Introduction to Emacs

TABLE OF CONTENTS

SOME HISTORY

The **text editors** we use today are very different from the text editors used in the early decades of computer science.

One of the first line editor was developed in 1969 for the **UNIX** operating system, and it is called **ed**

ed – Line editor

How to use di ed (1/2)

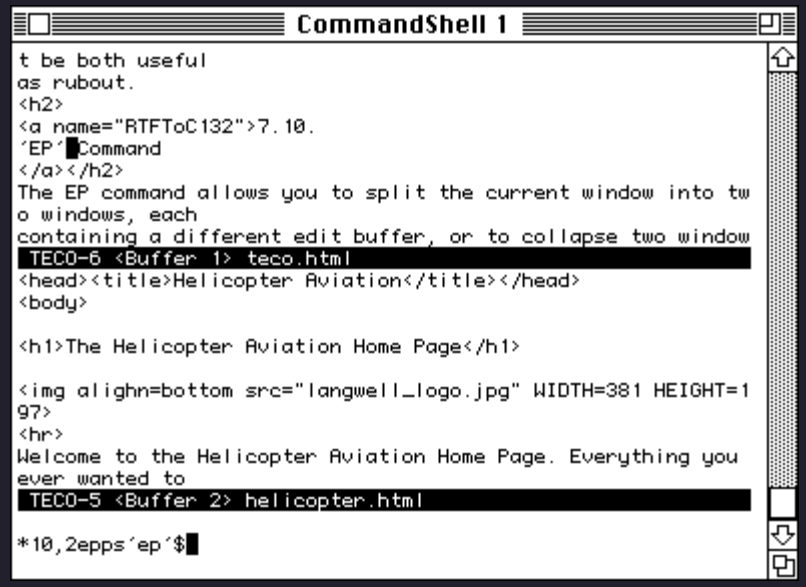
Write `"Hello World!"` in the file `hello.txt`

```
$ ed
a                # start append mode
Hello World!     # write text
.                # stop append mode
w hello.txt      # write to file
13
```

Out of **ed** and other similar editors came the **vi** family of text editors

- ed, 1969
- vi, 1976
- vim, 1991
- neovim, 2014

Another editor of that time was called **TECO**, and it was developed in 1962 by **Dan Murphy**, during his period at MIT.



TECO → Text Editor & COrrector

TECO implemented the notion of **macros**

TECO is not only an editor *but also an* ***interpreted programming language*** *for text manipulation.*

Arbitrary programs (called "macros") for searching and modifying text give it great power. (Wikipedia)

What are **macros**?

What are **macros**?

Macros are methods for saving **arbitrary sequences of keybinds** under simple alises, which can then be called on in order to execute again the saved sequence.

Example of a TECO's macro

So complex it is pratically a "**write-only**" language

```
!START! j 0aua
!CONT! l 0aub
qa-qb"g xa k -l ga 1uz '
qbua
l z-."g -l @o/CONT/ '
qz"g 0uz @o/START/ '
```

(Craig Finseth, The Craft of Text Editing)

(Real Programmers Don't Use Pascal)

Out of TECO came different extensions of the editor.

Among these we find **TECMAC** e **TMACS**, developed around 1976 from **Guy Steele**, **Dave Moon**, **Richard Greenblatt**, **Charles Frankston**, et al.

$$\mathrm{TECO} \longrightarrow \mathrm{TECMAC}, \quad \mathrm{TMACS}$$

Also in 1976, **Richard Stallman**, **Guy Steele** and **Dave Moon** develop the first version of **EMACS**

$$?\mathrm{MACS} \longrightarrow \mathrm{EMACS} \longrightarrow \mathrm{Editor\ MACroS}$$

After this first implementation of Emacs various others came later

- TECMAS, TMACS (1976)
- EMACS, EINE (Eine is Not Emacs) (1976)
- Multics Emac, ZWEI (ZWEI Was EINE Initially) (1978)
- ZMACS (1980)
- Gosling Emacs (1981)
- …
- **GNU Emacs** v13.0 (1985)
- …

Emacs Timeline

Of these, one of the most important has been **GNU Emacs**, released in 1985 as the first software of the **GNU project**, founded by **Richard Stallman** in 1978.

Key difference between Emacs and vi

vim was developed to edit text, and only edit text. Emacs was developed as a comprehensive software system that can also edit text and that runs on top of the underlying OS.

Let us now understand in depth what makes Emacs special.

WHAT IS EMACS?

Initially, you can think of Emacs as a **very, very powerful text editor**.

Emacs can indeed edit any sort of text, and in general it is very powerful when it comes to textual manipulations.

We must be careful however to not think of Emacs as only a text editor.

That's because Emacs is much, much more.

Emacs contains an interpreter for the programming language called **emacs-lisp**

Elisp code that computes the n th Fibonacci number

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$

```
(defun fib (n)
  (cond ((eq n 1) 1)
        ((eq n 2) 1)
        (t
         (+ (fib (- n 1))
            (fib (- n 2))))))
```

$$F_1 = F_2 = 1$$
$$F_n = F_{n-2} + F_{n-1},\ n \geq$$

Questo è un file di testo.

Emacs permette di modificare e gestire i file di testo.

Ma Emacs non si limita ad essere un text editor.

Tramite Emacs siamo infatti in grado di eseguire codice elisp.

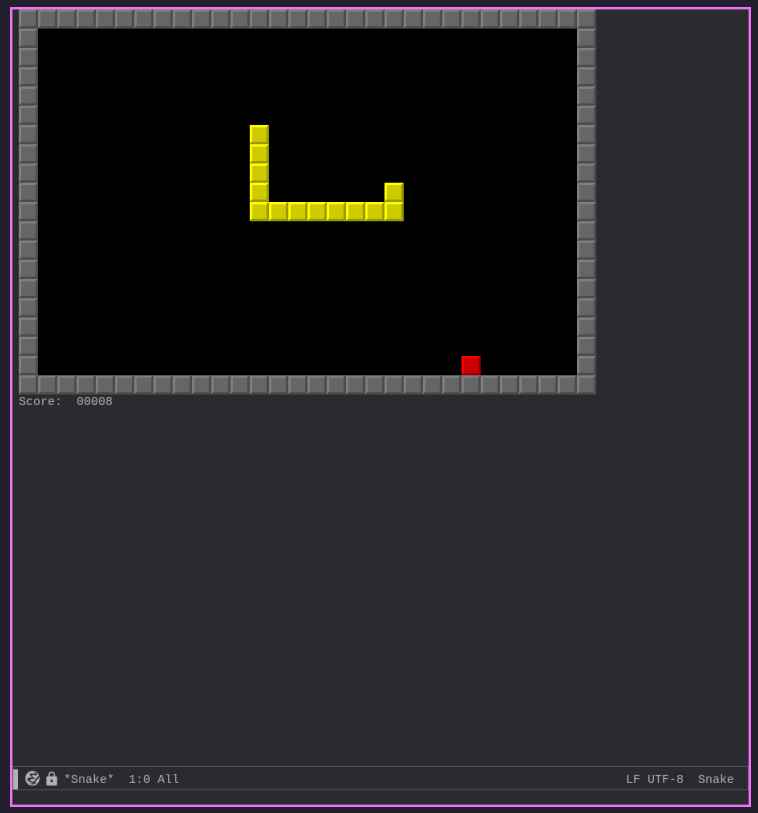
```
#+begin_src elisp
(defun fib (n)
  (cond ((eq n 1) 1)
        ((eq n 2) 1)
        (t
         (+ (fib (- n 1))
            (fib (- n 2))))))

(fib 10)
#+end_src
```

```
#+RESULTS:
: 55
```

test.txt 21:4 11:50 LF UTF-8 Org (+5)

What makes Emacs extremely flexible the ability to execute emacs lisp code, which is a **Turing-Complete language** with many interesting characteristics.

For example, we can play games in emacs



Learning Emacs comes down to learning about **emacs-lisp** and the ways in which you can use elisp code to program any sort of actions and behavior that Emacs can execute.

It is about learning what it truly means to have a fully programmable and extensible software.

I suggest you to read the 1981 article by Stallman himself

https://www.gnu.org/software/emacs/emacs-paper.html

EMACS
The Extensible, Customizable Self-Documenting Display Editor

Richard M. Stallman
Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge, MA 02139

Paper abstract

EMACS is a display editor which is implemented in an interpreted high level language. This allows users to extend the editor by replacing parts of it, to experiment with alternative command languages, and to share extensions which are generally useful.

The ease of extension has contributed to the growth of a large set of useful features. This paper describes the organization of the EMACS system, emphasizing the way in which extensibility is achieved and used.

The importance of emacs-lisp (1/2)

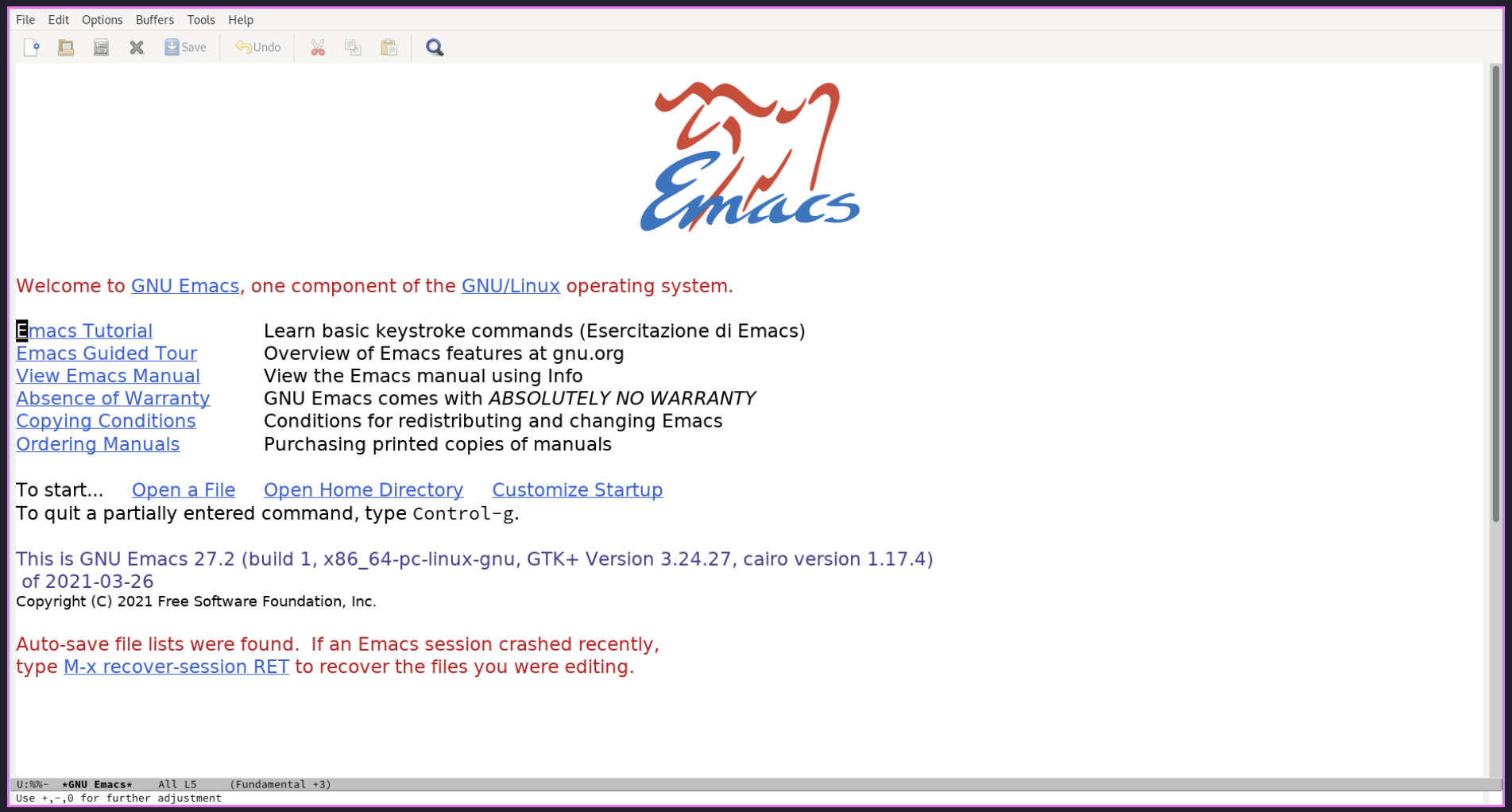
Adherents of non-Lisp programming languages often conceive of implementing an EMACS for their own computer system using PASCAL, PL/I, C, etc. ***In fact, it is simply impossible to implement an extensible system in such languages****. This is because their designs and implementations are batchoriented; a program must be compiled and then linked before it can be run.*

The importance of emacs-lisp (2/2)

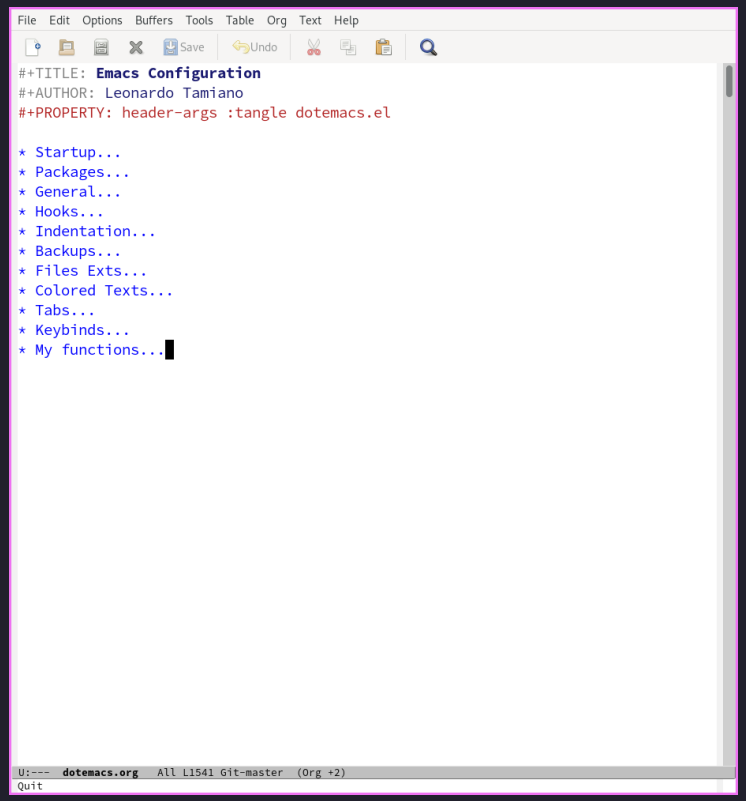
***An on-line extensible system must be able to accept and then execute new code while it is running**. This eliminates most popular programming languages except LISP, APL and Snobol. At the same time, Lisp's interpreter and its ability to treat functions as data are exactly what we need.*

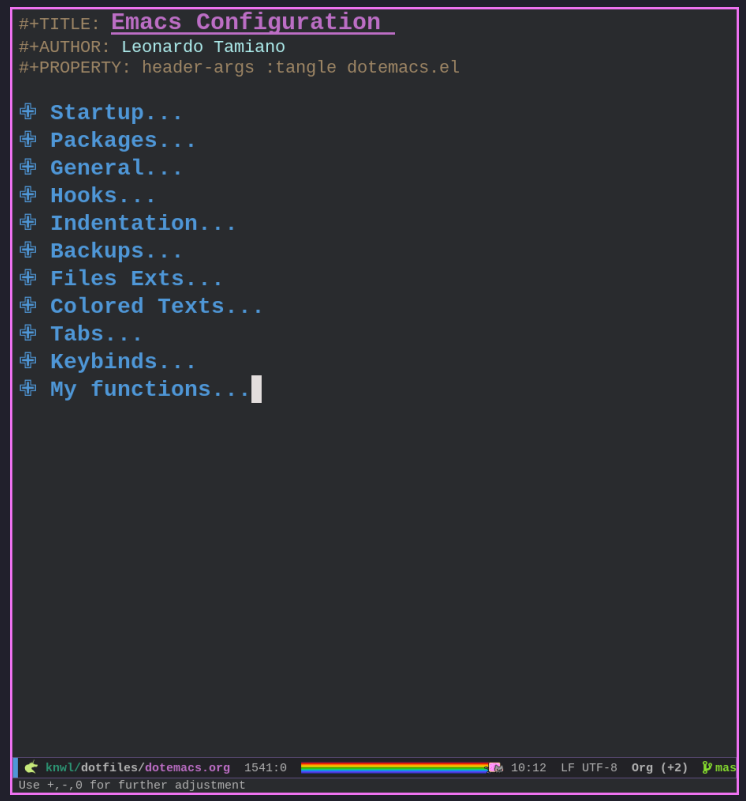
HOW I USE EMACS

Even though Emacs is a very powerful tool, it's learning curve is non-trivial.



It takes a lot of dedication, effort and time to unlock it and to begin to understand its power.





I will now show some (few) personal use-cases in order to make you understand a bit better what Emacs allows you to do.

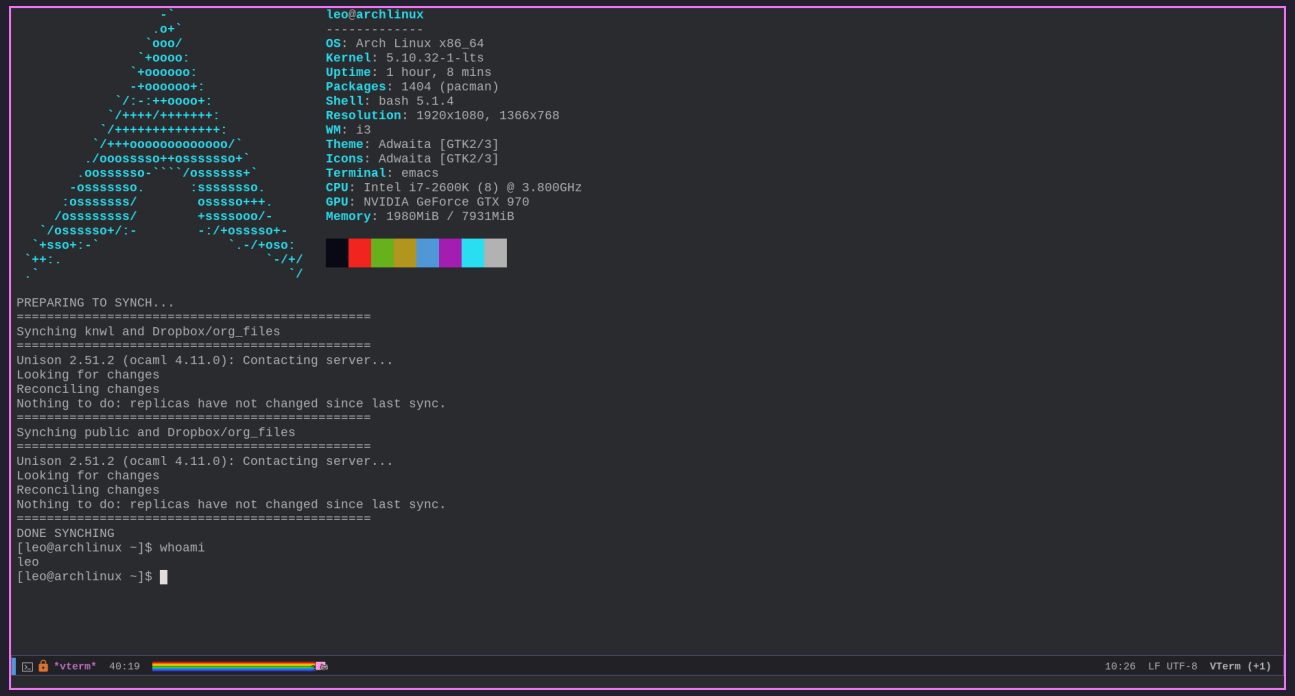
I have divded my use-cases into the following categories:

- OS interface layer
  - Terminal emulator
  - File explorer
  - Email manager
- Programming
  - Git client
  - Rust IDE
- Org-Mode
  - Notes
  - Blog
  - Slides

OS INTERFACE LAYER

TERMINAL EMULATOR

There are various **packages** for transforming Emacs into a typical **terminal emulator**



(**vterm**, **term-mode**, **shell-mode**, **eshell**)

FILE EXPLORER

Emacs has a built-in file explored called **dired**, which allows us to deal with files and directories.

```
  /home/leo/repos/TBD/test:
  total used in directorye 44 available 334.3 GiB
  drwxr-xr-x 11 leo users 4096 24 apr 10.52 .
  drwxr-xr-x  8 leo users 4096 24 apr 10.52 ..
  drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_1
D drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_2
  drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_3
* drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_4
  drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_5
  drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_6
  drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_7
  drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_8
  drwxr-xr-x  2 leo users 4096 24 apr 10.52 dir_junk_9

~/repos/TBD/test/                                    10:52  Editable Dired (+4)
```

EMAIL MANAGER

There are packages for dealing with emails (**mu4e**, **gnus**, **notmuch**)

```
* mu4e - mu for emacs version 1.4.5

  Basics

        * [j]ump to some maildir
        * enter a [s]earch query
        * [C]ompose a new message

  Bookmarks

        * [bu] Unread messages      (0/0)
        * [bt] Today's messages     (0/0)
        * [bw] Last 7 days
        * [bp] Messages with images (0/414)

  Misc

        * [;]Switch context
        * [U]pdate email & database

        * [N]ews
        * [A]bout mu4e
        * [H]elp
        * [q]uit

  Info

        * database-path       : /home/leo/.cache/mu/xapian
        * maildir             : /home/leo/Maildir
        * in store            : 7824 messages
        * personal addresses  : leonardotamiano95@gmail.com, leo95.shop@gmail.com, leo95.game@gmail.com, leo95.dev@gmail.com,
leotwork@protonmail.com, leonardo.tamiano@alumni.uniroma2.eu

*mu4e-main*  1:0                                                    [Personal]10:29  LF UTF-8  mu4e:main (+2)
Use +,-,0 for further adjustment
```

For those interested in a possible mu4e configuration walkthrough, check out my blog post on the matter

https://blog.leonardotamiano.xyz/tech/mu4e-setup/

PROGRAMMING

The extreme flexibility of Emacs allows to create custom IDE-like experiences in various programming languages.

GIT CLIENT

The **magit** package allows a simple and intuitive client interface to the versioning software **git**.

```
Head:     master Update
Push:     local/master Update

Untracked files (13)
Unstaged changes (12)
modified   dotfiles/.compton.conf
modified   dotfiles/.i3blocks.conf
@@ -5,6 +5,10 @@ separator=true
 separator_block_width=15
 markup=pangon

+[covid19]
+label=VAC
+interval=once
+
 # Disk usage
 #
 # The directory defaults to $HOME if the instance is not specified.
modified   dotfiles/emacs_bookmarks
modified   languages/bash/ppa
modified   languages/latex/cv/auto/cv.el
modified   languages/latex/cv/cv.log
modified   languages/latex/cv/cv.pdf
modified   languages/latex/cv/cv.tex
modified   languages/python/utils/index_generator.py
modified   notes/languages.org
modified   notes/os.org
modified   notes/tools.org

Recent commits

magit: knwl  30:0                                          10:41  LF UTF-8  Magit (+2)
```

RUST IDE

Thanks to **lsp**, **rustic** and **rust-analyzer**, we can have a comfortable experience while programming in **rust**

```
> src >  crypto.rs >  cbc_encrypt
192     (&(&bytes)[0..(&bytes).len() - padding_byte as usize]).try_into().unwrap()
193 }
194
195 // -------------------
196
197 pub fn cbc_encrypt(plaintext_bytes: &Vec<u8>, key: [u8; 16], iv: [u8; 16]) -> Vec<u8> {
198     // add padding when needed
199     let pbytes : Vec<u8> = (&*plaintext_bytes).clone();
200     let plaintext_bytes : Vec<u8> = if &((&*plaintext_bytes).len() % 16) != &0 {
201         pad_pkcs7(pbytes, 16)
202     } else {
203         pbytes
204     };
205
206     let mut ciphertext_bytes : Vec<u8> = Vec::new();
207     let mut iv_bytes : [u8; 16] = iv;
208
209     let mut i: usize = 0;
210
211     while &(i + 16) <= &(&plaintext_bytes).len() {
212         let plaintext_chunk : Vec<u8> = (&(&plaintext_bytes)[i..i+16]).try_into().unwrap();
213
214         let iv : Vec<u8> = iv_bytes.try_into().unwrap();
215         let chunk : [u8; 16] = apply_xor(&plaintext_chunk, &iv).try_into().unwrap();
216         let encrypted_chunk : [u8; 16] = ecb_encrypt_block(chunk, key);
217
218         (&mut ciphertext_bytes).append(&mut *&mut (&encrypted_chunk).to_vec());
219
220         iv_bytes = encrypted_chunk;
221         &mut i += 16;
222     }
223
224     ciphertext_bytes
225 } fn cbc_encrypt
226
227 pub fn cbc_decrypt(ciphertext_bytes: &Vec<u8>, key: [u8; 16], iv: [u8; 16]) -> Vec<u8> {
228     // TODO: add check on vector length. If it is not divisible by 16, trigger a panic.
229
230     if &((&*ciphertext_bytes).len() % 16) != &0 {
231         panic!("[INFO]: CBC can only decrypt blocks of 16 bytes length");
232     }
233
234     let mut plaintext_bytes : Vec<u8> = Vec::new();
235     let mut iv_bytes : [u8; 16] = iv;
236
237     let mut i: usize = 0;
238     while &(i + 16) <= &(&*ciphertext_bytes).len() {
239         let encrypted_chunk : [u8; 16] = (&(&*ciphertext_bytes)[i..i+16]).try_into().unwrap();
240
-UU-:----F1  crypto.rs      44% (209,18)  Git:main  (Rustic company Lens FlyC LSP[rust-analyzer:48] (*) ivy ElDoc Wrap) 2/2 7:47AM ----------
Quit
```

In general, for most programming languages you will find an Emacs package for that language.

ORG-MODE

Out of the many functionalities of Emacs, **org-mode** is without a doubt among the greatest thing you will find in Emacs.

Org-mode can be seen from two point of views:

1. As a **markup language**, such as markdown, xml or html
2. As a **major-mode** in **Emacs**

Org-mode files are plaintext files that are dynamically processed by Emacs

```
#+TITLE: Esempio file Org-Mode
#+AUTHOR: Leonardo Tamiano

* Outline 1
  Prova.

** Sub-Outline 1.1
*** Sub-Sub-Outline 1.1.1
*** Sub-Sub-Outline 1.1.2
** Sub-Outline 1.2
*** Sub-Sub-Outline 1.2.1
    Prova prova.

* Outline 2
** Sub-Outline 2.1
** Sub-Outline 2.2
   Prova prova.

*** Sub-Sub-Outline 2.2.1
*** Sub-Sub-Outline 2.2.2
**** Sub-Sub-Sub-Outline 2.2.2.1
```

test.org 21:32 All LF UTF-8 Text (+4)
Use +,-,0 for further adjustment

```
#+TITLE: Esempio file Org-Mode
#+AUTHOR: Leonardo Tamiano

* Outline 1
  Prova.

** Sub-Outline 1.1
*** Sub-Sub-Outline 1.1.1
*** Sub-Sub-Outline 1.1.2
** Sub-Outline 1.2
*** Sub-Sub-Outline 1.2.1
    Prova prova.

* Outline 2
** Sub-Outline 2.1
** Sub-Outline 2.2...
```

test.org 16:0 11:30 LF UTF-8 Org (+3)
Use +,-,0 for further adjustment

The syntax used by org-mode is called **Org-Mode Markup Language**

```
* This Is A Heading
** This Is A Sub-Heading
*** And A Sub-Sub-Heading
    Paragraphs are separated by at least one empty line.

    *bold* /italic/ _underlined_ +strikethrough+ =monospaced=
    [[http://Karl-Voit.at][Link description]]
    http://Karl-Voit.at → link without description

    - list item
      - sub-item
        1. also enumerated
    - [ ] yet to be done
    - [X] item which is done

    : Simple pre-formatted text such as for source code.
```

Source: karl-voit - Org Mode Is One of the Most Reasonable Markup Languages to Use for Text

NOTES

Anytime I have to take a note on something, I use Emacs with org-mode.

- Studying
- Penetration testing
- Writing

HTB writeup written in org-mode

```
#+TITLE: HTB - Reddish
#+AUTHOR: Leonardo Tamiano

* Enumeration...
* Privilege Escalation...
* Original Walkthrough...
* Video notes
  [2021-12-25 Sat 03:37]

  - Scans with nmap
    nmap -sC -sV reddish

    #+begin_example
All 1000 scanned ports on reddish (10.129.180.63) are closed
    #+end_example

    nmap -p- reddish

    #+begin_example
PORT     STATE SERVICE
1880/tcp open  vsat-control
    #+end_example

    nmap -sC -sV -p 1880 reddish

    #+begin_example
PORT     STATE SERVICE VERSION
1880/tcp open  http    Node.js Express framework
|_http-title: Error
    #+end_example

  - RCE on Docker #1...
  - Docker #1 (NODE-red)...
  - Docker #2 (Redis)...
  - Docker #3 (Web server)...
  - Pivoting from Docker #1 to Docker #3...
  - PrivEsc on Docker #3 (user)...
  - Docker #4 (Backup server)...
  - Pivoting from Docker #3 to Docker #4...
  - PrivEsc on Docker #4 (root)...
* Resources and Extra...
* Flags...

htb/m/r/reddish_log.org  1172:1                    All [500]  LF UTF-8  Org  maste
```

BLOG

I manage my blog in a single file org.

```
#+TITLE: Blog Content
#+STARTUP: content
#+AUTHOR: Leonardo Tamiano
#+HUGO_BASE_DIR: ../
#+HUGO_AUTO_SET_LASTMOD: t
#+OPTIONS: author:nil ^:nil

About...
Favorites...
Reviews...
Posts
  :PROPERTIES:...

  DONE How to manage e-mails in Emacs with Mu4e    :tech:emacs:...
  DONE The Complete Beginners Introduction to Emacs - Part 1/3 - How I use Emacs :tech:emacs:...
  DONE The Complete Beginners Introduction to Emacs - Part 2/3 - The Basics of Emacs
:tech:emacs:...
  DONE Se questo è un uomo                                    :life:books:...
  DONE The Complete Beginners Introduction to Emacs - Part 3/3 - A Crash Course on Emacs-Lisp
:tech:emacs:...
  DONE Sulla pigrizia                                                 :life:...
  DONE The Importance of RSS                                :tech:university:...
  DONE How I Learned the Power of the Command Line     :life:htb:university:...
  DONE How I Track my Expenses with Ledger          :tech:finance:...
  DONE On Doubly-Linked Lists, and how they are Implemented in the Linux Kernel
:tech:data_structures:programming:...
  DONE Implementing a Crossword Solver in C++       :tech:programming:games:...
  DONE La tregua                                              :books:life:...
  DONE Lettera pre COVID-19                                           :life:...
HTB Writeups
  :PROPERTIES:...

  DONE HTB Writeup - Bashed                                  :tech:htb:...
  DONE HTB Writeup - Nibbles                                 :tech:htb:...
  DONE HTB Writeup - Poison                                  :tech:htb:...
  DONE HTB Writeup - Valentine                               :tech:htb:...
Scratch...
Footnotes...
COMMENT Local Variables                                          :ARCHIVE:...

blog/content-org/content.org  141:7 All                LF UTF-8  Org  master
```

cd ~

cv whoami favorites archive

Hello, whoever you are, and welcome to my blog (♥).

I'm a university student, and I enjoy writing about a bunch of different things, such as:

books data-structures emacs finance games htb life programming tech university

I also share all my university lecture notes, the code I write, and other stuff in my ppa, with the hope that it may help you in some way.

And now, some quotes:

The heart has its reasons which reason knows nothing of.

Blaise Pascal

HTB Writeup - Valentine

2021-03-29

#tech #htb

Complete writeup of the Valentine machine in Hack The Box.

Read more →

```
† Bonus
 † Inside heartbleed (CVE-2014-0160)
   To understand better this vulnerability the following resources
   might help:

   - Commit in which it was fixed
   - CVE-2014-0160
   - https://www.exploit-db.com/exploits/32745
   - https://www.exploit-db.com/exploits/32764
   - https://stackabuse.com/heartbleed-bug-explained/

   By downloading the version of the code right before it got fixed,
   we can start to analyze the flaw. The files of interested are
   two, as is shown the commit: ssl/d1_both.c and
   ssl/t1_lib.c. Since however the d1_both.c is used for the DTLS
   implementation, which a version of TLS that uses the UDP
   transport protocol - as opposed to the traditional version of TLS
   that uses TCP - we will only check out t1_lib1.c. Let us
   first discuss the flawed version of the code, and at the end we
   will discuss how it was fixed.

   The function of interest is tls1_process_heartbeat().  As soon as
   we enter in the function the server reads the number bytes that
   the client declared to have sent, and puts such number in the
   payload variable.

   #+begin_src c
#ifndef OPENSSL_NO_HEARTBEATS
int
tls1_process_heartbeat(SSL *s)
{
  // -- p points to record data from client
  unsigned char *p = &s->s3->rrec.data[0], *pl;
  unsigned short hbtype;
  unsigned int payload;
  unsigned int padding = 16; /* Use minimum padding */

  /* Read type and payload length first */
  hbtype = *p++;
  // -- read 2 bytes from p and put them in payload
  n2s(p, payload);
  pl = p;

  // -- now payload has length of data from client

   #+end_src

   If the message was an heartbeat request, the server then proceeds
   to allocate memory. The memory allocated depends on the amount of
   data specified by the client, that is on the variable payload.

   #+begin_src c
blog/content-org/content.org  9407:70 95%          LF UTF-8  Org  master
```

Bonus

Inside heartbleed (CVE-2014-0160)

To understand better this vulnerability the following resources might help:

- Commit in which it was fixed
- CVE-2014-0160
- https://www.exploit-db.com/exploits/32745
- https://www.exploit-db.com/exploits/32764
- https://stackabuse.com/heartbleed-bug-explained/

By downloading the version of the code right before it got fixed, we can start to analyze the flaw. The files of interested are two, as is shown the commit: `ssl/d1_both.c` and `ssl/t1_lib.c`. Since however the `d1_both.c` is used for the `DTLS` implementation, which a version of TLS that uses the UDP transport protocol – as opposed to the traditional version of TLS that uses TCP – we will only check out `t1_lib1.c`. Let us first discuss the flawed version of the code, and at the end we will discuss how it was fixed.

The function of interest is `tls1_process_heartbeat()`. As soon as we enter in the function the server reads the number bytes that the client declared to have sent, and puts such number in the `payload` variable.

```c
#ifndef OPENSSL_NO_HEARTBEATS
int
tls1_process_heartbeat(SSL *s)
{
  // -- p points to record data from client
  unsigned char *p = &s->s3->rrec.data[0], *pl;
  unsigned short hbtype;
  unsigned int payload;
  unsigned int padding = 16; /* Use minimum padding */

  /* Read type and payload length first */
  hbtype = *p++;
  // -- read 2 bytes from p and put them in payload
  n2s(p, payload);
```

SLIDES

I prepare my slides using org-mode

```org
#+TITLE: Introduzione ad Emacs
#+AUTHOR: Leonardo Tamiano
#+OPTIONS: num:nil toc:2
#+REVEAL_THEME: cyberpunk

* Che cos'è Emacs?...
* Come utilizzo Emacs
  #+REVEAL: split

  Per quanto Emacs sia un tool estremamente potente, la configurazione
  iniziale lascia molto a desiderare, sia in termini di estetica che
  di funzionalità.

  #+REVEAL_HTML: <img src="./images/emacs_default.png" class="stretch" >

  #+REVEAL: split

  Investendo abbastanza tempo ed impegno però è possibile trasformarlo
  in ciò che si vuole.

  #+REVEAL_HTML: <div>
  #+REVEAL_HTML: <div class="column" style="float:left; width:40%">
  #+REVEAL_HTML: <img src="./images/emacs_conf_default.png">
  #+REVEAL_HTML: </div>

  #+REVEAL_HTML: <div class="column" style="float:left;margin-top:130px; margin-left:50px;">
  $$\longrightarrow$$
  #+REVEAL_HTML: </div>

  #+REVEAL_HTML: <div class="column" style="float:right; width:40%">
  #+REVEAL_HTML: <img src="./images/emacs_conf_cool.png">
  #+REVEAL_HTML: </div>
  #+REVEAL_HTML: </div>

  #+REVEAL: split

  Andiamo adesso a vedere alcuni (pochi) use-case personali di
  utilizzo di Emacs.

  + Interfacciamento OS...
  + Programmazione...
  + Org-Mode...
* Perché imparare Emacs?...
```

```
intro_emacs.org  270:0                                  11:38  LF UTF-8  Org
Mark set
```

INTRODUZIONE AD EMACS

LEONARDO TAMIANO

Created: 2021-04-24 sab 11:37

WHY YOU SHOULD LEARN EMACS

Learning Emacs is difficult, mostly because it's ideas are not trivial to understand, since they are vastly different from what we're typically exposed in our development years.

Yet, there are many reasons why someone might decided to learn Emacs.

Why should you learn Emacs?

- it can be fun!
- it can expand your mind
- it can increase your productivity

At the end of the day, I do not think it is useful to approach Emacs just simply because you want to be more productive.

You will get stuck, you will lose (momentarily) your productivity, and you will go back to tools that work "ok" on first install.

Approach Emacs with an open mind.

Allow yourself to explore, to waste time, to get stuck.

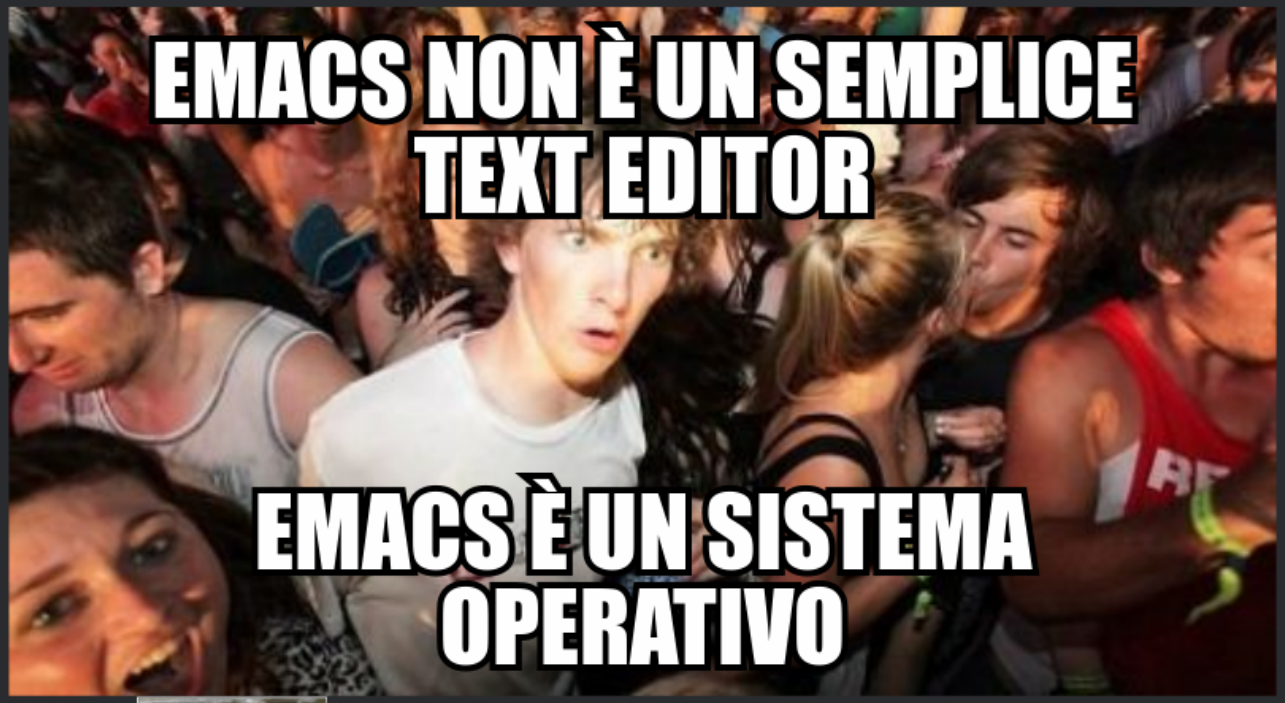
This will bring you closer to the true power of Emacs.

The more you learn about Emacs, the more you will understand all the different problems that ca be tackled with it, with uniquely fascinating solutions that only makes sense within Emacs.

You can even make memes in Emacs.

Yes, memes.

```
Top     Emacs non è un semplice text editor
        20   60   white   middle   impact
Bottom  Emacs è un sistema operativo
        20   60   white   middle   impact
```




```
*meme*   3:18                                   1:08   LF UTF-8   Meme
email_sync: finished.
```