An Extremely Quick and Simple Introduction to the Vi Text Editor

Norm Matloff

last updated October 18, 2003

Contents

1 Overview

A **text editor** is a program that can be used to create and modify text files. One of the most popular editors on Unix systems (it is also available on Windows and many other platforms) is **vi**.

2 5-Minute Introduction

As a brief introduction to vi, go through the following: First, type

```
vi x
```

at the Unix prompt. Assuming you did not already have a file named x, this command will create one. (If you have tried this example before, x will already exist, and **vi** will work on it. If you wish to start the example from scratch, simply remove x first.)

The file will of course initially be empty. To put something in it, type the letter 'i' (it stands for "insert-text mode"), and type the following (including hitting the Enter key at the end of each of the three lines):

```
The quick
brown
fox will return.
```

Then hit the Escape key, to end insert-text-mode.

This mode-oriented aspect of the vi editor differs from many other editors in this respect. With modeless editors such as joe and emacs, for instance, to insert text at the cursor position, one simply starts typing, and to stop inserting, one just stops typing! However, that means that in order to perform most commands, one needs to use the Control key (in order to distinguish a command from text to be inserted). This has given rise to jokes that heavy users of modeless editors develop gnarled fingers.

Now save the file and exit **vi**, by typing 'ZZ' (note the capitals).

Again, the key to learning vi is to keep in mind always the difference between insert-text mode and command mode. In the latter mode, as its name implies, one issues commands, such as the ZZ above, which we issued to save the file and exit **vi**. The characters you type will appear on the screen if you are in insert-text mode, whereas they will not appear on the screen while you are in command mode. By far the most frequent problem new **vi** users have is that they forget they are in insert-text mode, and so their commands are not obeyed.

For example, suppose a new user wants to type ZZ, to save the file and exit **vi**, but he has forgotten to hit the Escape key to terminate insert-text mode. Then the ZZ will appear on the screen, and will become part of the text of the file—and the ZZ command will not be obeyed.

You now have a file named x. You can check its contents by typing (at the Unix shell prompt)

```
more x
```

which will yield

```
The quick
brown
fox will return.
```

just as expected.

Now let's see how we can use **vi** again to modify that file. Type

```
vi x
```

again, and make the following changes.

First, suppose we wish to say the fox will *not* return: We need to first move the cursor to the word "return". To do this, type '/re' and hit the Enter key, which instructs **vi** to move the cursor to the first instance of 're' relative to the current cursor position. (Note that typing only '/r' would have moved the cursor to the first instance of 'r', which would be the 'r' in 'brown', not what we want.)

Now use the 'i' command again: Hit 'i', then type 'not ' (note the space), and then hit Escape.

Next, let's delete the word 'brown'. Type '/b' to move the cursor there, and then hit 'x' five times, to delete each of the five letters in 'brown'. (This will still leave us with a blank line. If we did not want this, we could have used the 'dd' command, which would have deleted the entire line.)

Now type 'ZZ' to save the file and exit vi. Use 'more' again to convince yourself that you did indeed modify the file.

3 Going Further: Other Frequently-Used Commands

You now know how to use **vi** to insert text, move the cursor to text, and delete text. Technically, the bare-bones set of commands introduced above is sufficient for any use of vi. However, if you limit yourself to these few commands, you will be doing a large amount of unnecessary, tiresome typing.

So, you should also learn at least some of these other frequently-used **vi** commands:

h	move cursor one character to left
j	move cursor one line down
k	move cursor one line up
l	move cursor one character to right
w	move cursor one word to right
b	move cursor one word to left
0	move cursor to beginning of line
$	move cursor to end of line
nG	move cursor to line n
control-f	scroll forward one screen
control-b	scroll backward one screen
i	insert to left of current cursor position (end with ESC)
a	append to right of current cursor position (end with ESC)
dw	delete current word (end with ESC)
cw	change current word (end with ESC)
r	change current character
~	change case (upper-, lower-) of current character
dd	delete current line
D	delete portion of current line to right of the cursor
x	delete current character
ma	mark currrent position
d‘a	delete everything from the marked position to here
‘a	go back to the marked position
p	dump out at current place your last deletion (‘‘paste’’)
u	undo the last command
.	repeat the last command
J	combine (‘‘join’’) next line with this one
:w	write file to disk, stay in vi
:q!	quit VI, do not write file to disk,
ZZ	write file to disk, quit vi
:r filename	read in a copy of the specified file to the current buffer
/string	search forward for string (end with Enter)
?string	search backward for string (end with Enter)
n	repeat the last search (‘‘next search’’)
:s/s1/s2	replace (‘‘substitute’’) (the first) s1 in this line by s2
:lr/s/s1/s2/g	replace all instances of s1 in the line range lr by s2 (lr is of form ‘a,b’, where a and b are either explicit line numbers, or . (current line) or $ (last line)
:map k s	map the key k to a string of vi commands s (see below)
:abb s1 s2	expand the string s1 in append/insert mode to a string s2 (see below)
%	go to the "mate," if one exists, of this parenthesis or brace or bracket (very useful for programmers!)

All of the ‘:’ commands end with your hitting the Enter key. (By the way, these are called “ex” commands, after the name of the simpler editor from which **vi** is descended.)

The ‘a’ command, which puts text to the right of the cursor, does put you in insert-text mode, just like the ‘i’ command does.

By the way, if you need to insert a control character while in append/insert mode, hit control-v first. For example, to insert control-g into the file being edited, type control-v then control-g.

One of **vi**’s advantages is easy cursor movement. Since the keys h,j,k,l are adjacent and easily accessible with the fingers of your right hand, you can quickly reach them to move the cursor, instead of fumbling around for the arrow keys as with many other editors (though they can be used in **vi** too). You will find that this use of h,j,k,l become second nature to you very quickly, very much increasing your speed, efficiency and enjoyment of text editing.

Many of the commands can be prefixed by a number. For example, 3dd means to delete (consecutive) three lines, starting with the current one. As an another example, 4cw will delete the next four words.

The p command can be used for “cut-and-paste” and copy operations. For example, to move three lines from place A to place B:

1. Move the cursor to A.

2. Type ‘3dd’.

3. Move the cursor to B.

4. Type ‘p’.

The same steps can be used to copy text, except that p must be used twice, the first time being immediately after Step 2 (to put back the text just deleted).

Note that you can do operations like cut-and-paste, cursor movement, and so on, much more easily using a mouse. This requires a GUI version of **vi**, which we will discuss later in this document.

4 Advanced Topics

4.1 Macros

When you are using **vi**, you can use the ‘map’ and ‘abb’ commands to save a lot of typing. For example, I often accidentally transpose two letters when I am typing fast, say typing ‘taht’ instead of ‘that’. Since I do this so often, I place the command

```
:map v xp
```

which means that the v key now performs the operations ‘x’ and ‘p’, (try ‘xp’ yourself and you will see it work), in my

```
~/.exrc
```

file ((without the colon; see below).

Also, since I often edit HTML files, I save myself typing by including lines like

```
abb cg <FONT color=green>
```

in my .exrc file. This means that whenever I am vi's insert/append mode and type "cg" and then hit the space bar, **vi** will automatically expand it to "¡FONT color=green¿".

Here are some more examples:

```
map ; $
map - 1G
map \ $G

map ^K ~

map ^X :.,$d^M

map! ^P ^[a. ^[hbmmi?\<^[2h"zdt.@z^Mywmx‘mP xi
map! ^N ^[a. ^[hbmmi/\<^[2h"zdt.@z^Mywmx‘mP xi

abb taht that
abb wb http://heather.cs.ucdavis.edu/~matloff
```

The first three simply perform cursor movement (to end-of-line, start-of-file, end-of-file) Most of them only saves one keystroke, but they require much less finger movement (for the standard touch-typing hand position) and since they are such frequently-used operations they are worthwhile. The fourth map is for case change, again (for me) a frequent operation.

The fifth map deletes all material from the current cursor position to the end of the file. I often find this useful, when editing a reply to an e-mail message for instance, or when I use :r to import another file into the one I am editing.

The sixth and seventh maps, which are labeled "map!" instead of "map" to indicate that they operate during append or insert mode, are modifications of some macros which are "famous" in the vi user community. They are used for "word completion," an extremely useful trick to save typing. Suppose for example I am currently in append/insert mode and I wish to type the word "investigation," and that I have used the word previously. If I just type, say, "inv" and then control-p, **vi** will search for a word earlier in my file which began with "inv" and complete my "inv" to that word, in this case "investigation ". Typing control-n will do the same thing, except that it will search forward instead of backwards.

Note again that in typing these macros in one's .exrc file, one must hit control-v first. For example, to insert control-g into the file being edited during append/insert mode, type control-v then control-g.

4.2 The .exrc Startup File

When you invoke the **vi** editor, it will look for the file

```
~/.exrc
```

and obey any "ex" commands it finds there. For example, I have lines in my startup file corresponding to the map and abb examples in the last section:

```
map v xp
abb cg <FONT color=green>
```

(Note that in the .exrc file we omit the colon, i.e. we type "map" instead of ":map", because **vi** assumes these are all "ex" commands.) That way I have those settings (and many others) permanently set, rather than my needing to type them in again each time I use vi.

4.3 Mainly for Programmers

There are a number of editing commands available in **vi** and most other sophisticated text editors which are especially useful for programmers. They are described in

`http://heather.cs.ucdavis.edu/~matloff/progedit.html`

The reader is urged to make daily use of these, which can really save a lot of time and effort.

4.4 Lots and Lots About Vi

There is a large compendium of information about **vi** at

`http://www.math.fu-berlin.de/~guckes/vi/`

A nice compact reference for **vi** commands is available at

`http://www.ungerhu.com/jxh/vi.html`

5 Other Editors, Including Other Versions of Vi

Arguments over the pros and cons of various editors become almost "religious" in their ferocity. I have tried all of the major Unix text editors, but have always come back to **vi**.

In my view, the best versions of **vi** currently available are **elvis** and **vim**. Both have really good features, especially their X11 GUI versions. It is much easier to do a cut-and-paste operation, for example, using the mouse instead of "by hand."

I have Web pages on both of these versions of vi, at

`http://heather.cs.ucdavis.edu/~matloff/vim.html`

and

`http://heather.cs.ucdavis.edu/~matloff/elvis.html`

But in the end it is a matter of taste. I have used **vi** as the introductory editor here because it is so prevalent in the Unix world, but you may wish to give others, say emacs or some of the X11-only editors, a try.