# Emacs...

E sulla resilienza del software

# TABLE OF CONTENTS

# WHOAMI?

Modo migliore di presentarsi al **linux day**?

Modo migliore di presentarsi al **linux day**?

---

Tramite la variabile d'ambiente **PS1** utilizzata da **Bash**!

## **Default shell prompt** (1/2)

```
[leo@archlinux linux_day_emacs]$ echo $P
[\u@\h \W]\$ \[$(vterm_prompt_end)\]
```

# **Default shell prompt** (2/2)

## Qualche info su di me...

---

$$
\begin{aligned}
\text{Nome} &\longrightarrow \text{Leonardo Tamiano} \\
\text{Lavoro} &\longrightarrow \text{Ricercatore @ CNIT (ci provo)} + \\
&\qquad\quad \text{Inizio PhD a breve} \\
\\
\text{Nel Tempo Libero} &\longrightarrow \text{Quale tempo libero?} \\
&\longrightarrow \text{Video su youtube} \\
&\longrightarrow \text{CTFs} \\
&\longrightarrow \text{Psicologia/Comportamento Umano}
\end{aligned}
$$

Tre domande per iniziare…

- Come siamo arrivati ad **Emacs**?
- Che cos'è, esattamente, **Emacs**?
- Cosa è possibile fare con **Emacs**?

Iniziamo quindi da un **po' di storia**…

# UN PO' DI STORIA

Gli utilizzatori delle **tecnologie digitali** sono

# Manipolatori di Informazioni

La differenza nei vari utilizzi è da ritrovarsi nella **distanza** presente tra l'**utente finale** e le **informazioni** che sta manipolando.

In ogni caso, siamo tutti **manipolatori di informazioni**.

# Siamo tutti **manipolatori di informazioni**

## Visione utente

## Visione programmatore

```
af ec e0 95 ca a3 9c 95  48 eb a8 78 8c bb 2d 48
99 ff f9 15 b9 30 40 5c  45 aa 8a b9 ac 30 08 76
46 c0 93 2f 3f ac 63 ef  c7 ef 09 cf 7e 0b 32 ce
89 af 35 b7 6f 54 e3 b0  e8 4c d1 76 c2 54 5d ec
dd 57 69 fa ad e2 8e 65  8d 0a e9 46 99 d1 bd b1
```

# La vera visione del programmatore…

---

# Utente $\longleftrightarrow$ Software $\longleftrightarrow$ Informazioni

---

Gli **strumenti** che utilizziamo per manipolare queste informazioni sono dunque molto importanti.

Nasce dunque una **domanda**:

---

# Ma inizialmente, come si manipolavano le informazioni?

Nel 1801 **Joseph Marie Jacquard** (1752-1834) dimostrò per la prima volta l'utilizzo delle **punched cards** (schede perforate) per codificare sequenze di operazioni complesse che potevano essere eseguite dal suo **telaio**.

Jacquard's Loom

# Esempio di **punched card** (scheda perforata)

---

# Dalle schede perforate ai pattern nei tessuti

Intorno al 1848 **Ada Lovelace** scrisse il primo programma per la **Analytical Engine**, ideata da **Charles Babbage**.

# Charles Babbage's Analytical Engine

---

**Il programma era codificato in una sequenza di schede perforate**, e permetteva di calcolare i **numeri di bernoulli**.

---

$$B_m^- = \sum_{k=0}^{m} \sum_{v=0}^{k} (-1)^v \binom{k}{v} \frac{v^m}{k+1}$$

# Primo text editor della Storia?

---

Nel 1890 la tecnologia delle **schede perforate** fu ripresa da **Herman Hollerith**, che costruì una macchina elettro-meccanica per l'analisi delle informazioni statistiche riguardanti il censimento del 1890 effettuato negli **Stati Uniti**.

# La Tabulatrice di Hollerith (1890)

---

Nel $1924$ la sua società, assieme ad altre tre, si unirono per formare la **International Business Machine Corporation** (IBM).

---

## Intorno al $1940$ furono costruiti i primi computers

- ZE machine (1941)
- Atanasoff-Berry Computer (ABC, 1941)
- Elecronic Numerical Integrator and Calculator (ENIAC, 1945)
- **Automatic Computing Engine** (ACE, 1946)
- Elecronic Delay Storage Automatic Calcualtor (EDSAC, 1949)

---

- **TRAnsistorized DIgital Computer** (TRADIC, 1954)
- **Transistorized Experimental computer zero** (TX-0, 1955)
- **Programmed Data Processor-1** (PDP-1, 1959)

L'utilizzo dei primi computers era limitato a personale altamente specializzato, in quanto bisognava manipolare **interruttori** e **spine** (switches and plugs).

Colossus Computer (1943-1945)

La cultura **hacker MIT** è stata formata scrivendo e condividendo codice per il **TX-0** e il suo successore, il **PDP-1**.

---

TX-0 (1956)

Uno dei primi videogiochi digitali, **Spacewar!**, è stato scritto proprio per il **PDP-1** da **Steve Russel** in collaborazione con **Martin Graetz**, **Wayne Wiitanen**, **Bob Saunders**, **Steve Piner**, ed altre persone.

# The **PDP-1** (1962)

---

Nel 1968, **Douglas Engelbart** presentò la

"**Madre di tutte le demo**"

mostrando un utilizzo del computer a cui oramai siamo abituati, con **tastiera**, **mouse** e via dicendo.

La demo è ancora disponibile su **youtube**

https://www.youtube.com/watch?v=yJDv-zdhzMY

Nel $1969$ viene sviluppato **ed**, uno dei primi software per il futuro **Unix** operating system

# ed – Line editor

(Da **ed** è poi derivato **vi**, seguito da **vim**)

# **Utilizzo di ed** (1/2)

---

## Scrivere "`Hello World!`" nel file `hello.txt`

```
[leo@archlinux ~]$ ed
a                   # start append mode
Hello World!        # write text
.                   # stop append mode
w hello.txt         # write to file
13
```

# **Utilizzo di ed** (2/2)

---

## Cambiare "`World`" in "`everyone`"

```
[leo@archlinux ~]$ ed
r test.txt              # read input
12
,p                      # print all lines
hello world
1                       # select first line
hello world
s/world/everyone/       # replace
1                       # see the changes
hello everyone
w test.txt              # write to file
15
```

Un altro text editor di quel tempo era **TECO** , sviluppato nel 1962 da **Dan Murphy** quando era uno studente all'MIT.

---

TECO → Text Editor & COrrector

Dalla pagina su **wikipedia**…

***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.*

# Che cosa sono le **macro**?

# Che cosa sono le **macro**?

---

Sono dei modi per salvare **sequenze arbitrarie di tasti** e per eseguire queste sequenze tramite delle **abbreviazioni**.

# Esempio macro di TECO

---

## Linguaggio nella pratica "**write-only**"

```
!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)

A partire dai comandi di **TECO** sono poi state sviluppate diverse estensioni dell'editor.

Tra queste troviamo **TECMAC** e **TMACS**, sviluppate intorno al $1976$ da **Guy Steele**, **Dave Moon**, **Richard Greenblatt**, **Charles Frankston**, et al.

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

Successivamente, sempre nel $1976$, **Richard Stallman**, **Guy Steele** e **Dave Moon** sviluppano la prima versione di **EMACS**.

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

Dopo questa prima implementazione di **EMACS** ne sono segue tante altre.

---

- 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

Tra tutte, **GNU Emacs** è sicuramente la versione che ha ricevuto più attenzioni ed effort nello sviluppo durante questi anni.

---

# Riguardando indietro

---

Dalle **schede perforate** a **GNU Emacs**.

# EMACS

# Ma quindi, che cos'è Emacs?

Emacs nasce per gestire **contenuti testuali**, da semplici file di testo (**.txt**) a codice (**.c**).

# **Emacs nasce per gestire contenuti testuali** (1/3)

---

# **Emacs nasce per gestire contenuti testuali** (2/3)

---

# **Emacs nasce per gestire contenuti testuali** (3/3)

---

A prima vista quindi Emacs potrebbe sembrare un

**Text editor "potente"**

# Ok… ma, quanto "potente"?

# Ok… ma, quanto "potente"?

---

Tanto, tanto, potente.

Emacs contiene al suo interno un **interprete** per un **linguaggio di programmazione turing-complete**, chiamato **Emacs-Lisp**.

Codice in Emacs-Lisp che calcola l' $n$ esimo numero di Fibonacci.

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

$$F_1 = F_2 = 1$$
$$F_n = F_{n-2} + F_{n-1},\ n \geq 3$$

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

# Esecuzione di codice in Emacs

---

L'integrazione del linguaggio Emacs-Lisp all'interno di Emacs significa che, teoricamente, possiamo implementare **qualsiasi cosa implementabile** e farla girare in Emacs.

L'integrazione del linguaggio Emacs-Lisp all'interno di Emacs significa che, teoricamente, possiamo implementare **qualsiasi cosa implementabile** e farla girare in Emacs.

Anche **snake**.

**Tetris** e **snake**, implementati in Emacs.

---

In ultima analisi quindi, Emacs non è né un **text editor**, né un **sistema operativo**.

# Emacs è un interprete di Emacs-Lisp

# **Il potere di Emacs** (1/3)

---

# **Il potere di Emacs** (2/3)

---

# **Il potere di Emacs** (3/3)

---

# LISP

Originariamente specificato nel 1958 dal team guidato da John McCarthy (MIT), **Lisp** è il secondo linguaggio ad alto livello più vecchio di sempre.

$$\text{Lisp} \longrightarrow \text{LISt Processing}$$

Riferimento: History of LISP

L'influenza che **LISP** ha avuto sui successivi linguaggi di programmazione è notevole.

**LISP** ha introdotto e/o popolarizzato:

- **REPL** (Read Eval Print Loop).
- **Homoiconicity**.
- **Automatic Garbage Collection**.
- **Symbolic oriented**.
- …

Oramai LISP non è più un singolo linguaggio, quanto piuttosto una **famiglia di linguaggi**.

---

# I linguaggi della famiglia LISP sono alquanto semplici da riconoscere…

```
(defun int2bin (i)
  (defun int2bin-rec (i res)
    (cond ((= i 0) (concat "0" res))
          ((= i 1) (concat "1" res))
          ('t
           (int2bin-rec (lsh i -1)
                        (concat (format "%s" (logand i 1))
                                res)))))
  (int2bin-rec i ""))
```

Codice ELISP che, dato un intero, ritorna la rappresentazione binaria dell'intero come stringa

$$(\text{int2bin } 1337) \longrightarrow 10100111001$$

# La tipica reazione di una persona che vede per la prima volta del codice **LISP**: **terrore**!

---

```
(defun my/txt2hex (beg end)
  "opens new buffer *out* which contains the selected text as
well as the hexadecimal bytes of the text."
  (interactive
   (if (use-region-p)
       (list (region-beginning) (region-end))
     (list nil nil)))
  (if (and beg end)
      (progn
        (let ((word (buffer-substring-no-properties beg end)))
          (switch-to-buffer-other-window "*out*")
          (erase-buffer)

          ;; -- first, for each word
          (dotimes (i (length word))
            ;; -- write char and hex
            (insert (format "%c -> 0x%X\n" (aref word i) (aref word i

          (insert "\n")

          ;; -- then, write each hex byte in a row
          (insert "0x")
          (dotimes (i (length word))
            (insert (format "%X" (aref word i))))

          ))))
```

Lisp $\longrightarrow$ Lost In Stupid Parentheses

In ELISP tutto ciò che scriviamo è una **S-expression**.

---

**Def**: Una S-expression può essere:

1. Un **atomo**
2. Una **espressione composta** della forma

$$(x \quad . \quad y)$$

dove $x$ e $y$ sono a loro volta S-expressions.

Due notazioni per scrivere le **S-expressions**:

## DOTTED-PAIR NOTATION

```
("world" . (5 . (2 . NIL)))
```

## LIST NOTATION

```
("world" 5 2)
```

## **Valutare una S-expression** (1/4)

---

Quando vediamo una S-expression dobbiamo sempre avere in mente la seguente immagine:

- Il primo elemento della lista è un **OPERATORE**.
- I restanti elementi sono degli **OPERANDI**.

# **Valutare una S-expression** (2/4)

---

## **Valutare una S-expression** (3/4)

---

L'idea quindi è quella di:

1. Valutare gli operandi uno alla volta.
2. Applicare l'operatore agli operandi.

## **Valutare una S-expression** (4/4)

Gli operandi a loro volta possono essere delle espressioni non atomiche.

```
(+ (* 2 3) (* 4 5))
(+ 6 20)
26
```

Questo tipo di notazione è detta **prefix-notation**, in quanto l'operatore precede sempre gli operandi.

$$+ \ 1 \ 2$$

Tipicamente noi siamo abituati alla **infix-notation**, in cui l'operatore sta al centro degli operandi.

$$1 + 2$$

## Osservazione

La difficoltà nel leggere codice LISP ha molto a che fare con le nostre abitudini che con il codice LISP in sé.

All'inizio sembra tremendamente difficile leggere codice LISP semplicemente perché **il modo di LISP è diverso**.

# SULLA OMOICONICITÀ DI LISP

I **linguaggi di programmazione** sono implementati tramite tecniche di **traduzione**:

- $L_1$, linguaggio espressivo e semplice da utilizzare.
- $L_0$, linguaggio efficiente ma difficile da utilizzare.

$$L_1 \longrightarrow L_0$$

# **Esempio**: C ⟶ x86-64 assembly

```c
#include <stdio.h>

int main(void) {
  printf("Hello World!\n");
  return 0;
}
```

Sorgente C

```
push   rbp
mov    rbp,rsp
sub    rsp,0x10
mov    DWORD PTR [rbp-0x4],edi
mov    QWORD PTR [rbp-0x10],rsi
lea    rax,[rip+0xeb5]
mov    rdi,rax
call   0x555555555030 <puts@plt>
mov    eax,0x0
leave
ret
```

Codice assembly

Svariati approcci per implementare questa **traduzione**:

- **Interpretazione** (Python)
- **Compilazione** (C)
- etc..

**In generale sono necessari almeno:**

---

**Lexer**

$$\text{codice sorgente} \longrightarrow \text{sequenza di tokens}$$

---

**Parser**

$$\text{sequenza di tokens} \longrightarrow \text{abstract syntax tree (AS}$$

L'**abstract syntax tree** è una struttura alborea che contiene le informazioni sintattiche relative al codice scritto dal programmatore.

Questa struttura viene poi utilizzata per interpretare o compilare il codice.

# L'abstract Syntax Tree è tipicamente molto diverso dal codice iniziale

---

Source: Abstract syntax tree for Euclidean algorithm

```
while b ≠ 0:
    if a > b:
        a := a - b
    else:
        b := b - a
return a
```

Codice sorgente

Nella maggior parte dei linguaggi di programmazione:

**Le strutture esterne, visibili dal programmatore, e quelle interne, utilizzate dall'interprete per eseguire il codice, sono diverse tra loro**.

---

In **LISP**, questa differenza non è necessaria.

Questa è l'essenza della **omoiconicità**, in inglese **homoiconicity**.

---

$$\text{homo} \longrightarrow \text{the same}$$

$$\text{icon} \longrightarrow \text{representatio}$$

In un linguaggio omoiconico, le rappresentazioni interne e quelle esterne sono uguali.

L'utilizzo della parola "uguale" nella definizione appena data dovrebbe essere interpretata con le giuste considerazioni del caso.

Consiglio le seguenti letture per farsi un'idea più chiara in merito a ciò che si intende per omoiconicità:

- Expression of change – Don't say “Homoiconic”
- Expression of change – Homoiconicity revisited
- Hacker News thread on homoiconicity

Conseguenza della omoiconicità:

---

# È possibile scrivere un interprete di LISP utilizzando LISP in modo intuitivo ed elegante.

# codice LISP per interpretare codice LISP

## (Paul Graham – Roots of Lisp)

---

```
(defun my-eval (e a)
  (cond
   ((atom e) (cadr (assoc e a)))
   ((atom (car e))
    (cond
     ((eq (car e) 'quote) (cadr e))
     ((eq (car e) 'atom) (atom (my-eval (cadr e) a)))
     ((eq (car e) 'eq) (eq (my-eval (cadr e) a)
                           (my-eval (caddr e) a)))
     ((eq (car e) 'car) (car (my-eval (cadr e) a)))
     ((eq (car e) 'cdr) (cdr (my-eval (cadr e) a)))
     ((eq (car e) 'cons) (cons (my-eval (cadr e) a)
                               (my-eval (caddr e) a)))
     ((eq (car e) 'cond) (my-evcon (cdr e) a))
     ('t
      (progn
        (my-eval (cons (cadr (assoc (car e) a))
                       (cdr e))
                 a)))))

   ((eq (caar e) 'lambda)
    (my-eval (caddar e)
          (append (my-pair (cadar e)
                           (my-evlis (cdr e) a))
                  a)))))
```

```
(defun my-evcon (c a)
  (cond ((my-eval (caar c) a) (my-eval (cadar c) a))
        ('t (my-evcon (cdr c) a))))

(defun my-evlis (m a)
  (cond ((null m) '())
        ('t (cons (my-eval (car m) a)
                  (my-evlis (cdr m) a)))))

(defun my-pair (x y)
  (cond ((and (null x) (null y)) '())
        ((and (not (atom x)) (not (atom y)))
         (cons (list (car x) (car y))
               (my-pair (cdr x) (cdr y))))))
```

# LISP che esegue LISP

# ORG-MODE

La flessibilità offerta dall'abilità di eseguire codice **Emacs-Lisp** all'interno di **Emacs** è mostrata brillantemente da **org-mode**

**Org-mode** nasce nel 2003 da **Carsten Dominik**.

Source: Carsten Dominik blog

**Org-mode** può essere principalmente utilizzato per:

- **Document outlining**.
- **Personal Information Management (PIM)**.
- **Literate Programming**.
- **Publishing/Exporting tool**.

**Org-mode** può essere visto da due punti di vista diversi:

1. Come **markup language**.
2. Come **major-mode** in Emacs.

I file **org** sono **plaintext** files che processati in modo dinamico da Emacs.

# Esempio file org

La particolare sintassi utilizzata è detta **Org-Mode Markup Language**, e rappresenta la parte più importante di org-mode.

Org Mode Is One of the Most Reasonable Markup Languages to Use for Text

A partire del 2006 **org-mode** è strato introdotto come major-mode all'interno di Emacs.

**Bastien Guerry** è il maintainer corrente di org-mode.

# ESEMPI PRATICI

Vediamo adesso Emacs in azione…

# 1 – EDITING CON LE MACRO

# Abbiamo un file **csv** con dei dati

| name | email | password |
|---|---|---|
| Lance Davis | hollowaydanielle@example.net | b2a3479d1e3fea6baba165b5dd9da869727806f8 |
| Joseph Thompson | stevensonalexander@example.net | b07652ec2e19f253db09298a5253fc56e478ec8b |
| Joseph Benitez | jonathan08@example.net | 9ae29fafe618aaf5d291afdf7cbf3cd4a99e3e17 |
| … | … | … |

# Vogliamo trasformare ogni riga **csv** in un oggetto **json**

```
Lance Davis, hollowaydanielle@example.net, b2a3479d1e3fea6baba165b5dd9da869727806f8
```

```
{
  "nome": "Lance Davis",
  "email": "hollowaydanielle@example.net",
  "password": "b2a3479d1e3fea6baba165b5dd9da869727806f8"
}
```

# Il fatto è che abbiamo **tanti records**:

---

```
name,email,password
Lance Davis,hollowaydanielle@example.net,b2a3479d1e3fea6baba165b5dd9da869727806f8
Joseph Thompson,stevensonalexander@example.net,b07652ec2e19f253db09298a5253fc56e478
Joseph Benitez,jonathan08@example.net,9ae29fafe618aaf5d291afdf7cbf3cd4a99e3e17
Sheri Pierce,perezsharon@example.com,b1d872bf4221e12c8ab4c098a6bc9d981292f297
Peggy Mitchell,valerie05@example.org,927d5b8628635f098cc37844d4ad58f59caf8263
Tony Ingram,florespamela@example.net,d9e66f4d65d785470ee61a74af888bde7b1ef6e1
Brian Huber,logansmith@example.org,cdf9c7dfd7b566f69923d87ce0d72f62573ceef5
...
```

Situazione scomoda:

- Troppi record da trasformare a mano.
- Troppo pochi per scrivere un programma da zero.

Situazione scomoda:

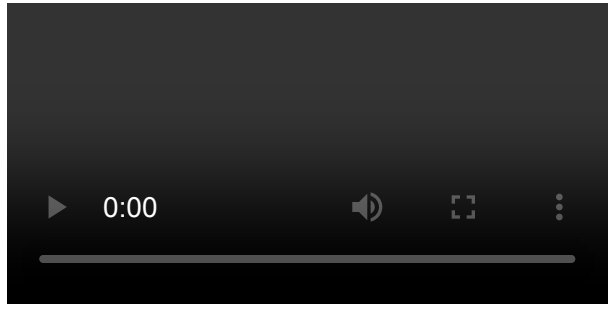
- Troppi record da trasformare a mano.
- Troppo pochi per scrivere un programma da zero.

**Problema perfetto per le macro**!

**IDEA**: Registrare una **sequenza di comandi per trasformare una riga arbitraria del CSV nel formato JSON**, e ripetere i comandi per trasformare tutte le righe.

# LIVE DEMO

# PLAYBACK VIDEO



# 2 – PROGRAMMAZIONE ELISP

# Consideriamo il seguente codice elisp

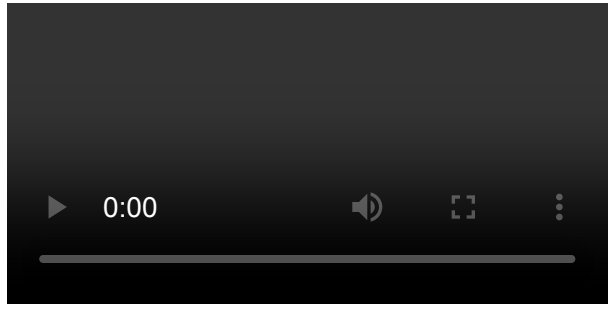
```
(defun my/txt2hex (beg end)
  "opens new buffer *out* which contains the selected text as
well as the hexadecimal bytes of the text."
  (interactive
   (if (use-region-p)
       (list (region-beginning) (region-end))
     (list nil nil)))
  (if (and beg end)
      (progn
        (let ((word (buffer-substring-no-properties beg end)))
          (switch-to-buffer-other-window "*out*")
          (erase-buffer)
          ;; -- first, for each word
          (dotimes (i (length word))
            ;; -- write char and hex
            (insert (format "%c -> 0x%X\n" (aref word i) (aref word i))))
          (insert "\n")
          ;; -- then, write each hex byte in a row
          (insert "0x")
          (dotimes (i (length word))
            (insert (format "%X" (aref word i)))))))))
```

Tale codice sta introducendo una nuova funzione, la funzione `my/txt2hex`, che può essere utilizzata in ogni momento per calcolare la rappresentazione esadecimale di una sequenza di caratteri.

# Esempio di my/txt2hex

# LIVE DEMO

# PLAYBACK VIDEO



# 3 – SLIDES IN ORG-MODE

Le slides che sto presentando ora, in questo momento, sono state create utilizzando una serie di programmi, tra cui **Emacs**.

---

# File sorgente org

# File finale html+js+css

---

# 4 – SITO WEB IN ORG-MODE

## **Gestione materiale accademico**:

---

- Sito statico con **hugo**
- Interfaccia emacs con **org-mode** e **ox-hugo**

## Workflow

1. Scrittura in org-mode
2. Esportazione **.org** $\rightarrow$ **.md** con **ox-hugo**
3. Esportazione **.md** $\rightarrow$ **.html** con **hugo**
4. Sito in cartella **public**

# LIVE DEMO

## Per maggiori informazioni

---

- Youtube – Come gestisco un BLOG con un SOLO file ORG MODE
- Blog post – How I manage my blog

# CONCLUSIONI

**Emacs** può essere utilizzato come un ulteriore layer di astrazione dal sottostante sistema operativo.

- Emacs
- OS
- kernel
- hardware

Fondamentalmente rappresenta **una interfaccia testuale 2D programmabile tramite Emacs-Lisp**.

Nel 1981 **Richard Stallman** scrive

---

# Abstract del paper

***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.*

L'aspetto più importante che **Stallman** sottolinea varie volte è proprio **l'abilità di estendere Emacs tramite del codice Emacs-Lisp**.

# **L'importanza di 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.*

## **L'importanza di 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.*

Altri aspetti fondamentali di Emacs:

- **Self-documentation**
- **Package archives (MELPA, GNU ELPA)**
- **Community**

# SULLA COMMUNITY...

L'importanza di avere una forte **community** fatta da persone che **utilizzano Emacs ogni giorno** non può essere messa da parte.

Se **Emacs** è riuscito a sopravvivere tutti questi anni, questa sorprendente sopravvivenza è stata resa possibile solamente **dalle persone che lo continuano a sviluppare e utilizzare ogni giorno**.

Perché col passare del tempo le cose si dimenticano se non vengono più utilizzate.

## **Alcuni link della community** (1/3)

- Emacs Official Documentation
- Sacha Chua
- Protesilaos Stavrou
- SystemCrafters
- emacswiki
- awesome-emacs
- emacsconf
- ergoemacs

## **Alcuni link della community** (2/3)

---

Stop jumping between note-taking tools: use org-roam

# **Alcuni link della community** (3/3)

---

https://academy.leonardotamiano.xyz/

# Comunità di Emacs italiane?

# LA GUERRA È FINITA?

***The editor war is the rivalry between users of the Emacs and vi*** *(now usually Vim, or more recently Neovim) text editors. The rivalry has become a lasting part of hacker culture and the free software community. (Wikipedia)*

# The Editor War

---

Oramai poco combattuta.

Oramai ci sono altre priorità, altri software.

Con **evil-mode** Emacs diventa simile a vi!

---

https://github.com/emacs-evil/evil

Tempo di crescere.

# QA?