text. To yank a particular word, try

```
yw
```

while

```
yy
```

yanks a complete line and places it in the temporary buffer, ready to be "put" somewhere new.

You're probably wondering what all this has to do with shampoo. Absolutely nothing − this particular lesson was originally titled "Cut, Copy, Paste...", but I thought "Lather, Rinse, Repeat..." had a nicer ring to it...

One, Two, Buckle My Shoe...

One of vi's most powerful features is the ability to automatically repeat a command a specified number of times, by preceding the command with a number. For example, let's suppose you needed to delete eight lines of a document. Based on what you've just learnt, you would use the

```
dd
```

command eight times, creating the truly horrible command string

```
dddddddddddddddd
```

What you could have done, had I informed you of it earlier, was

```
8dd
```

As you'll see when you try it out, this simply executes the command eight times, deleting eight lines from the document without needing you to run to the drugstore for an aspirin after. And this technique can be used with any of the commands you've already learned − you can use it to copy multiple lines, repeat insertions a specified number of times, and even move the cursor to specific occurrences of words within the document.

Searching For Hope

Vi also comes with powerful search and replace features, both based largely on regular expressions. Unfortunately, regular expressions are not part of this course of study; they will be covered in "Reg−ex 101", and so I'm going to restrict myself to an explanation of vi's string search functions, which can be accessed with the / and ? commands.

Let's say that you've written a treatise on the role played by Mark Hamil in "Star Wars: A New Hope" − except that, when proof−reading your golden prose, you find that you've mistakenly spelt the word "hope" as "dope" in different places in your document. Obviously, you can't ignore this error − it would lead to a completely new interpretation of the planet's favourite sci−fi flick. What do you do?

```
/dope
```

The /pattern command tells vi to search the document for the specified pattern; once it finds it, it will position the cursor at the first character of the string, and await further commands.

To search backwards, use the ?pattern command, like this:

```
?dope
```

It's unlikely − not to mention naïve on your part − to assume that this error would only have occurred once. You can repeat the last search by pressing

```
n
```

or, in vi lingo, "next match".

It isn't enough to just identify your mistake − the next step, according to most major religious doctrines and television preachers, is to repent and make sure it doesn't happen again. Vi can help you there too − although it will involve memorizing a command which, at first glance, is almost enough to make you wonder if there really is a God...

```
:1,$s/pattern_to_find/pattern_to_replace/g
```

In case you're wondering where the "s" came from − it's vi−talk for "substitute". The 1 and the $ are symbols indicating the range within which the substitution is to take place − in this case, from the beginning of the document [line 1] to the end [$]. The "g" is a flag indicating that *all* matches should be replaced.

And so, in the above example, the command to replace the word "dope" with "hope" would be

```
:1,$s/dope/hope/g
```

Now, if only getting Leia's phone number was this easy...

Bookmarks, Buffers and Beach Bunnies

Vi also allows you to define "bookmarks" within your document − this makes it possible to easily jump to and between specific sections of large files. As an example, consider the following blocks of text:

...this document outlines the proposed corporate structure for Muscle Cars, Inc., a company engaged in the business of manufacturing and selling...[page 2]

...the CEO and President of Muscle Cars, Inc., James van Hausen, who, in his misbegotten childhood, was best−known for wrapping expensive sports cars around trees, has succeeded in...[page 137]

So let's set a couple of bookmarks, shall we? On page 2, move your cursor to the word "document", and type

```
ma
```

This sets a bookmark, identified by the letter "a", at the cursor position. Now find your way to page 137, and set a bookmark named "b" at the word "cars". To switch back to page 2, all you now need to do is type

```
`a
```

and to get to page 137, simply type

```
`b
```

Any lowercase letter can be used as a bookmark identifier. Obviously, you're limited to a total of 26 bookmarks per file − but in most cases, that's more than enough. If you need more than that, I strongly suggest you try therapy.

In addition to multiple bookmarks, vi also supports multiple buffers. If you've been paying attention, you know that all the material you delete or copy finds its way to a temporary buffer until it's either replaced or removed. But vi also has "named buffers" − similar to a set of storage lockers, these buffers can be used to store different blocks of text, and make them available for insertion whenever required.

As an example, take a look at this section of an essay handed in to me by one of my former students:

...beach bunnies seem to have a pretty good life of it. All they seem to do is lounge around on the beach, drinking pina coladas and slathering suntan lotion all over themselves. Once in a while, they get up and toss a volleyball around. Sounds like fun − where do I sign up?...

I'm sure you understand why I said "former student" now...

Now let's suppose that I wanted to transpose the phrases "drinking pina coladas" and "slathering suntan lotion". Here's what I would do:
1. Move my cursor to the beginning of the word "drinking"

2. Delete the three words, and move them to a buffer named "p"

```
"p3dw
```

3. Move my cursor to the beginning of the word "slathering"

4. Delete those three words, and move them to a buffer named "s"

```
"s3dw
```

5. Move my cursor back to before the word "and"

6. Insert ["put"] the contents of buffer "s"

```
"sp
```

7. Move my cursor to after the word "and"

8. Insert the contents of buffer "p"

```
"pp
```

9. Congratulate myself loudly and exuberantly, as I'm doing right now!

As you can see, to use a buffer, simply precede the regular copy/delete/replace command with the buffer identifier. A buffer identifier consists of a double−quote mark, followed by a lowercase letter. In the example above, both "s and "p are buffer identifiers.

I should add here that neither bookmarks nor buffer contents are retained once you exit the editor.

Of Files And Windows

You've already seen that

```
$ vi <filename>
```

starts vi with the file specified already loaded. But why restrict yourself to one − you can load multiple files in this manner:

```
$ vi file1 file2 file3
```

To switch between files, use these commands:

:next = go to the next file in the file list
:rewind = go to the first file in the file list
:last = go to the last file in the file list

To exit all files at once, use

```
:qa!
```

to exit without saving, or

```
:wqa
```

to save all changes and exit [the "a" here means "all"]

Once you've started vi, you can also load a new file into the editor with the "edit" command, like this

```
:e /path/to/file.txt
```

If you'd prefer to insert the contents of another file directly into the document you're currently editing, there's a "read" command designed to do just that. Try it!

```
:read /path/to/file_to_be_inserted.txt
```

If you're a programmer, you'll definitely appreciate vi's ability to provide you with multiple views of the same file, referred to in geek speak as "window splitting". And the command is, naturally,

```
:split
```

Each of the windows created can be manipulated independent of the others, although changes made to the file will be immediately visible in all of them.

Let's take it one step further − how about having *different* files loaded into each of the windows? For example, wouldn't it be nice if you could load an HTML form into one, the associated CGI script into another, some help documentation into the third, and have all three visible at the same time?

Well, you can − simply use the "new" command to create a new window for each file, like this:

```
:new form.html

:new mailform.cgi

:new help.txt
```

To switch between windows, use

```
^W W [that's Ctrl-W, immediately followed by W]
```

Saved By The Bell

Oh, there goes the bell! How time flies when you're having fun...

Anyway, I hope you enjoyed this introductory lesson − your attention has been most flattering. As I tell my students every year, vi is like a sports car; it's only after you've understood the temperament of the beast that you can actually enjoy the ride. So hop in − and remember, don't drink and drive!

New: Vi 201 is now available!