Chapter 1

About this Tutorial

Based on "An Introduction to Display Editing with Vi" by William Joy and Mark Horton of UC Berkeley.

Compiled and edited by aleksey@verticalsysadmin.com

http://www.verticalsysadmin.com/vi/

Created: 2 Aug 2012

Last updated: 12 Aug 2012

Chapter 2

Getting Started

Chapter 3

[Getting Started] Editing a file

Make a copy of a file you are familiar with, and run vi on this file, giving the command

% **vi** *name*

replacing name with the name of the copy file you just created. The screen should clear and the text of your file should appear on the screen.

We suggest "Alice in Wonderland", courtesy of Project Gutenberg: http://www.verticalsysadmin.com/alice.txt

Chapter 4

[Getting Started] The editor’s copy: the buffer

The editor does not directly modify the file which you are editing. Rather, the editor makes a copy of this file, in a place called the buffer, and remembers the file’s name. You do not affect the contents of the file unless and until you write the changes you make back into the original file.

Chapter 5

[Getting Started] Notational conventions

In our examples, input which must be typed as is will be presented in **bold face**.

Text which should be replaced with appropriate input will be given in *italics*.

We will represent special characters in CAPITALS.

Chapter 6

[Getting Started] Arrow keys

The editor command set is independent of the terminal you are using.

On most terminals with cursor positioning keys, these keys will also work within the editor.

If you don’t have cursor positioning keys, or even if you do, you can use the h j k and l keys as cursor positioning keys.

Chapter 7

[Getting Started] ADM3A Terminal

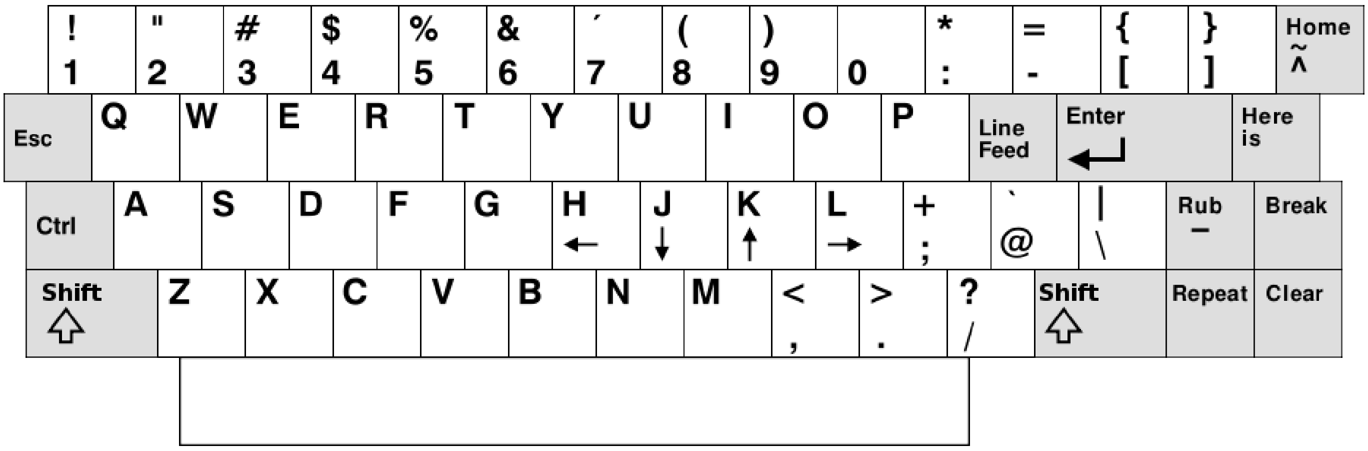
12 or 24 lines of 80 characters. First glass display (not paper!). Cursor position is addressable forward and backward; the display can be redrawn. Replaced the teletype.

Thanks to http://en.wikipedia.org/wiki/ADM3A

Chapter 8

[Getting Started] ADM3A Terminal: Keyboard

Note the Ctrl, Esc, and ~ keys.



Thanks to http://en.wikipedia.org/wiki/ADM3A

Chapter 9

[Getting Started] Special Characters: ESC

When you insert text in the file you end the text insertion with ESC.

Cancels partially formed commands.

The editor will ring the bell to indicate it is in a quiescent state (not doing anything) (i.e. there is nothing to cancel).

If you get confused, hit ESC once or twice until the editor rings the bell. Then you'll be in a known state (where the editor is not doing anything).

Chapter 10

[Getting Started] Special Characters: CR/RETURN/E TER

Used to terminate certain commands (like shell).

Chapter 11

[Getting Started] Special Characters: DEL/RUB/Ctrl-C

Originally: DEL or RUB.

Now: Ctrl-C.

Generates an interrupt, telling the editor to stop what it is doing. It is a forceful way of making the editor listen to you, or to return it to the quiescent state if you don't know or don't like what is going on. (For example, if you are searching a large file and it is taking too long.)

The editor often echoes your commands on the last line of the terminal. If the cursor is on the first position of this last line, then the editor is performing a computation, such as computing a new position in the file after a search or running a command to reformat part of the buffer. When this is happening you can stop the editor by sending an interrupt.

Chapter 12

[Getting Started] Getting out of the editor

ZZ Write the contents of the editor’s buffer back into the file you are editing, if you made any changes, and then quit from the editor.

:q! CR End the editor session (discarding all your changes).

Chapter 13

Moving around in the buffer

Chapter 14

[Moving around in the buffer] Scrolling and paging

ˆD Scroll down.

ˆU Scroll up.

Many editor commands are mnemonic and this makes them much easier to remember.

Scroll Slower

ˆE Expose one more line at the bottom of the screen, leaving the cursor

ˆY Expose one more line at the top of the screen, leaving the cursor where it is. (Y is next to U, for "up")

Page Faster

ˆF Move forward a page, keeping a couple of lines of continuity between screens.

ˆB Move forward and backward a page, keeping a couple of lines of continuity between screens.

Scrolling leaves more context, and happens more smoothly. You can continue to read the text as scrolling is taking place.

Chapter 15

[Moving around in the buffer] Searching, goto and previous context

Searching

Another way to position yourself in the file is by giving the editor a string to search for. Type the character **/** followed by a string of characters terminated by CR. The editor will position the cursor at the next occurrence of this string.

Try hitting **n** to then go to the next occurrence of this string. The character **?** will search backwards from where you are, and is otherwise like **/**.

If the search string you give the editor is not present in the file the editor will print a diagnostic on the last line of the screen, and the cursor will be returned to its initial position.

If you wish the search to match only at the beginning of a line, begin the search string with an **^**. To match only at the end of a line, end the search string with a **$**.

Chapter 16

[Moving around in the buffer] Searching, goto and previous context

Goto

The command **G**, when preceded by a number will position the cursor at that line in the file.

Example: **1G** will move the cursor to the first line of the file.

If you give G no count, then it moves to the end of the file.

If you are near the end of the file, and the last line is not at the bottom of the screen, the editor will place only the character "~" on each remaining line. This indicates that the last line in the file is on the screen; that is, the "~" lines are past the end of the file.

Exercise: Find out the state of the file you are editing by typing a **ˆG**. The editor will show you the name of the file you are editing, the number of the current line, the number of lines in the buffer, and the percentage of the way through the buffer which you are. Try doing this now, and remember the number of the line you are on. Give a **G** command to get to the end and then another **G** command to get back where you were.

Chapter 17

[Moving around in the buffer] Searching, goto and previous context

Previous context

You can also get back to a previous position by using the command `` (two back quotes). This is often more convenient than **G** because it requires no advance preparation.

Exercise: Try giving a **G** or a search with **/** or **?** and then a `` to get back to where you were. If you accidentally hit n or any command which moves you far away from a context of interest, you can quickly get back by hitting ``.

Chapter 18

[Moving around in the buffer] Moving around on the screen

h, **j**, **k**, **l** - arrow keys

+ or RETURN Advance the cursor to the next line in the buffer, at the first non-white position on the line.

- Like **+**, but goes the other way (to the previous line).

Notice that if you go off the bottom or top with these keys then the screen will scroll down (and up if possible) to bring a line at a time into view.

Vi also has commands to take you to the top, middle and bottom of the screen.

H will take you to the top (home) line on the screen. (Definition: Cursor home position: first character on the first line, or upper left corner of the screen for English language.). Try preceding it with a number as in **3H**. This will take you to the third line on the screen.

Note: Many vi commands take preceding numbers and do interesting things with them.

M takes you to the middle line on the screen.

L takes you to the last line on the screen. L also takes counts, thus **5L** will take you to the fifth line from the bottom.

Chapter 19

[Moving around in the buffer] Moving within a line

Try picking a word on some line on the screen, not the first word on the line. Move the cursor using RETURN and - to be on the line where the word is.

Try hitting the **w** key. This will advance the cursor to the next word on the line.

Try hitting the **b** key to back up words in the line.

Also try the **e** key which advances you to the end of the current word rather than to the beginning of the next word.

Also try SPACE (the space bar) which moves right one character and the BS (backspace or ˆH) key which moves left one character. The key **h** works as ˆH does and is useful if you don't have a BS key. Reminder: **l** moves right one character, too.

If the line had punctuation in it you may have noticed that that the **w** and **b** keys stopped at each group of punctuation. You can also go back and forwards words without stopping at punctuation by using **W** and **B** rather than the lower case equivalents. Think of these as bigger words. Try these on a few lines with punctuation to see how they differ from the lower case **w** and **b**.

The word keys (**w**, **b**, **e**) wrap around the end of line, rather than stopping at the end. Try moving to a word on a line below where you are by repeatedly hitting **w**.

Chapter 20

[Moving around in the buffer] Summary

SPACE	advance the cursor one position
ˆB	**backwards** to previous page
ˆD	scrolls **down** in the file
ˆE	**exposes** another line at the bottom
ˆF	**forward** to next page
ˆG	tell what is **going on**
ˆH	backspace the cursor
ˆN	**next** line, same column
ˆP	**previous** line, same column
ˆU	scrolls **up** in the file
ˆY	exposes another line at the top
+	next line, at the beginning
-	previous line, at the beginning
/	scan for a following string forwards
?	scan backwards
B	**back** a word, ignoring punctuation
G	**go to** specified line, last default
H	**home** screen line
M	**middle** screen line
L	**last** screen line
W	forward a **word**, ignoring punctuation
b	**back** a word
e	**end** of current word
n	scan for **next** instance of / or ? pattern
w	**word** after this word

Chapter 21

Viewing files

If you want to use the editor to look at a file, rather than to make changes, invoke it as **view** instead of **vi**. This will set the readonly option which will prevent you from accidently overwriting the file.

Chapter 22

Making simple changes

Chapter 23

[Making simple changes] Inserting: i and a

One of the most useful commands is the **i** (insert) command. After you type **i**, everything you type until you hit ESC is inserted into the file.

Exercise: position yourself to some word in the file and try inserting text before this word.

Exercise: find a word which can, but does not, end in an "s". Position yourself at this word and type **e** (move to end of word), then **a** for append and then **s**ESC to terminate the textual insert. This sequence of commands can be used to easily pluralize a word.

Try inserting and appending a few times to make sure you understand how this works; **i** placing text to the left of the cursor, **a** to the right.

Chapter 24

[Making simple changes] Inserting: o and O

It is often the case that you want to add new lines to the file you are editing, before or after some specific line in the file.

Exercise: Find a line where this makes sense and then give the command **o** to create a new line after the line you are on, or the command **O** to create a new line before the line you are on. After you create a new line in this way, text you type up to an ESC is inserted on the new line.

Chapter 25

Note: Editor Commands: Themes and Variations

Many related editor commands are invoked by the same letter key and differ only in that one is given by a lower case key and the other is given by an upper case key. In these cases, the upper case key often differs from the lower case key in its sense of direction, with the upper case key working backward and/or up, while the lower case key moves forward and/or down.

Chapter 26

[Making simple changes] Inserting: Inserting Multiple Lines

Whenever you are typing in text, you can give many lines of input or just a few characters. To type in more than one line of text, hit a RETURN at the middle of your input. A new line will be created for text, and you can continue to type.

Chapter 27

[Making simple changes] Inserting: Erasing while Inserting

While you are inserting new text, you can use the characters you normally use at the system command level (usually ˆH or BACKSPACE) to backspace over the last character which you typed, and the character which you use to kill input lines (such as ˆU) to erase the input you have typed on the current line. The character ˆW will erase a whole word and leave you after the space after the previous word; it is useful for quickly backing up in an insert.

Note on ˆU: it does not erase characters which you didn't insert.

Chapter 28

[Making simple changes] Making small corrections

You can make small corrections in existing text quite easily. Find a single character which is wrong or just pick any character. Use the arrow keys to find the character, or get near the character with the word motion keys and then either backspace (hit the BS key or ˆH or even just **h**) or SPACE (using the space bar) until the cursor is on the character which is wrong. If the character is not needed then hit the **x** key; this deletes the character from the file. It is analogous to the way you x out characters when you make mistakes on a typewriter (except it's not as messy).

If the character is incorrect, you can replace it with the correct character by giving the command **rc**, where *c* is replaced by the correct character.

Finally if the character which is incorrect should be replaced by more than one character, give the command **s** which substitutes a string of characters, ending with ESC, for it.

If there are a small number of characters which are wrong you can precede **s** with a count of the number of characters to be replaced.

Counts are also useful with **x** to specify the number of characters to be deleted.

Chapter 29

[Making simple changes] More corrections: operators

You already know almost enough to make changes at a higher level. All you need to know now is that the **d** key acts as a delete operator. Try the command **dw** to delete a word.

Try hitting **.** a few times. Notice that this repeats the effect of the **dw**. The command **.** repeats the last command which made a change. You can remember it by analogy with an ellipsis '. . . '.

Now try **db**. This deletes a word backwards, namely the preceding word. Try **d**SPACE. This deletes a single character, and is equivalent to the **x** command.

Another very useful operator is **c** or change. The command **cw** thus changes the text of a single word. You follow it by the replacement text ending with an ESC. Find a word which you can change to another, and try this now. Notice that the end of the text to be changed was marked with the character '$' so that you can see this as you are typing in the new material.

Chapter 30

[Making simple changes] Operating on lines

It is often the case that you want to operate on lines. Find a line which you want to delete, and type **dd**, the **d** operator twice. This will delete the line.

Try repeating the **c** operator twice; this will change a whole line, erasing its previous contents and replacing them with text you type up to an ESC.

You can delete or change more than one line by preceding the **dd** or **cc** with a count, i.e. **5dd** deletes 5 lines. You can also give a command like **dL** to delete all the lines up to and including the last line on the screen, or **d3L** to delete through the third from the bottom line. Try some commands like this now.

Note: the editor lets you know when you change a large number of lines so that you can see the extent of the change. The editor will also always tell you when a change you make affects text which you cannot see.

Chapter 31

[Making simple changes] Undoing

Now suppose that the last change which you made was incorrect; you could use the insert, delete and append commands to put the correct material back. However, since it is often the case that we regret a change or make a change incorrectly, the editor provides a **u** (undo) command to reverse the last change which you made.

Originally, the undo command only let you reverse a single change. After you make a number of changes to a line, you may decide that you would rather have the original state of the line back. The **U** command restores the current line to the state before you started changing it.

Chapter 32

[Making simple changes] Summary

SPACE	advance the cursor one position
ˆH	backspace the cursor
ˆW	erase a word during an insert
erase	your erase (such as ˆH), erases a character during an insert
kill	your kill (such as ˆU), kills the insert on this line
.	repeats the changing command
O	opens and inputs new lines, above the current
U	undoes the changes you made to the current line
a	appends text after the cursor
c	changes the object you specify to the following text
d	deletes the object you specify
i	inserts text before the cursor
o	opens and inputs new lines, below the current
u	undoes the last change

Chapter 33

Moving about; rearranging and duplicating text

Chapter 34

[Moving about; rearranging and duplicating text] Low level character motions: f and F

Now move the cursor to a line where there is a punctuation or a bracketing character such as a parenthesis or a comma or period. Try the command **f***x* where *x* is this character. This command finds the next *x* character to the right of the cursor in the current line.

Try then hitting a **;**, which finds the next instance of the same character. By using the **f** command and then a sequence of **;**'s you can often get to a particular place in a line much faster than with a sequence of word motions or SPACEs.

There is also an **F** command, which is like **f**, but searches backward. The **;** command repeats F also.

Chapter 35

[Moving about; rearranging and duplicating text] Low level character motions: t and T

When you are operating on the text in a line it is often desirable to deal with the characters up to, but not including, the first instance of a character. Try **df***x* for some *x* now and notice that the *x* character is deleted. Undo this with **u** and then try **dtx**; the **t** here stands for to, i.e. delete up to the next *x*, but not the *x*.

The command **T** is the reverse of **t**.

Chapter 36

[Moving about; rearranging and duplicating text] Low level character motions: ˆ and $

When working with the text of a single line, an ˆ moves the cursor to the first non-white position on the line, and a **$** moves it to the end of the line. Thus **$a** will append new text at the end of the current line.

Chapter 37

[Moving about; rearranging and duplicating text] Low level character motions: Control Characters

Your file may have tab (^I) characters in it. These characters are represented as a number of spaces expanding to a tab stop, where tab stops are every 8 positions. When the cursor is at a tab, it sits on the last of the several spaces which represent that tab. Try moving the cursor back and forth over tabs so you understand how this works.

On rare occasions, your file may have nonprinting characters in it. These characters are displayed in the same way they are represented in this document, that is with a two character code, the first character of which is "^". On the screen non-printing characters resemble a "^" character adjacent to another, but spacing or backspacing over the character will reveal that the two characters are, like the spaces representing a tab character, a single character.

The editor sometimes discards control characters, depending on the character and the setting of the beautify option, if you attempt to insert them in your file. You can get a control character in the file by beginning an insert and then typing a **^V** before the control character. The **^V** quotes the following character, causing it to be inserted directly into the file.

Chapter 38

[Moving about; rearranging and duplicating text] Higher level text objects

In working with a document it is often advantageous to work in terms of sentences, paragraphs, and sections.

Chapter 39

[Moving about; rearranging and duplicating text] Higher level text objects: Sentences

Sentences

The operations (and) move to the beginning of the previous and next sentences respectively. Thus the command **d)** will delete the rest of the current sentence; likewise **d(** will delete the previous sentence if you are at the beginning of the current sentence, or the current sentence up to where you are if you are not at the beginning of the current sentence.

A sentence is defined to end at a ".", "!" or "?" which is followed by either the end of a line, or by two spaces. Any number of closing ")", "]", """ and "’" characters may appear after the ".", "!" or "?" before the spaces or end of line.

The sentence command can be given counts to operate over groups of sentences.

Exercise

Try experimenting with the sentence command until you are sure how it works.

Chapter 40

[Moving about; rearranging and duplicating text] Higher level text objects: Paragraphs

Paragraphs

The operations **{** and **}** move over paragraphs.

A paragraph begins after each empty line, and also at each of a set of paragraph macros, specified by the pairs of characters in the definition of the string valued option paragraphs. The default setting for this option defines the paragraph macros of the -ms and -mm macro packages, i.e. the '.IP', '.LP', '.PP' and '.QP', '.P' and '.LI' macros.

Each paragraph boundary is also a sentence boundary.

The paragraph commands can be given counts to operate over groups of sentences and paragraphs.

Exercise

Try experimenting with the paragraph command until you are sure how it works.

Chapter 41

[Moving about; rearranging and duplicating text] Higher level text objects: Sections

Sections

The operations [[and]] move over sections. Sections in the editor begin after each macro in the sections option, normally '.NH', '.SH', '.H' and '.HU', and each line with a formfeed ^L in the first column. Section boundaries are always line and paragraph boundaries also.

Small Window

The section commands interpret a preceding count as a different window size in which to redraw the screen at the new location, and this window size is the base size for newly drawn windows until another size is specified. This is very useful if you are on a slow terminal and are looking for a particular section. You can give the first section command a small count to then see each successive section heading in a small window.

Exercise

Try looking through a large document using the section commands.

Chapter 42

[Moving about; rearranging and duplicating text] Rearranging and duplicating text

The editor has a single unnamed buffer where the last deleted or changed away text is saved, and a set of named buffers **a-z** which you can use to save copies of text and to move text around in your file and between files.

The operator **y** yanks a copy of the object which follows into the unnamed buffer. If preceded by a buffer name, **"***x***y**, where *x* here is replaced by a letter **a-z**, it places the text in the named buffer. The text can then be put back in the file with the commands **p** and **P**; **p** puts the text after or below the cursor, while **P** puts the text before or above the cursor.

If the text which you yank forms a part of a line, or is an object such as a sentence which partially spans more than one line, then when you put the text back, it will be placed after the cursor (or before if you use **P**). If the yanked text forms whole lines, they will be put back as whole lines, without changing the current line. In this case, the put acts much like a **o** or **O** command.

Try the command **YP**. This makes a copy of the current line and leaves you on this copy, which is placed before the current line. The command **Y** is a convenient abbreviation for **yy**. The command **Yp** will also make a copy of the current line, and place it after the current line. You can give **Y** a count of lines to yank, and thus duplicate several lines; try **3YP**.

To move text within the buffer, you need to delete it in one place, and put it back in another. You can precede a delete operation by the name of a buffer in which the text is to be stored as in **"a5dd** deleting 5 lines into the named buffer *a*. You can then move the cursor to the eventual resting place of the these lines and do a **"ap** or **"aP** to put them back. In fact, you can switch and edit another file before you put the lines back, by giving a command of the form **:e** *name*CR where *name* is the name of the other file you want to edit. You will have to write back the contents of the current editor buffer (or discard them) if you have made changes before the editor will let you switch to the other file. An ordinary delete command saves the text in the unnamed buffer, so that an ordinary put can move it elsewhere. However, the unnamed buffer is lost when you change files, so to move text from one file to another you should use an unnamed buffer.

Chapter 43

[Moving about; rearranging and duplicating text] Summary

ˆ	first non-white on line
$	end of line
)	forward sentence
}	forward paragraph
]]	forward section
(	backward sentence
{	backward paragraph
[[	backward section
fx	find x forward in line
p	put text back, after cursor or below current line
y	yank operator, for copies and moves
tx	up to x forward, for operators
Fx	f backward in line
P	put text back, before cursor or above current line
Tx	t backward in line

Chapter 44

High level commands

Chapter 45

[High level commands] Writing, quitting, editing new files

So far we have seen how to enter vi and to write out our file using either **ZZ** or **:w**CR. The first exits from the editor, (writing if changes were made), the second writes and stays in the editor.

If you have changed the editor's copy of the file but do not wish to save your changes, either because you messed up the file or decided that the changes are not an improvement to the file, then you can give the command **:q!**CR to quit from the editor without writing the changes.

You can also reedit the same file (starting over) by giving the command **:e!**CR. These commands should be used only rarely, and with caution, as it is not possible to recover the changes you have made after you discard them in this manner.

You can edit a different file without leaving the editor by giving the command **:e** *name*CR. If you have not written out your file before you try to do this, then the editor will tell you this, and delay editing the other file. You can then give the command **:w**CR to save your work and then the **:e** *name*CR command again, or carefully give the command **:e!** *name*CR, which edits the other file discarding the changes you have made to the current file.

Note: To have the editor automatically save changes, include set autowrite in your EXINIT, and use :n instead of :e.

Chapter 46

[High level commands] Escaping to a shell

You can get to a shell to execute a single command by giving a vi command of the form **:!** *cmd*CR. The system will run the single command *cmd* and when the command finishes, the editor will ask you to hit a RETURN to continue. When you have finished looking at the output on the screen, you should hit RETURN and the editor will clear the screen and redraw it. You can then continue editing. You can also give another **:** command when it asks you for a RETURN; in this case the screen will not be redrawn.

If you wish to execute more than one command in the shell, then you can give the command **:sh**CR. This will give you a new shell, and when you finish with the shell, ending it by typing a **ˆD**, the editor will clear the screen and continue.

On systems which support it, **ˆZ** will suspend the editor and return to the (top level) shell. When the editor is resumed, the screen will be redrawn.

Chapter 47

[High level commands] Marking and returning

The command `` returned to the previous place after a motion of the cursor by a command such as **/**, **?** or **G**. You can also mark lines in the file with single letter tags and return to these marks later by naming the tags. Try marking the current line with the command **m***x*, where you should pick some letter for *x*, say "a". Then move the cursor to a different line (any way you like) and hit **`a**. The cursor will return to the place which you marked. Marks last only until you edit another file.

When using operators such as **d** and referring to marked lines, it is often desirable to delete whole lines rather than deleting to the exact position in the line marked by **m**. In this case you can use the form **'x** rather than **`x**.

Used without an operator, **'x** (single-quote x) will move to the first non-white character of the marked line; similarly **''** (double single-quote) moves to the first non-white character of the line containing the previous context mark `` (double back-quote).

Chapter 48

[High level commands] Adjusting the screen: re-drawing the screen

If the screen image is messed up because of a transmission error to your terminal, or because some program other than the editor wrote output to your terminal, you can hit a **ˆL**, the ASCII form-feed character, to cause the screen to be refreshed.

Chapter 49

[High level commands] Adjusting the screen: positioning a line

If you wish to place a certain line on the screen at the top middle or bottom of the screen, you can position the cursor to that line, and then give a **z** command:

z RETURN if you want the line to appear at the top of the window,

z. if you want it at the center,

z- if you want it at the bottom.

Chapter 50

Legacy/ubiquity of vi

Many other programs either can kick into vi when appropriate, or (can) use the same navigation keys, or some logical mapping that somehow seems appropriate. Among them bash, zsh, screen, emacs, almost anything using readline, lynx, firefox/vimperator/pentadactyl etc., etc. etc. GMail is another modern example.

Chapter 51

Last Slide

References: http://www.verticalsysadmin.com/vi

Survey:

1. What did you get out of this training?
2. What was best about it?
3. What should be improved?

Please email aleksey at verticalsysadmin.com