The VI nano-HOWTO for the NewbieDoc Project

Romain Lerallut

rom1@catsrule.garfield.com

VI: a powerful console-based text editor for every user and administrator. This documentation is part of the NewbieDoc Project (http://www.egroups.com/group/newbieDoc/).

1. Introduction

VI is one of the most used text editors in the UNIX world. It is found on most (if not all) UNIX workstations, *and not only on Linux ones* , contrary to most other text editors.

It is very powerful, which means it can be hard to do simple things, BUT it also means that complicated thing are equally hard to do. So if you manage to understand how VI works for simple things, you can quickly move on to very advanced text editing.

[Besides, it looks awfully cool to edit text in vi, rather than in emacs or other text editors that actually display command keys, or even have menus :-)]

2. Credits

I wish to extend my thanks to the all the doc writers of the NewbieDoc Project (http://www.egroups.com/group/newbiedoc), and to those who submitted patches to make this doc better (most notably DMan whose patch staid there, forgotten, for quite some time...). And also to Tom Huckstep for his SGML templates and detailed howto, his patch(es ?) and of course to Will Trillich for starting this whole thing.

Thanks...

3. Starting VI

You can start vi by typing **vi** alone (that wasn't hard to figure out, was it ?),

or you can specify a filename: **vi hop.txt**

In this case, either:

1. the file exists already and you open it for edition
2. the file doesn't exist and vi starts a new file with that name

4. Quitting VI

Written for Xucaen and all the ones who ever got stuck in the vi quicksands :-)

Sometimes people start vi "just to see what it looks like" and can't find how to getout !!

To quit vi, press ESC at least once to quit whichever mode you're in (the box should 'beep') and type **:q** for normal quit. *[Don't forget the : !!]*

Other options:

- write and quit **:wq** (aka **:x**)
- force quitting **:q!** quits VI even if the file was not saved (it is most useful when editing a read-only buffer, eg: forgetting to su root before editing a /etc/xxxx.conf file. Happens to me all the time :-)

5. The Command and Insert Modes

There are two main modes in VI: *command* and *insert*. To that we can add the "ex" mode, called up with ":". Cursor mode is the default mode when starting vi, it's also the top-level mode. To go back to it when in any other mode, press **ESC** a few times (once is actually enough for most command modes).

5.1. Cursor motions

There are tons of ways to move the cursor around, so I'll just give the basic ones:

- **Normal motion:** cursor keys, (which are emulated by **h**,**j**,**k**,**l** if the terminal (or the currrent keymap) doesn't support cursor keys)
- **Line motion: ^** go to the first non blank character in current line, **$** last character in current line
- **gg** go to beginning of file **Goto: G** go to end of file, **xxG** go to line xx

5.2. Repeating commands

In order to repeat a command several times, you may specify the number of repeats and then the command.(eg: **10h** will move the cursor 10 characters to the left)

5.3. Inserting Text

Now that you can move the cursor around, let's see how you can type text:

BE CAREFUL:

1. When in insert mode, you can delete only the text you've just insered.To delete other parts of the file, get out of insert mode (ESC) and go to*Deleting Text*

Vim specific:

If you add the following command to your ~/.vimrc or type it while running vim you will be able to backspace through the entire buffer, not just the text entered during the current "input session".
:set backspace=2

2. When you delete text that has just been inserted, see that the letters aren't erased from the screen. Instead the cursor just moves backwards and what you type will overwrite what you had "deleted". Just get used to it.

Vim specific:

This occured in original vi, and apparently in some modern clones (it was probably originally due to lack of processing power). In vim this isn't a problem -- all the characters you delete will disappear immediately, except perhaps when being used over a slow dialup line, but then the problem is the dialup line and not vi(m).

Most common insertion commands, all the following commands will get you in insert mode, but in different ways:

- **Appending (a)**: insert text *after* current cursor position
- **Inserting (i)**: insert text *before* current cursor position
- **Open New Line (o)**: open new line *below* current line and insert

All those commands have their counterparts in capital letters:

- **Appending at end of line (A)**: insert text *after* last character on the line

- **Inserting at beginning of line (I)**: insert text *before* first non blanck character on the line
- **Open New Line (O)**: open new line *above* current line and insert

N.B.: insertion (as well as most commands) can be combined with a "multiplier", in the same way than cursor commands:

10a(type your text...) will repeat 10 times the command **a**. So vi will append the text you've just typed in ten times.

5.4. Deleting Text

The delete command, as many others can be mixed with cursor commands, so that all text is deleted between the current cursor position and the new cursor position.

Deletion commands:

- **Deleting a character (x or X)**: deletes the character under (**x**) or before (**X**) the current cursor position
- **Deleting a word (dw or db)**: deletes word after (**dw**) or before (**db**) cursor
- **Deleting half a line (d^ or d$)**: deletes from the beginning of the line to the current cursor's position (**d^**) or from the cursor's position to the end of the line (**d$**)
- **Deleting a line (dd):** deletes current line

Deletion can be combined with a "multiplier", in the same way as cursor commands: **10dd** will delete 10 lines

Deleted text will be stored in the 'yank' buffer, replacing the previous one. (see *Yanking and Pasting*)

5.5. Yanking and Pasting

Copy and paste in vi is very powerful. We'll concentrate on single-buffer yanking, but be informed that, just like emacs, you can copy text into many buffers, although it is not automated.

Yanking text means 'copy'. The corresponding command is **y** and its syntax is *exactly* that same as delete's:

Yanking commands:

- **Yanking a word (yw or yb)**: yanks word after (**yw**) or before (**yb**) cursor
- **Yanking half a line (y^ or y$)**: yanks line from the current cursor position to the first non-blank character on the line (**y^**) or from current to the end of line (**y$**)
- **Yanking a line (yy):** yanks current line

WARNING: entering insert mode will erase the yank buffer.

Vim specific:

Vim doesn't erase the yank buffer when you enter insert mode.

Pasting commands:

There is only one pasting command: **p** , which means paste text in yank buffer after current cursor position.There is also its counterpart **P** for pasting before cursor.

6. Global commands

Sometimes you need to do things to the whole buffer. Instead of moving to the top, and then doing a **dG** to delete everything,you can issue global commands, based on line numbers:

syntax is

```
:line1,line2command
```

(command is appended to line2): execute command on lines line1 to line2

Examples:

```
:1,20y
```

Yanks lines 1 to 20.

```
:1,$d
```

Deletes the whole buffer. BE CAREFUL!! There $ refers to the end of the text, not to the end of the line as it did previously...

```
:%d
```

This also deletes the entire buffer. **%** is a shortcut for **1,$**.

6.1. Searching and Replacing

Searching text is done with the command **/xxx** for a forwards search or **?yyy** for a backwards search. **n** will skip to the next occurrence.When specifying **/** without argument vi will default to the argument of the last search.

Global Search and Replace:

The magic command is

```
:line1,line2s/old_string/new_string/g
```

The **/g** is optional, it means 'do the replace everytime'.If not specified, vi will replace only the first occurrence in each line.

Special ^XX characters

To search for a ^XX character, you must use **Ctrl**-**v** (**^v**) in order to disable interpretation of the Ctrl commands.

A *useful* example:

Windows (MS-DOS) text files use RETURN/LINEFEED to end every line; Mac uses only RETURN; and Unix/Linux uses only NEWLINE (which is the same as the linefeed in DOS). To use the linux programming style:

\r\n = chr(13)chr(10) = MS-DOS

\r = chr(13) = Mac

\n = chr(10) = Linux/Unix

So when displaying an msdos ascii file with vi (or with any other text editor), you will find each and every line ended by a ^M (it's character 13, aka \r, aka ENTER). When displaying a mac ascii file, you will have a single line with a ^M at what should be each end of line.

Our MSDOS text file should look like this:

```
Friday the 13th^M
^M
^M
^M
Dear Sir,^M
^M
....
```

And our mac text file should look like that

```
"Friday the 13th^M^M^MDearSir,^M^M...."
```

[The Macintosh->Unix conversion isn't easy to do with vi macros, so we'll concentrate on msdos/windows->Unix]

MS-DOS/Windows -> UNIX conversion:

In order to remove these ugly ^M, you search for them and replace them by....nothing!

So first, let's search for those weird ^M ... but, how can you search for character 'ENTER'?

By preceeding it with **^V** (**Control**-**V**). Any keystroke after **^V** is accepted literally -- that is, it won't have its usual command function, if it's something like **ESC**, **ENTER**, **^Z**, etc...

What the following command tells VI to do is to replace the first (since the **/g** option isn't set, but anyway, we only expect one) ^M on every line, with *nothing* (there is nothing between the last two slashes: **//**):

This is what you type

```
:1,$s/^V^M//
```

(where **^V** is **Control**-**V**, and **^M** is **ENTER** or **Control**-**M**)

note that VI doesn't display the ^V, so you'll only see

```
:1,$s/^M//
```

This what you actually see

And it should work...

[FYI the "text-edition task force" is working on an elegant way to convert mac ascii files to unix, but the first research campaign hasn't brought us much]

7. Miscellaneous Stuff

- **Undo: u** to undo last command.
- **Inserting a file: :read** `file_name` will insert file after current line. As a shortcut only **:r** `file_name` needs to be typed.
- **Info about current file: Ctrl**-**g** will display info about the current file
- **Redraw screen: Ctrl**-**l** will redraw the screen if it gets cluttered for some reason. This can happen, for example, if you are using vi over a poor quality dialup connection or you have a broken terminal emulator (such using Windows telnet).
- **Line numbers: :set nu** to display line numbers and **:set nonu** to not display them.

- To paste text without problems: **:set paste**
- **Useful Stuff For Writing Code or Newbiedoc HOW-TOs**

  ```
  :set expandtab softtabstop=4 shiftwidth=4
  ```

  Always leave **tabstop** set to 8. This is the rule, and it is best not to break it. If you set your tabstop to something different, then when you read something written by someone else it won't line up right and vice versa when someone else tries to read something you wrote. Instead, set **softtabstop** and **shiftwidth** and vi(m) will behave as if you had set **tabstop** without violating the "1 tab == 8 spaces" rule. If you use only these 2 options vi will insert any combination of tabs and spaces to meet your indentation specification. The **expandtab** option instructs vi to always insert spaces and never tabs. Using this option you can avoid the problems associated with using tabs because spaces are always interpreted properly by all text processors. The command above can be abbreviated as **:set et sts=4 sw=4** . The indentation level you use is up to you, but I prefer a 4-space indent.
- **Get Additional Information About A Command** **:help <command>** Will display the on-line help document for the specified command. If no command is given, then the Table Of Contents is displayed.