VI (and Clone) Introductory Guide

Susan Liebeskind
(shl@cc.gatech.edu)

Miles O'Neal
(meo@rru.com)

Systems & Software Solutions

December 12, 1998

ABSTRACT

Vi, the standard screen editor on Unix machines, is a very powerful editor. However, it is quite different from word processors or editors in most other environments. This document serves as a friendly introduction to vi. Once you are comfortable with the commands described here, consult the Vi Editor Reference Manual.

Revision History:

Rev. 1.0 Aug 22, 1989 shl
Rev. 1.1 Jan 9, 1990 meo
Rev. 1.2 Dec 31, 1991 meo
Rev. 1.4 Mar 22, 1994 meo
Rev. 1.5 Aug 3, 1998 meo
Rev. 1.6 Dec 12, 1998 meo

Table of Contents

1. Introduction

This guide will help you with vi (the visual editor), the standard unix text editor. It is by no means complete - vi can do a lot more for you. However, mastering the commands in this guide will give you the basics for quick, efficient editing with vi. The information here also applies to all the clones of vi, such as vim, elvis, and stevie.

1.1. Using this document

When I specify commands, I will set them off on a line by themselves, and precede them with ==> , like this:

```
==>  xyzzy
```

(This is not a vi prompt; vi almost never displays a prompt.) Wherever possible, I have tried to give you some sort of mnemonic device for associating the command with its function. Some are better than others.

1.2. Vi concepts

The editor is **case sensitive**. This means that when you enter commands, you must enter them exactly as specified, upper, lower, or mixed case. For example, the command

```
==>  A
```

and the command

```
==>  a
```

do two different things. If you use the [CAPS LOCK] key when entering data, remember to toggle it off when you finish.

Commands in vi are a sequence of characters, not characters which must be pressed together. Only control key commands, specified by the string **[CTRL]**, followed by another key, must be entered by pressing the Control key and the other key at the same time. Control key commands are not case sensitive. If you need to press return following a command, I have specified **[RETURN]** after the command.

You can always abandon the vi session without saving your work. No work is saved until you say so - this is your ace in the hole.

The term white space means any character that is not actually shown on the terminal. This includes **[SPACE]**, **[TAB]** and **[RETURN]** characters.

2. Command vs Input Modes

2.1. What are the editing modes?

There are two modes in vi:

- command mode
- input mode.

Command mode is the editing mode in which editing commands (delete, exit, move the cursor to the previous line etc.) are issued. This is the initial mode when the you first enter vi.

Input mode is the mode for entering text into the file. Everything you type in this mode will become part of whatever you are editing.

2.2. How to enter command mode

As mentioned earlier, you begin every vi session in command mode. You can also enter command mode from input mode by pressing the **[ESC]** key. If you are not sure what mode you are in, press **[ESC]** until the terminal beeps at you. At this point, you are definitely in command mode.

2.3. How to enter input mode

Input mode is entered by issuing the append command

```
==>  a
```

This command inserts text beginning to the right of the cursor. or by issuing the insert command

```
==>  i
```

This command inserts text beginning to the left of the cursor.

Mnemonics:

> Keep the command pair straight by remembering that the command ‘a’ means append (a)fter the cursor.

There are two other ways to enter insert mode which give you a blank line on which to enter data.

```
==>  O
```

This is the capital letter Oh, not a zero. This command will put a new line above the cursor, and put you in insert mode.

```
==>   o
```

This will put a new line below the cursor, and put you in insert mode.

Mnemonics:

> The ‘O’ and ‘o’ represent (o)pening a new line for input. The big O takes precedence, so it goes abOve. The little o goes below.

2.4. How to enter data in input mode

Type away. Pressing the **[RETURN]** key will advance you to the next line. Pressing either the **[DELETE]** key or the [BACK SPACE] key will allow you to backspace over what you have typed. The only exception is that you cannot backspace over the beginning of a line; you must change to command mode to move anywhere on a previous line.

2.5. How to leave input mode

Since there are only two editing modes in vi, this is the same as entering command mode. Press the **[ESC]** key. If you are unsure of your mode, you may press it two or three times until you hear a beep.

2.6. Undo Command

This command allows you to undo the last action you just took.

```
==>   u
```

If you issue this command in command mode, it will undo the last change you just made. If you issue the undo command again it restores the change (undoes the undo).

NOTE: Some vi clones allow multiple undo commands. See your editor’s man page or other documentation for details.

Mnemonics:

> ‘u’ means (u)ndo the last change.

This is very useful when you have accidentally garbled the last line of text you entered and want to get back what you started with. WARNING!!! Undo only restores the last change you made. If you have made massive changes which you wish to undo, you will have to quit vi without saving your text; undo won’t help. For this reason you may wish to save your work fairly often at times when you are certain you want to keep the changes you have made so far. Alternatively, save the changes to a different file name.

3. Cursor Motion

3.1. Moving a character at a time

On most terminals and workstations the cursor keys on the keypad will move you up, down, left or right when you are in command mode. On some keyboards, you must use the following keys to move around in command mode (these will work on all keyboards):

```
==>  h
```

This will move the cursor back (left) 1 character.

```
==>  l
```

This will move the cursor forward (right) 1 character.

```
==>  j
```

This will move the cursor down 1 line.

```
==>  k
```

This will move the cursor up 1 line.

Mnemonics:

The keys are all in a row on the middle row of letters on the standard keyboard. From left to right, the cursor motion associated with the keys is:

```
 Basic Cursor Motion
 h     j     k     l

<--    |     ^    -->
       |     |
       v     |
```

Of the group,

- ‘h’ is leftmost, which helps to remind you that it will move left.
- ‘l’ is rightmost, which helps to remind you that it will move right.
- I am hardpressed to come up with a good mnemonic for ‘j’ (down - jump down?) and for ‘k’ (up - kick up?).[1]

3.2. Moving a screen at a time

```
==>  <CTRL>F or <CTRL>f
```

This command will move you forward one full screen. If you are at the end of the file, the terminal will beep instead. The ‘f’ key can be upper or lower case.

```
==>  <CTRL>B or <CTRL>b
```

This command will move you back one full screen. If you are at the beginning of the file, the terminal will beep instead. The ‘b’ key can be upper or lower case.

Mnemonics:

'F' means forward through the file and 'B' means backward through the file.

3.3. Moving 1/2 screen at a time

```
==>  <CTRL>D or <CTRL>d
```

This command will move you forward one half screen. If you are at the end of the file, the terminal will beep instead. The 'd' key can be upper or lower case.

```
==>  <CTRL>U or <CTRL>u
```

This command will move you back one half screen. If you are at the beginning of the file, the terminal will beep instead. The 'u' key can be upper or lower case.

Mnemonics:

'D' means down through the file and 'U' means up through the file.

3.4. Moving to start and end of a line

```
==>  ^ or 0
```

Pressing the '^' key (usually the shifted 6 key on the main keyboard) will move the cursor to the first non-white-space character of the current line.

Pressing the '0' (zero) key moves the cursor to the first character of any type on the current line.

```
==>  $
```

Pressing this key will move the cursor to the end of the current line.

Mnemonics:

0 ^Think of this line here.$

3.5. Moving from start to end of a file

```
==>  1G
```

Pressing this key sequence will place the cursor on the first character of the first line of the file.

A 'G' preceded by any other positive integer will go to that line number in the file, if the file has that many lines. Otherwise the terminal will beep at you.

```
==>  G
```

Pressing this key will place the cursor on the first character of the last line of the file.

Mnemonics:

> Think of ‘G’ as standing for (G)o to a line. The 1G command means go to line 1 and 123G means go to line 123. The ‘G’, with no preceding number, means go to the bottom line. Since its physical line number may vary, you let vi figure it out; no line number needs to be specified.

4. Editing Commands

These are the commands which operate on your text in command mode. I have listed only a small subset. As you become comfortable see the VI Editor Reference Manual or other vi manuals for additional suggestions.

4.1. Delete Commands

```
==>   x
```

This command removes the character under the cursor.

Mnemonics:

> You ‘x’ out or cross out the character underneath by deleting it.

```
==>   dd
```

This command deletes the current line.

```
==>   dw
```

This command deletes the current word.

Mnemonics:

> A mnemonic would be ‘d’ for (d)elete something. You add a ‘w’ to affect a word. You double the ‘d’ to affect a line.

4.2. Replace Commands

These commands replace/overwrite existing text.

```
==>   r
```

This command allows you to replace the current character.

```
==>   cc
```

This command allows you to change the contents of the current line.

```
==>   cw
```

This command allows you to change the contents of the current word.

Mnemonics:

> ‘r’ is for replacing a character. ‘c’ is for (c)hanging something. You add a ‘w’ to affect a word. You double the ‘c’ to affect a line.

4.3. Search Commands

These commands let you search the current file looking for a given string. When you press the ‘/’ or ‘?’ key, the cursor will move to the bottom of the screen, where you will enter the string you wish to search for. Pressing the **[RETURN]** key will signal the end of the string, and cause the search to take place.

To include a ‘/’ or ‘?’ (or the ‘’) as part of the search string, you must "escape" any of these characters with the ‘’ character.

`==>  /<search string>`**`[RETURN]`**

This command will let you search forward through the file looking for the string. The search will wrap around to the start of the file.[2]

`==>  ?<search string>`**`[RETURN]`**

This command will let you search backward through the file looking for the string. The search will wrap around to the end of the file.[3]

Mnemonics:

> The ‘/’ is a forward slash, and is used to search forward. The ‘?’ on standard keyboards is the shifted ‘/’ key, and you can think of the act of shifting as reversing the command direction.

To repeat the search, press ‘/’ or ‘?’ depending on whether you want to search forward or backward for the next occurrence. When the cursor moves to the bottom of the screen, press **[RETURN]** without entering anything. Vi will use the last search string entered as its target.

4.4. Saving & Exiting Commands

These commands allow you to save or abandon your work as you choose. You can save and quit ONLY from command mode. All these commands begin with a ‘:’ (colon). Once you enter the ‘:’, the cursor moves to the bottom of the screen where the remainder of the command is entered.

`==>  :w`

This command will save all changes that you have made so far. You will still be editing the file when this command finishes.

`==>  :q!`

This command will abandon all changes that you have made since you entered vi, or since you last wrote the file with the :w command, whichever came later.

```
==>   :wq
```

This command will save your work and exit vi immediately after the save.

Mnemonics:

'w' is for (w)riting the file and 'q' is for (q)uiting vi.

5. Help!

How do I know if I am in input mode or command mode?

If everything you type appears on screen, you are in input mode. If you are not sure, press **[ESC]** until you hear a beep. You will then be in command mode for certain.

I am trying to use the cursor keys to move around but things aren't working properly!

You are probably in input mode. Press **[ESC]** till you hear the beep.

I'm typing in text, but it is not going into the file, and the cursor is moving and the terminal is beeping...

You are in command mode. Move the cursor to the place where you want to begin inserting text, and issue an insert command.

I just changed the last line inadvertantly. How can I get back what I had before?

Issue the undo command, 'u'. Your line will be restored. Recall that you will only get your last change back, though.

I don't know what I did. HELP!

Relax. No work is saved until you issue the :w or the :wq command. Here is a sure fire-way to abandon without saving:

1. Press **[ESC]** till you hear the beep.
2. Enter the command

```
==>   :q![RETURN]
```

You have abandoned all work in your last editing session since you last wrote the file with a :w command, or since you invoked the editor, whichever came later.

Notes:

1. *The use of these characters dates back to the teletypewriter days of computing. If this doesn't mean anything to you, be grateful!*
2. *Most systems default to wraparound in this fashion. An option is available to set this via ":set*

*wrapscan**[RETURN]**" or unset it via ":set nowrapscan**[RETURN]**".*
3. *See previous footnote.*

Last updated: *12 December 1998*

Miles O'Neal <meo@XYZZYrru.com> [remove the XYZZY to make things work!] c/o RNN / 11501 Johnson Rd / Leander, TX / 78641-5823