# VI Editor

EE1356 Introduction to Information Systems

*Learning the vi Editor by Linda Lamb, 5th Edition, Oreilly 1994.*

## Why Learn VI?

- For many users, vi is an arcane and ancient beast that is hard to understand and control, but…
- vi is THE ubiquitous editor of UNIX
- VIM (Vi Improved): modern and much improved implementation of vi
  - www.vim.org
  - There is a windows version of vim.
- Powerful, flexible and efficient by design
  - Linus works on Linux with vi and xterm.
  - you might want to check emacs, too.

## What about Text Editor?

- Text file format is the only truly interchangeable format.
  - Most information can be presented without any formating (i.e. word processors)
    - Work on what you think, not on what words look like!
  - Everyone can read what you write
- You need a powerful text editor for programming
  - C/C++
  - HDL
  - Spice
  - Matlab
  - …

## vi and Keyboards

- vi was invented in the age of limited keys of a keyboard
  - vi only assumes you have a keyboard that you can type English characters (plus a few control keys).
  - Of course, no mouse (but vim can work with a mouse happily).
- You can use vi without hands leaving the keyboard’s core typing area.
  - This can tremendously improve the speed of typing.

## The Tale of Two Modes

- To use the limited keys, vi has two operating modes:
  - command mode
  - insert mode
- Command mode
  - As soon as you enter a file, you are in command mode, and the editor is waiting for you to enter a command.
  - Commands enable you to move anywhere in the file, to perform edits, or to enter insert mode to add new text.
  - The things you typed are commands!
- Insert mode
  - The things you typed are characters to be saved in a file.
  - As a typewriter.
  - Switch back to the command mode with [ESC]

## More on the Command Mode

- One or two characters are used for the basic commands.
  - For example:
  - i *(means "insert")*
  - cw *(means "change word")*
- Using letters as commands, you can edit a file with great speed
- Most of the commands can be remembered by the letter that performs them
- *vi* commands:
  - Are case-sensitive.
  - Are not shown on the screen when you type them.
  - Do not require a [RETURN] after the command.
- a group of commands that echo on the bottom line of the screen.
  - The slash (/) and the question mark (?) begin search commands.
  - A colon (:) begins all *ex* commands.

## Open a File

- $ **vi** [*filename*]
  - Example: $ **vi practice**
  - Since this is a new file, the buffer is empty and the screen appears as follows:
  - The tildes (~) down the left-hand column of the screen indicate that there is no text in the file, not even blank lines.

```
~
~
~
"practice" [New file].
```

## Close a File

- To save the file, first check that you are in command mode by pressing [ESC] and then enter ZZ.
- Or use ex command
  - :q *(meaning quit, vi will check if you have something in the buffer not saved)*
  - :q! *(meaning quit without saving)*
    - note the ! means *"don't ask, just do it"*
  - :wq *(meaning write to disk and quit; same as ZZ)*

## Entering Insert Mode

- There are several ways to tell *vi* that you want to begin insert mode.
  - One of the most common is to press i.
  - The i doesn't appear on the screen, but after you press it, whatever you type *will* appear on the screen and will be entered into the buffer.
  - The cursor marks the current insertion point.
  - To tell *vi* that you want to stop inserting text, press [ESC].
  - Pressing [ESC] moves the cursor back one space (so that it is on the last character you typed) and returns *vi* to command mode.
- For example, you have opened a new file and want to insert the word "introduction". If you type iintroduction, what appears on the screen is:

```
introduction
```

- Note that All keystrokes made after the insert command are considered text until you press [ESC].
  - To correct a mistake while in insert mode, backspace and type over the error.
  - Note that you can't use the backspace key to back up beyond the point where you entered insert mode. *(This limitation is lifted in vim)*

# Moving the Cursor

- **Single Movements**
- h
  - left, one space.
- j
  - down, one line.
- k
  - up, one line.
- l
  - right, one space.
- Note that you cannot move beyond the boundary. For example, you cannot use h or l to wrap around to the previous or next line (there will be a beep sound).

## Numeric Arguments

- You can precede movement commands with numbers
- For example:
  - 4l mean llll (four times l), so the cursor will move to the left four characters.
- This can be applied to most commands.

## More Movement Commands

- 0
  - Move to beginning of line.
- $
  - Move to end of line.
- w
  - moves the cursor forward one word at a time, counting symbols and punctuation as equivalent to words. The line below shows cursor movement by w:

    cursor, delete lines, insert characters,
- W
  - move by word, not counting symbols and punctuation,. Cursor movement using W looks like this:

    cursor, delete lines, insert characters,
- To move backward by word, use the b command. Capital B allows you to move backward by word, not counting punctuation.

## A Few Simple Editing Commands

- **Changing Text (c)**
- Replace Text (r)
- **Changing Case (~)**
- **Delete Text (d)**
- Moving Text (dd and p)
- Copying Text (yy and p)
- Repeat and Undo (. and u)
- Joining two lines (J)

## Changing Text

- c can be used to change text from the cursor:
- cw
  - to the end of a word.
- c2b
  - back two words.
- c$
  - to the end of line.
- c0
  - to the beginning of line.

## vi Command General Form

- (*command*)(number)(*text object*)
- (number)(*command)text object*)
- *command* is the change command c, and *text object* is a movement command (you don't type the parentheses).
- For example,
- d2w or 2dw is a command to delete two words

## Replacing Text

- r
  - replaces a single character with another single character.
  - You do *not* have to press [ESC] to return to command mode after making the edit.
  - For example,
  - rW (at Pith)

```
Pith a screen editor you can scroll the page,
```

```
With a screen editor you can scroll the page,
```

## Changing Cases

- ~
  - change a lowercase letter to uppercase, or an uppercase letter to lowercase.
  - No numeric arguments allowed
  - But in vim, you can combine “marking” and ~.

## Deleting Texts

- dw
  - Delete a word
- dd
  - Delete a line
- d$
  - Delete to the end of a line
- x
  - delete a single character

## Moving Text

- "cut and paste."
  - delete (d) and put back (p)
- p puts the text that is in the buffer *after* the cursor position.
- P puts the text *before* the cursor
- dd followed by p
  - delete a line and put it back after the cursor
- xp
  - swap two characters

## Copying Text

- "copy and paste."
  - yank (y) and put back (p)
- yw
  - copy a word
- yy
  - copy a line
- y$
  - copy to the end of a line
- Note that in vim, you can make complex text selection by “marking” texts
  - v, V, CTRL-v
  - And use command to operate on the selected texts (e.g., d, y, etc)

## Repeat and Undo

- . (the dot character)
  - repeat the last command
- u
  - undo the last operation
  - original vi has only one undo available
  - vim has unlimited undo level

## Joining Two Lines

- J
  - merge two lines into one
  - will remove the newline character between two lines.

## More Ways to Enter Insert Mode

- A
  - Append text to end of current line.
- I
  - Insert text at beginning of line.
- o
  - Open blank line below cursor for text.
- O
  - Open blank line above cursor for text.
- s
  - Delete character at cursor and substitute text.
- S
  - Delete line and substitute text.
- R
  - Overstrike existing characters with new characters.

## More Movement Commands

- **Scrolling the Screen**
- ^F
  - Scroll forward one screen.
- ^B
  - Scroll backward one screen.
- ^D
  - Scroll forward half screen (down).
- ^U
  - Scroll backward half screen (up).

## Repositioning the Screen with z

- z
  - to scroll the screen up or down, but the cursor to remain on the line where you left it.
- z[RETURN]
  - Move current line to top of screen and scroll.
- z.
  - Move current line to center of screen and scroll.
- z-
  - Move current line to bottom of screen and scroll.

## Movement Within a Screen

- You can also keep your current screen, or view of the file, and move around within the screen using:
- H
  - Move to home  - top line on screen.
- M
  - Move to middle line on screen.
- L
  - Move to last line on screen.
- *n*H
  - Move to *n* lines below top line.
- *n*L
  - Move to *n* lines above last line.

## Movement by Line

- [RETURN]
  - Move to first character of next line.
- +
  - Move to first character of next line.
- -
  - Move to first character of previous line
- ^
  - Move to first nonblank character of current line.
- *n*|
  - Move to column *n* of current line

## Movement by Text Blocks

- e
  - Move to end of word.
- E
  - Move to end of word (ignore punctuation).
- (
  - Move to beginning of current sentence.
    - To find the end of a sentence, *vi* looks for one of the punctuation marks ? . !
- )
  - Move to beginning of next sentence.
- {
  - Move to beginning of current paragraph.
    - A paragraph is defined as text up to the next blank line
- }
  - Move to beginning of next paragraph.

## Movement by Searches

- */pattern*
  - In command mode, type “/” will put you in the search mode.
  - The *pattern* can be words or regular expressions.
- *?pattern*
  - To search backward
- n
  - *meaning next*
  - Repeat search in same direction.
- N
  - Repeat search in opposite direction.

## Show the Line Number

- [CTRL-G]
  - causes the following to be displayed at the bottom of your screen: the current line number, the total number of lines in the file, and what percentage of the total the present line number represents.
  - For example, for the file *practice*, [CTRL-G] might display:

```
"practice" line 3 of 6 --50%--
```

- You can use ex command to display the line numbers on the screen
  - :set number (or :set nu)

## Move to A Line Number

- G (go to)
  - uses a line number as a numeric argument and moves directly to that line.
  - For instance, 44G moves the cursor to the beginning of line 44.
  - G without a line number moves the cursor to the last line of the file.
- `` (two back quotes) returns you to your original position
  - the position where you issued the last G command
  - If you have issued a search command (/ or ?), ` ` will return the cursor to when you started the search
  - If you have made an edit, ` ` will return the cursor to the site of your last edit.

## Using Deleted Buffers

- **Recovering Deletions**
- <"> (double quote)+number
  - identify the deleted text in the buffer
  - Example
  - "2p
    - Will put back the second to last deleted texts
- "1pu.u.u …
  - Try to put back the last deleted text (“1p) and undo (u, will cancel the text) and then redo (.==“1p) put back the last deleted…

## Yanking to Named Buffers

- “[a-z]
  - Represent 26 named buffers
  - Examples
  - "dyy *Yank current line into buffer d.*
  - "a7yy *Yank next seven lines into buffer a.*
  - "dP *Put the contents of buffer d before cursor.*
  - "ap *Put the contents of buffer a after cursor.*
- “[A-Z]
  - Yanked or deleted text will be appended to the current contents of that buffer.
  - allows you to be selective in what you move or copy.
  - "Zy)
    - Add the next sentence to buffer z

## Bookmark

- m*x*
  - Marks current position with *x* (*x* can be any letter).
- '*x*
  - (apostrophe) Moves cursor to first character of line marked by *x*.
- `` ` ``*x*
  - (backquote) Moves cursor to character marked by *x*.
- ``` `` ```
  - (backquotes) Returns to exact position of previous mark or context after a move.
- ''
  - (apostrophes) Returns to the beginning of the line of the previous mark or context. Place markers are set only during the current *vi* session; they are not stored in the file.

## Ex Commands

- :n
  - go to nth line and print it
- :n,m
  - print line n to m
- :s/pattern1/pattern2
  - substitute patttern1 with pattern2
- :3,18d
  - Delete lines 3 through 18.
- :160,224m23
  - Move lines 160 through 244 to follow line 23. (Like delete and put in *vi*.)
- :23,29co100
  - Copy lines 23 through 29 and put after line 100. (Like yank and put in *vi*.)

## Ex Line Address Symbols

- A dot (.) stands for the current line
- $ stands for the last line of the file
- % stands for every line in the file
- Examples
  - :.,$d
    - Delete from current line to end of file.
  - :20,.m$
    - Move from line 20 through the current line to the end of the file.
  - :%d
    - Delete all the lines in a file.
  - :%t$
    - Copy all lines and place them at the end of the file (making a consecutive duplicate).

## Ex Incremental Line Addresses

- Examples
  - :.,.+20d
    - Delete from current line through the next 20 lines.
  - :226,$m.-2
    - Move lines 226 through the end of the file to two lines above the current line
  - :.,+20#
    - Display line numbers from the current line to 20 lines further on in the file.
- In fact, you don't need to type the dot (.) when you use + or -

## Ex Addressing by Search Patterns

- :/*pattern*/d
  - Delete the next line containing *pattern*.
- :/*pattern*/+d
  - Delete the line *below* the next line containing *pattern*. (You could also use +1 instead of + alone.)
- :/*pattern1*/,/*pattern2*/d
  - Delete from the first line containing *pattern1* through the first line containing *pattern2*.
- :.,/*pattern*/m23
  - Take text from current line (.) through the first line containing *pattern* and put after line 23.

## Global Searches

- :g
  - search for a pattern and display all lines containing the pattern when it finds them.
- :g! (or its synonym :v)
  - search for all lines that do *not* contain *pattern*.
- :g/*pattern*
  - Finds (moves to) the last occurrence of *pattern* in the file.
- :g/*pattern*/p
  - Finds and displays all lines in the file containing *pattern*.
- :g!/*pattern*/nu
  - Finds and displays all lines in the file that don't contain *pattern*; also displays line number for each line found.
- :60,124g/*pattern*/p
  - Finds and displays lines between lines 60 and 124 containing *pattern*.

## Saving and Quitting in Ex

- :w
  - Writes (saves) the buffer to the file but does not exit.
- :q
  - Quits the file (and returns to the UNIX prompt).
- :wq
  - Both writes and quits the file.
- :w practice.new
  - Save to another file named practice.new
- :.,600w *newfile*
  - Saves from the current line to line 600 in *newfile*.
- :340,$w >>*newfile*
  - Append texts from line 340 to the end of newfile
- :r afile
  - Read the contents of afile to the current cursor position.

## Editing Multiple Files

- vi file1 file2
- Invokes *file1* first. After you have finished editing the first file, the *ex* command :w writes (saves) *file1* and :n calls in the next file (*file2*).
  - In vim, there are :bn (next buffer) and :bp (previous buffer)
  - Note that you can yank texts in buffers and put them back in the other file.
- :e file3
  - Edit a new file, file3
- :e!
  - discards your edits and returns to the last saved version of the current file.

## Ex Global Replacement

- :s/*old*/*new*/
  - Changes the first occurrence of the pattern *old* to *new* on the current line.
- :s/*old*/*new*/g
  - Changes *every* occurrence of *old* to *new* on the current line
- :1,$s/*old*/*new*/g
- :%s/*old*/*new*/g
  - Change every occurrence of *old* to *new* within the entire file

# Ex Substitution Confirmation

- :1,30s/his/the/gc
  - display the entire line where the string has been located, and the string will be marked by a series of carets (^^^^).
  - to make the replacement, y (for yes) and [RETURN]
  - don't want to make a change, simply press [RETURN].
- How to do this in vi
  - /his Search for *which*.
  - cwthe [ESC] Change to *that*.
  - n Repeat search.
  - .Repeat change (if appropriate).

## Ex Context Sensitive Replacement

- You can use a search pattern to locate the lines for substitution
  - :g/*pattern*/s/*old*/*new*/g
  - if pattern==old
    - :g/*pattern*/s//*new*/g
- Examples
  - :g/editer/s//editor/g
  - :%s/editer/editor/g
  - Are the same

# Pattern Matching in vi (I)

- .
  - Matches any *single* character except a new line.
- *
  - Matches any number (or none) of characters that immediately precedes it.
- ^
  - Requires that the following regular expression be found at the beginning of the line
  - ^Part matches *Part* when it occurs at the beginning of a line
- $
  - Requires that the preceding regular expression be found at the end of the line
  - here:$.
- \
  - Treats the following special character as an ordinary character.

# Pattern Matching in vi (II)

- [ ]
  - Matches any *one* of the characters enclosed between the brackets
  - [A-Z]
- [^]
  - the brackets will match any one character *not* in the list.
- \(  \)
  - Saves the pattern enclosed between \( and \) into a special holding space or "hold buffer."
  - :%s/\(That\) or \(this\)/\2 or \1/
- \<  \>
  - Matches characters at the beginning (\<) or at the end (\>) of a word.
  - \<ac will match only words that begin with *ac*, such as *action*.
- ~
  - Matches whatever regular expression was used in the *last* search.

# Pattern Matching Examples

- To substitute the word *child* with the word *children* throughout a file.
- :%s/child/children/g
  - There is *childrenish* which is from unintentionally matched word *childish*. With :e!, you try again:
- :%s/child_/children /g
  - But this command misses the occurrences *child.*, *child,*, *child:* and so on. so you come upon the solution:
- :%s/child[ ,.;:!?]/children[ ,.;:!?]/g
  - but you've ended up with a bunch of punctuation marks after every occurrence of *children*. You need to save the space and punctuation marks inside a \( and \). Then you can "replay" them with a \1. Here's the next attempt:
- :%s/child\([ ,.;:!?]\)/children\1/g
  - When the search matches a character inside the \( and \), the \1 on the right-hand side restores the same character, but this is still very crumbersome.
- Here's the substitution command you should use:
- :%s/\<child\>/children/g

# Customizing vi

- :set *option*
  - turn on *option*
- :set no*option*
  - turn off *option*
- There are many options in vi
  - :set all
  - to view all options
  - you can use :h *option*
- For setting up regular options, use .exrc
  - VIM has a .vimrc
  - Simply write your own set options commands in the file

## Executing UNIX Commands within VI

- :!*command*
  - will temporarily execute a shell and *command*
- Examples
  - :!date
  - :r !date

## Word Abbreviation

- :ab *abbr phrase*
  - *abbr* is an abbreviation for the specified *phrase*.
- :ab nthu National Tsing Hua University
  - writing “nthu” in insert mode will actually give the full name
- :unab *abbr*
  - cancel the abbreviation
- :ab
  - list your currently defined abbreviations

## Command Alias

- :map *x sequence*
  - Define character *x* as a *sequence* of editing commands.
- :unmap *x*
  - Disable the *sequence* defined for *x*.
- :map
  - List the characters that are currently mapped.
- Example
- :map v dwelp
  - enables you to reverse the order of two words with “v”
- Mapping CTRL-x  (x are characters)
  - Type CTRL-V CTRL-x to write the CTRL-x on the console.
- There are many useful mapped commands available on the internet, especially for VIM.

## Features For Programmers

- :set autoindent
  - when you indent a line with spaces or tabs, the following lines will automatically be indented by the same amount.
- :set tabstop=4
  - for <TAB>
- :set shiftwidth=4
  - >> (<<)
    - to move lines towards left (right) with shiftwidth
- :set list
  - Display a tab appears as the control character ^I and an end-of-line appears as a $.
- %
  - Find the matched pair of brackets.
- ctags
  - A command to generated index for all names in the C/C++ codes.
  - CRTL-[ to find the function definition
  - CTRL-T to return to the original files