The first hour of your next ten years: An Emacs introduction

Diego Vicente Martín

2017-11-21

Outline

About me

- Computer Science Graduate, M.Sc. @ UC3M
- Almost data scientist.
- Emacs user for around two years.

GitHub	DiegoVicen
Twitter	@DiegoVicen
Telegram	@DiegoVicente

- If everything else fails, send me an email

About this talk

1. Some brief context
2. Getting started with Emacs
3. Don't fear Emacs-Lisp
4. Packages worth checking out
5. Ask me something if I messed up

The most important Emacs insight

- Emacs has terrible defaults.
- It is super powerful once you configure it.
- Emacs is really old. (that's good)
- There are packages for (almost) everything.
- Is free, like in beer and freedom!

Wait, that was not the deal!

- "But Diego, I don't want to configure anything, I just want an editor!"
- I have good news for you!
- I can show you the best text editor ever.
- Zero configuration needed.
- "What is it?!"

. . . It's vim!

- Yup. For real.
  - It's everywhere*.
  - It's comfortable.
  - Zero configuration needed.
- You can exit with `:q`, `:quit`, `:wq`, `:q!`, `:exit`, `:x`, `ZZ` and `ZQ`. Stop being a baby.

Where's the trick?

- Probably, you want to do more things apart from editing text.
- Vim is pure UNIX philosophy: it can only edit text$^{\text{(but it's super good)}}$
- Configure, rebind keys, install plugins... You need vimscript.

If you want your own editor, welcome to Emacs

- Run this. If the result is larger than 20, install Emacs.

```
wc -l .vimrc
```

- You will be faster if your tools apply to your whole workflow.
- Starting is slow, it eventually pays off.
- In any case, learn vim. You can thank me later.

Installing Emacs

- Install it from your package manager of choice:

System	Build
Ubuntu	`emacs-snapshot` from `ppa:ubuntu-elisp/ppa`
Arch	`$ pacman -S emacs`
macOS	`brew cask install emacs`
Windows	May the gods help you

Booting up!

- Use the GUI version for proper support and eye-candy.
  - Some terminal emulators will capture your modifiers.
  - Don't worry, you'll keep your beloved terminal! – more later.
- Welcome to Emacs!

Emacs notation

`C-f`	Press at the same time ctrl and f (ctrl+f)
`M-d`	Press alt+d (probably)
`C-M-s`	Press ctrl+alt+s
`C-x C-c`	Press ctrl+x and then press ctrl+c
`C-x g`	Press ctrl+x and then only press g
`C-c p P`	Press ctrl+x, then p, and finally shift+p

First level is always a tutorial

- You can enter the tutorial clicking on the link on the splash screen.
- Another way to enter the tutorial is `C-h t`
- You will learn by practice movement, edition, and basic concepts of Emacs.
- You should do this. Actually, several times.
- When using the editor at first, put effort on practising new skills.

Muscle memory is the key

- Start using Emacs for small tasks.
- Learn by conscious practice!
- Incremental learning usually helps.
- If you don't get used to something, don't worry too much.

Introducing you to your best friends

- `C-h` prefix is used for help

`C-h f`	define a function
`C-h v`	define a variable
`C-h m`	define the current modes
`C-h k`	which function is bind to a keystroke
`C-h w`	where a function is bind

- For everything else, `C-h a`.

Emacs is the self-documenting editor

- Everything is extremely well documented.
- Only your own Emacs can help you understand your own Emacs.
- Third party packages are included in this documentation as well.
- Use them every single time before googling stuff.

Things I recommend you do:

- Rebind Caps-lock to Ctrl
- Unbind right alt key
- Take a look at the basic things in my configuration
- Use the GUI for full power!
- You can use your terminal inside Emacs.

Things people recommend but I don't:

- Use evil-mode to get vim inside Emacs.
- Run Emacs as a daemon.
- Install a pre-configured distribution.
- Use `customize`.

Demystifying Lisp syntax

- This is not an elisp introduction.
- I will show you what you need to configure things.
- Let's get this straight: Lisp is easy.
    - `f(x, y, z)` is written as `(f x y z)`
    - Now you can read Lisp!

A quick insight beforehand

- Emacs is not a text editor: Emacs is a REPL
- Executing elisp code can change the state of the editor.
- This is the way to customize Emacs.
- `M-:` lets you evaluate elisp code

Starting your own configuration

- Your configuration is an elisp file
- It contains the code to be executed when starting Emacs.
- All the code in there can be executed after starting and will also change the state of the editor!

Where the code lives

- You can host your configuration in two different places:
  1. `~/.emacs`
  2. `~/.emacs.d/init.el`
- I recommend the latter (tidier for VCS)

Some basic concepts

- There are some important configuration concepts you need to understand.
- Understanding these should be enough to configure most packages.

Setting a variable

- All variables are global
- Variables are used as switches some times
- You can set a variable global, local or to a default value.

```
(setq compilation-scroll-output t)
(setq-local compilation-scroll-output t)
(setq-default compilation-scroll-output t)
```

What can a variable be set to?

- Basically everything, there is no type system.
- You can even set variables to functions by quoting them

```
(setq inhibit-splash-screen t
      initial-scratch-message nil
      initial-major-mode 'org-mode)
```

Quoting lists and functions

- Quoting makes a function not evaluated.

```
(foo bar baz)  ;; => means apply function foo with
'(foo bar baz) ;; => is just a list of 3 values: foo
```

Hook, line and sinker

- You can use hooks to automatically execute something when an event happens
- Almost all major modes and important events have a hook.

```
(add-hook 'prog-mode-hook 'git-gutter-mode)
;; enable git-gutter-mode every time a programming 
```

Binding keys

- You can bind or rebind everything into everything.
- Use key-maps to make mode-dependant bindings.
- The `C-c <single-key>` is reserved for user bindings.

```
(global-set-key (kbd "C-c␣t") 'shell)
```

What can I configure?

- Virtually everything:
  - All keys are functions that can be modified
  - You can build on top of other functions
  - Graphical stuff, including other tools. . .
- Don't reinvent the wheel! You can also install packages.

Including MELPA

- Emacs comes with a package manager.
- MELPA is much more complete than the regular repository.

```
(when (>= emacs-major-version 24)
  (require 'package)
  (add-to-list
   'package-archives
   '("MELPA" . "https://melpa.org/packages/") t)
  (package-initialize))
```

Now you can install packages!

- Use `M-x list-packages` to see the packages available.
- Use `M-x install-package` to install a package by name.
- Include `(require <package>)` to force Emacs loading the package's functions.
- After getting some grip on it, I recommend checking `use-package` for building your configuration.

The concept of package

- Almost every language has dedicated packages for it:
  - Usually, a major mode and complementary minor modes.
  - A quick search in MELPA or Google probably gives you ideas.
  - *All* languages have great support and easy setup (except Java).
- Check other people's configuration for a language you want to use to get ideas.
- You can start by checking if I did something on your language.

An absolute must: Magit

- Magit is a git porcelain for Emacs.
- You can execute almost all kinds of git commands with a few keystrokes.
- It is just a reason by itself to use Emacs.

Organizing your life in projects: Projectile

- Uses the concept of project:
  - A project is something with a configuration file or a VCS
- Lets you compile, run, test, search, etc per project
- Combine it with `persp` to get window distribution per project

Connecting to a remote server: TRAMP

- TRAMP is black magic included in Emacs by default
  1. Run `find-file` (`C-x C-f`) and erase the current directory
  2. Write `ssh:<user>@<server>` to connect to a remote server
- You keep all your configuration and tools when using TRAMP
- You can even bookmark remote files to access them later

Using the terminal inside Emacs

- There two main alternatives:
  1. `shell`: A terminal emulator inside Emacs
     - It uses your `zshrc` / `bashrc` / `fishrc`
     - For that reason, it may break if your configuration is too complex
  2. `eshell`: A terminal emulator written in elisp
     - It has its own configuration and implementation of some commands
     - It is a buffer
     - Inadequate for certain tasks
- `shell` should suffice everything you do in your current setup.

Other tools for programming

`smartparens` automatically close parenthesis/braces/quotes

`auto-complete` / `company` code auto-completion back-ends

`flycheck` code linting

Now Emacs is a small, Swiss knife IDE!

Better functions and completion: `helm`

- `helm` is a narrowing framework
- It overrides some functions (find files, search, `M-x`...)
- Enables search in candidates, fuzzy search, patterns, etc
- It's maybe a bit too much for you

A lighter and powerful alternative: ivy, counsel

- ivy is a lighter and minimal alternative to helm
- Most of the times is faster
- Reduced and less intrusive interface
- It has *almost* all of helm's features

A more powerful search: `swiper`

- Built on top of `ivy`
- Its `isearch` with an overview of results
- This may seem dumb but is f***ing amazing.

Your file system in Emacs: `dired`

- Default in Emacs
- Copy, rename, move and delete files.
- `C-x C-q` lets you make changes by writing in the buffer.
- Pro tip: `(add-hook 'dired-mode-hook 'dired-hide-details-mode)`

What is org?

- `org` mode is a note manager
- It is super extensible and you can work magic with it
- It is the other deal-breaker

Note taking in org

- Org has its own markdown language
  - **Bold**
  - *Italics*
  - underline
  - And yeah, this is a list.

It also has tables!

This is a cool thing to do a live demo.

Task management in org

- Org headings can be marked as to-do
- You can also schedule tasks and add deadlines
- If you schedule things, you can see them in the agenda.

Exporting notes

- What is the point of org if you can't share notes?
- `C-c C-e` brings up the export engine
- You can add more export modes with packages

Source blocks are OP

```
import numpy as np

A = np.array( [[7, 5, -3],
               [3, -5, 2],
               [5, 3, -7]])

B = np.array( [[16],
               [-8],
               [ 0]] )

return np.dot(np.linalg.inv(A), B)
```

What can you do with source blocks?

- Literate programming
- Tutorials
- Scripts
- Include code to generate figures

What else to do with org?

`org-ref`	citation manager written in org
`hugo`	Static site generator that supports org
	Literate devops, according to Howard Abrams
	Remember, this presentation is org!

My 2 cents:

- Don't over-do it:
  - Start with a minimal configuration.
  - Control your Emacs before building more on top.
  - DON'T COPY SOMEONE ELSE'S CONFIGURATION.
- Be patient:
  - It takes time to develop insight and muscle memory.
  - Don't move your workflow to Emacs just yet.

Several resources

- r/emacs in Reddit
- #emacs in IRC (`M-x erc` is a IRC client in Emacs)
- Emacs! Remember to ask Emacs in case of doubt.
- Just send me an email/tweet if you have a question!

Questions?

- Have you heard of a feature in Emacs and I didn't mention it?
- Are you unsure of how to do something you need in Emacs?
- Do you want me to talk about anything related to it?
- Do you want me to whistle In a Sentimental Mood, by Duke Ellington?
- Ask me!

Thank you!