'(emacs ism)

ChillarAnand

'(emacs ism)

ChillarAnand

This project can be followed at:

https://www.penflip.com/chillaranand/emacsism

Contents

1 Prologue

Emacs is originally written in 1976. It has been almost 60 years since it came into picture and yet none of the latest & advanced softwares weren't able to replace it.

There is something magnificient about Emacs by which it is able to draw users to use it, develop it further and form a cult around it through out all these years. It is one of the longest lived programming applications in the history of computers.

You might have read a lot of flattery about Emacs. If You try and learn Emacs for a while(may be few weeks to few months), You will realize that the flattery is actually true and everything You have read is very less when compared to what Emacs is actually capable of.

2 Why This Book?

I was using PHP for server side programming & I was looking for a good IDE. I tried Sublime text, Eclipse & Vim for a while and settled for Vim.

After a year or so, I switched from PHP to Python. I found quite a few people in Python community were using something called Emacs.

I started digging about Emacs and came across editor war. There were few articles which supported Vim and mentioned why Vim is better than Emacs in many aspects. So I was thought Vim is much better than Emacs. Even when I looked at statistics Vim is quite popular than Emacs. In addition to that I have already spent a year learning & customising Vim. So I wasn't ready to switch.

Even though Emacs had a small user base, it had a cult following. In addition to that there were some good programmers in Python community who were using Emacs and they were recommending to use it. So I decided to try it out for a few weeks and figure out myself what's so great about Emacs.

For a couple of weeks I have used it almost like a notepad. I started learning elisp & customising Emacs little bit. After a couple of months I was surprised when I realized that I was spending almost all the time in Emacs itself. Then I started wondering what made Emacs such a great text editor and understanding fundamentals(not basics) of Emacs.

This little book is created to share those things. This book

- Explains fundamentals of Emacs & how those fundamentals made Emacs different from other softwares.
- Debunk few popular myths about Emacs like `Emacs users use a lot of modifier keys like CTRL, ALT`.

Explaining how to use Emacs itself is beyond the scope of this book.

3 Cautions

About Emacs

If You are a programmer and if You are looking for a piece of software which has features like syntax highligthing, autocompletion, debuggers e.t.c., which help to write & execute code, You can stop here and go for an IDE.

If You are not satisfied with that, if You need something much simpler yet more powerful than IDE's and if You are looking for a divine editing experience, Emacs might help You!

I see Emacs as an inward spiral. Once You start learning Emacs and realize the power of Emacs, You start spending more and more time within Emacs and it gets impossible to get out of Emacs. You have been warned!!

About Book

I am neither a professional writer nor an expert in Emacs. I just tried to put my thoughts about Emacs on a paper. When I

got struct in explaining something, I copied some ideas from few blog posts. Any mistakes are my own.

4 Lisp

Emacs is written in a programming language called Emacs-lisp(a variant of lisp). There were some attempts to port Emacs to other languages. But none of them were successful because those programming languages weren't as powerful as lisp there by loosing all the power of Emacs. So, before diving into the Emacs, it is good to have a brief introduction to Lisp features and see how it differs from traditional languages (like C, Python, Perl e.t.c).

Expressions

Lisp is an expression-oriented programming language which means almost everything is an expression. Hello world in Lisp looks like this

```
(message "Hello, World!")
```

To add `x` and greatest of `y`, `z`, we can use a **single** expression to do it.

```
(if (> y z) (+ x y) (+ x z))
```

That might look bit odd because lisp uses prefix notation.

In a traditional programming language(like Python) it is not possible to do that in a single expression like this.

```
x + (if y > z: y else: z)
```

Because those languages distinguish statements and expressions. A conditional statement(`if y>z: y else: z`) can't be used inside an expression(`x + something`).

What's more interesting is, in Lisp, it is possible to compose same expressions in many ways.

The above expression

```
(if (> y z) (+ x y) (+ x z))
```

can also be written as

```
(+ x (if (> y z) y z))
```

Lisp by making everything an expression expands possibilites to combine parts of language in unusual ways.

Evaluation

Syntax is the set of rules which define the combinations of symbols that are considered to be a correctly structured document. Programming languages like C, Python and Markup languages like HTML, XML, Tex have their own syntax

Lisp hardly has any syntax at all. Everything is an expression. Every expression can be evaluated. If you don't want it evaluated, you put a quote on it. That's it.

Availability

Traditional programming language have read time, compile time & run time. In Lisp, the whole language is always available. There is no real distinction between read-time, compile-time, and runtime. You can compile or run code while reading, read or run code while compiling, and read or compile code at runtime.

Extensibility

Macro is a piece of code which treats code as data and generates new code. Macros will be used by compiler to generate code that will be compiled just before its evaluation. Most programming languages are not capable of this manipulation, they advise not use this kind of stuff. For example, consider a simple macro in C language.

```
#define square(x)      x*x
```

It is almost impossible to get it work in various contexts.

Lisp never breaks any of macros in any situation.

Imagine how difficult it is to add new syntax to traditional languages like Python. We need to modify grammar, AST and then compile it to get new syntax. With lisp it is as easy as evaluating a macro.

These features set Lisp apart from traditional languages.

References

http://gigamonkeys.com/book/

http://www.paulgraham.com/diff.html

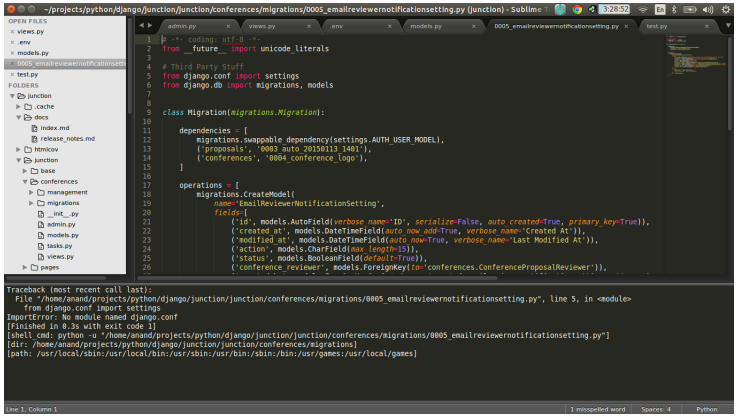
http://practicaltypography.com/why-racket-why-lisp.html

5 Why Emacs?

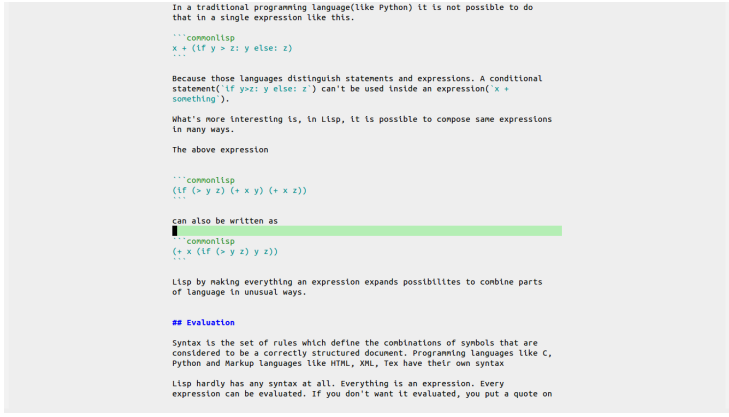
In this chapter, lets see some of Emacs features by comparing Emacs with other editors/IDE's. This is just to have better understanding of Emacs fundamental features. However note that Emacs is not just a text editor but it is an extensible computing environment.

Real Estate

If You look at typical IDE, its layout looks like this.



Some IDE's have more components than this. Here is the screenshot of Emacs as I am writing this book.



Pretty neat, huh? Since it is completely distraction less, You can visually concentrate on getting things done :D

Buffers

Most IDE's use tabs to hold files when a file is opened. One serious disadvantage is it doesn't scale well. If You open more than 10 files, the bar that shows file names will get clogged. It becomes quite difficult to switch between files.

Emacs uses something called "buffer" to hold files/text. Every time a file is opened or a process is started, a buffer is used to hold text. Buffers can scale to very large extent. You can open thousands of files and You can switch to any file with just few key presses.

Extensibility

Extensibility lies at the heart of Emacs. What I mean by extensibility is

- End user should be able to change any piece of editor
- Editor should have the ability to add new functionality, without affecting its internal structure and data flow

For example, most editors have fuzzy search to find files in a project. Fuzzy search is awesome. If You have several projects, it would be great if You could fuzzy search project names also. If You are not using Emacs, try to implement this in Your editor and see how it goes.

In Emacs, it is as easy as calling an existing function with a different variable. Same thing applies to most other features also. With few lines of code, You can customize/extend any piece of it the way You want it.

Composability

Composability refers to inter-relationships of components. In Emacs, every component can be selected and assembled in various combinations as per user requirements.

If we take previous example of fuzzy search, most editors combine two pieces to achieve it. They combine `fuzzy search` code with `open file` code. So You can `search` for a file and then press enter, it `opens` the file. Now after searching for a file, instead of `opening` the file, lets say You have to `rename` it or `delete` it. Its not possible.

In Emacs, once You `searched` for a file, You can perform any action on it. You can open it, rename it, delete it or You can define Your own function to do something else.

This composability helps a lot to either modify existing features or define new features with very few lines of code.

DWIM

Emacs has quite a few DWIM(Do What I Mean) functions. Instead of blindly executing user's input, these functions will try to do the right thing by programmatically responding to past actions and/or document state. This reduces number of keystrokes a user has to use for a particular task.

A simple example is `comment-dwim-2`[1]. This function based on past cursor movement & context, figures out whether user is trying to comment a piece of code or inserting an inline comment or uncommenting a piece of code and responds accordingly.

[1] http://www.Emacswiki.org/Emacs/key-chord.el

Replication

To customize an IDE, You have to click menus/checkboxes. If You are using IDE, You might also need terminal or terminal multiplexer. You might have some custom configuration for terminal tools also. You might even need some other third party tools for Your development environment.

Consider some situation where You have to

- Reinstall Your OS
- Use multiple machines for development
- Use friends system for some quick bug fixing

In these situations, You need a lot of time just to replicate Your development environment.

If You are using emacs, You will be using it for most of the tasks. To customize Emacs, You have to write some code(which is a good thing) instead of clicking menus/checkboxes. You can keep all Your config in dot files and use a version control to manage it.

In above situations I have mentioned, replicating Your environment is as easy as cloning Your Emacs config and restarting Emacs. You can replicate Your development environment in just few minutes.

6 Why Editor War Is Stupid

Both Emacs and Vim has cult following and they are the longest lived programming applications of all time. Both groups insist that their editor is best. This has led to many flame wars and lots of jokes about both.

Vim

Vim is a superior **text editor** and is far better than many editors out there. Most of the systems have Vim pre-installed and You can use Vim with same commands on any system. You can quickly open a file, make bunch of changes and close it.

Emacs

Eventhough Emacs is commonly known as text editor, it is somewhat misleading. Emacs is an **extensible computing environment**. So You can use Emacs as

- Mail Client

- Document Viewer
- Twitter Client
- Irc Client
- Web Browser
- Video Editor
- Blog Editor
- Paint(Yes, You can draw pics)
- Spreadsheet
- Todo List
- Wiki
- Translator
- Terminal Multiplexer
- SSH Client
- Slides Editor
- Stack Exchange client
- Playing Games
- Version Control Interface
- Web server
- Lisp Interpreter
- SQL client
- Writing & Debugging code

and thousands of other things.

Basically, You will keep it open 24/7 and live inside it.

I think comparing both of them is kind of stupid. If we want to compare two things, they have to be atleast in the same category. There is no point in comparing a chair and a projector.

7 Why Use Emacs For Everything?

As explained in previous chapter, Emacs can be used for all sorts of things. There are many advantages of using Emacs everywhere. Lets take a look at some examples.

1. Write a python script to update table in MySQL.

Without Emacs: For writing Python code, I will be using IDE. Sine I dont know how to update table in MySQL, I have to open browser, go to stackoverflow, search for it and find relevant code. Then I have to open a terminal, create a mysql process and test the query. Once I know that it is working fine, I have to go back to editor and use that query in script.

With Emacs: Emacs has stackexchange client. So I can search for it in stackoverflow from Emacs. When I get the results, I can select relevant code and then send that directly to MySQL process

which is running in Emacs. If that code works, I can switch back to python script and paste it.

There is no need to switch between various applications as I can do most of the things in Emacs.

2. Play the dark knight movie.

Lets us assume movie is located in `/home/anand/videos/movies/english/dark-`

Without Emacs: First I have to open a file manager. By default, it will show home folder `/home/anand`. Now I have to go to `videos`, then `movies`, then `english`, then `dark-knight` and double click the video file. This video will be opened in some media player.

With Emacs: Since I am using singularity, if I just press `fj drk`, Emacs shows me with the relevant video file `dark-knight.mkv`. Now if I just press enter, it will open in a media player.

By using Emacs I will be able to access any file in the operating system with very little effort.

3. Indent code while creating a github issue.

Without Emacs: While creating a github issue if I want a piece of text to be recognised as code, I need to do proper indentation. So I have to manually indent those lines. Alternatively, I can copy that text to IDE, indent there and then copy back.

With Emacs: Emacs has a package which provides an interface to edit text areas directly in Emacs. So, I by calling a single function, I can easily format code part.

Since Emacs acts as an interface, the entire power of Emacs will be available for any task.

These are just a few examples. You can checkout wide variety of packages available for various things for more.

8 Escaping Meta Alt Control Shift

> An old joke: Emacs stands for "Escape Meta Alt Control Shift."
>
> Emacsism: Emacs stands for "Escaping Meta Alt Control Shift."

One of major problems with Emacs is that users need to use modifier keys[Shift, Ctrl, Alt/Meta, Super] repetetively. If we look at the key board laYout, `Ctrl` is placed at the corners. `Ctrl` is the most used modifier key in Emacs. Everytime we need to press it, we have to stretch our little finger all the way to the corner or move the hand, both of them are not ergonomic/convinient.

To avoid that problem, it is a customary practice to bind caps lock to ctrl. this solves the problem to an extent. If I have to press `C-c C-e`, I have to hold `caps lock` with little finger, and then press `c` & `e` with any remaining fingers. An alternate solution is to use `right control key` with right little finger and press `c` & `e` which takes back to the previous problem.

Also if we look at strength of our fingers, thumb stands and the top, little finger at bottom and remaining fingers somewhere in the middle. So binding caps lock to control doesn't solve the problem ergonomically.

Escaping Ctrl

One of the best solution for this problem is using `space` as `space` and `ctrl`. For Ubuntu, there is a package called space2ctrl[1] that modifies space to act as

- Space if You press it and leave
- Ctrl if You press and hold it.

This might seem bizzare at first but once You get used to the idea, it will be very useful. Once You do this, there is no need to touch ctrl key in Your life time.

Lets look at some common situations where we use shift most and how to avoid it Emacs.

Special characters:

The usage of symbols is more than that of numbers in programming(atleast for me). If You are using Emacs for programming, there is a snippet to interchange symbols and numbers. By this number of times to press shift will be reduced a lot.

[1] http://www.Emacswiki.org/Emacs/key-chord.el

Capitalize Words

If You are writing a book or documenting something, You need to capitalize words after a sentence/paragraph, always capitilize few characters/words e.t.c. You can use auto capitalize package mode which automatically does that for You.

Casing Words

Based on the programming language, You have to case the identifiers(as per language - snake case for python, camel case for java, capitalized keywords for SQL). Emacs has a package called electric-case which automatically does that for You. If You are using Emacs for SQL database access, there is a package called sql-up mode, which automatically upcases SQL keywords.

You can find many other packages based on Your requirements.

Escaping Meta/Alt

I don't see any problem with using meta since I am using thumb to press it. However if You are willing to rebind, You can do it. Most of the times meta is used only to invoke `M-x`. There is a package called key-chord[2] which lets You to rebind any key combination to any two keys pressed automatically. So You can rebind `M-x` to `m-x`.

I do recommend using key-chord, to rebind most used key bindings. It saves a lot of key presses, instead of typing `C-c C-p r` everytime, You can just press `p-r`(or any other keys which feel intuitive for You).

[2] http://www.Emacswiki.org/Emacs/key-chord.el

Whatever I have explained so far is Emacs part of less-keys project. Less-keys is a project I have started to minimize the number of keys and key presses to interact with computer to overcome RSI.

9 Emacs Singularity

> Singularity is an era in which our intelligence will become increasingly nonbiological and trillions of times more powerful than it is today—the dawning of a new civilization that will enable us to transcend our biological limitations and amplify our creativity.

Let us try to move to a directory called `Emacsism` which is present in `/home/anand/projects/books/lisp/Emacs/`.

Level 0: Typing

If I want to go to that directory using a bash shell, I have to type `cd /home/anand/projects/books/lisp/Emacs/Emacsism`. It will be difficult since I have to type such a long path. It even becomes annoying if I have to type it everytime I have to go to that directory.

Level 1: Autocompletion

If I use `zsh` instead of `bash`, I will get autocompletion for the path `/home/anand/projects/books/lisp/Emacs/Emacsism`. Now it becomes much easier to go to that directory as the amount of typing is decreased.

Level 2: Selection Narrowing

If I use a plugin called `jump` on top of `zsh`, I can use selection narrowing for path. By this, I can run `j Emacsism` on terminal, which takes me directly to that directory. Since it uses selection narrowing, I don't even need to type entire folder name. So instead of typing `j Emacsism` I can just type `j ism` which takes me to `Emacsism` as `ism` is unique to that directory. Thus selection narrowing tries to narrow down all available options based on few characters given by user.

This selection narrowing greatly improves productivity. However the method explained above is **not** interactive. Consider a case where I have two directories `Emacsism1` and `Emacsism2`. Now if I run `j ism` it will take me to recent visited of `Emacsism1` & `Emacsism2`.

On the other other hand Emacs selection narrowing(using Helm) is interactive. To change a directory in Emacs, I will call a command which is responsible for changing directories. By default it shows all the options so that I can navigate through them. If I type `i`, it narrows all the directories whose path has `i` in them. If I type `ism`, it filters down to two directories `Emacsism1` & `Emacsism2` and I can select any one. Thus by having interactive selection narrowing, Emacs will steer in the right direction.

Selection narrowing is not just limited to switching directories in Emacs. You can use it for anything.

If You have opened lot of files, just by typing a few characters, You can open that file.

Emacs has lots of commands. Some of them have keybindings. Most of them wont have keybindings. So selection narrowing can be used to narrow down commands.

If You have bookmarked lots of urls in Your browser, You can do selection narrowing on them from Emacs and when You select one, it will automatically takes You to browser with that page opened.

If You have starred thousands or repositories in github, You can do selection narrowing on them too.

You can define Your own commands which does selection narrowing on any items You want and this can be done with just one line of code.

Level 3: Singularity

> `Emacs singularity` is a hypothetical function, which is capable of doing anything!!

Singularity is a higher order of selection narrowing. In previous section, I have used selection narrowing for one set of options. To take if further, I can do selection narrowing on anything(buffers, files, commands, dirs, bookmarks).

selected one command & I have used selection narrowing for the argument. In addition to arguments, I can even do selection

narrowing on commands. In addition to performing selection narrowing individually on arguments or commands, I can do selection narrowing on a combination of command & argument. This is where singularity comes into picture.

Lets see how some general actions that are performed in Emacs.

If I want to activate a mode in a buffer, I have to run `M-x <mode-name>`.

If I want to switch from one buffer to another, I have to run `M-x switch-to-buffer <buffer-name>`. Since `switch-to-buffer` function is binded to `C-x b`, I can run `C-x b <buffer-name>`.

If I want to go to a file called `test.py` in some project, I have to run `M-x projectile-find-file <file-name>` or `C-c p f <file-name>`.

If I want to install a package in Emacs, I have to run `M-x package-install <package-name>`.

So most functions are invoked with `M-x <command-name>` or using a keybinding for that function. Then required arguments are passed to the function to perform some action.

Performing actions like this has some limitations.

- I have to remember quite a few keybindings.
- If I want to troubleshoot something in others Emacs, I don't know key bindings for that configuration.
- Its not possible to interchange order of command name and arguments.

Sometimes it will be much easier to specify argument to a command first and then specify command name. For example, if I have opened a file called `test.py`, it will be opened in a buffer called

`test.py`. Lets say I am in some other buffer and I want to switch back to `test.py` buffer, I have to run `M-x switch-to-buffer test.py`. After typing `M-x switch-to-buffer tes`, I have realized that buffer was killed. Now I need to cancel this command and I have to use `M-x projectile-find-file test.py` to open `test.py` file. On the other hand I could have typed argument to a function first which is `test.py` and then if buffer is present I can run `switch-to-buffer` or if it is not present I could run `projectile-find-file`.

Now, lets see how above actions can be performed with Singularity. Since I have bounded singularity to `fj` keys, I can invoke singularity by pressing `f` & `j` simultaneously.

If I want to activate a mode, I have to run `fj <mode-name>`.

If I want to switch from one buffer to another, I have to run `fj <buffer-name>`.

If I want to go to a file called `test.py` in some project, I have to run `fj <file-name>`.

If I want to install a package in Emacs, I have to run `fj ins <package-name>`(no need to type `package-install` as I am using selection narrowing). Commands will be narrowed down to `package-install` after typing `ins`.

Since singularity is capable of interacting with large set of options, it becomes much easier to interact with Emacs with very very few keys.

10 Epilogue

I would like to conclude this with a zen story by Osho which relates to emacs.

The Mysterious Wall

There was an ancient mysterious wall(Emacs) which stood at the edge of a village, and whenever anyone climbed the wall to look onto the other side, instead of coming back he or she smiled and would jump to the other side, never to return.

The inhabitants of the village became curious as to what could draw these people to the other side of the wall. After all, their village had all the necessities(IDE's and other software) of living a comfortable life.

They made an arrangement to where they would tie a person's feet, so that when he or she looked over and wished to jump, they could be pulled back. The next time someone tried to climb the wall to see what was on the other side, they chained her feet so that she could not go over.

She looked on the other side and was delighted at what she saw(lisp enlightment), and smiled. Those standing below grew curious to question her and pulled her back, but to their great disappointment she had lost the power of speech.

"Those who have seen cannot say. That which has been seen cannot be painted, cannot be reduced to words. But still each one has to give a try - and the world goes on becoming more and more beautiful because of these efforts."

Happy Emacsing & Lisping!!