




Introduzione ad Emacs

Premesse sul documento

Questo documento è tratto da Leonardo Tamiano, lui mi ha dato lo spunto per scrivere questo documento, che farà parte del mio archivio personale di appunti. In seguito io ho scelto di fare le mie ricerche personali per approfondire degli aspetti, consiglio anche di approfondire da:

The Emacs Editor

Emacs is the advanced, extensible, customizable, self-documenting editor. This manual describes how to edit with Emacs and some of the ways to customize it; it corresponds to GNU Emacs version 28.1. The GNU Emacs website is at <https://www.gnu.org/software/emacs/>.

 https://www.gnu.org/software/emacs/manual/html_node/emacs/

Introduzione al Text Editor Emacs

La prima versione pubblica di **Emacs** (versione 13) fu rilasciata il 20 Marzo del 1985 come il primo software del **Gnu Project**, iniziativa fondata da *Richard Stallman* nel 1978.

Nonostante sia un software abbastanza “vecchio” non è affatto obsoleto, anzi, tutt’ora ha un grosso seguito. A prima vista Emacs può essere descritto come un **text editor** molto potente. **Emacs** infatti

nasce per gestire contenuti testali di ogni natura. Essendo un **text editor**, esso può ricoprire tutti quei campi della scrittura, iniziando dall'uso di uno scrittore che può usare **Emacs** per la stesura di una sua opera fino al programmatore più classico che deve scrivere un programma.

Definire **Emacs** solo un text editor, dunque un programma per la scrittura è però limitante, questo perchè al suo interno ha un interprete per il linguaggio elisp. Ecco un esempio di codice **Elisp**:

Codice elisp che calcola l' n esimo numero di Fibonacci.

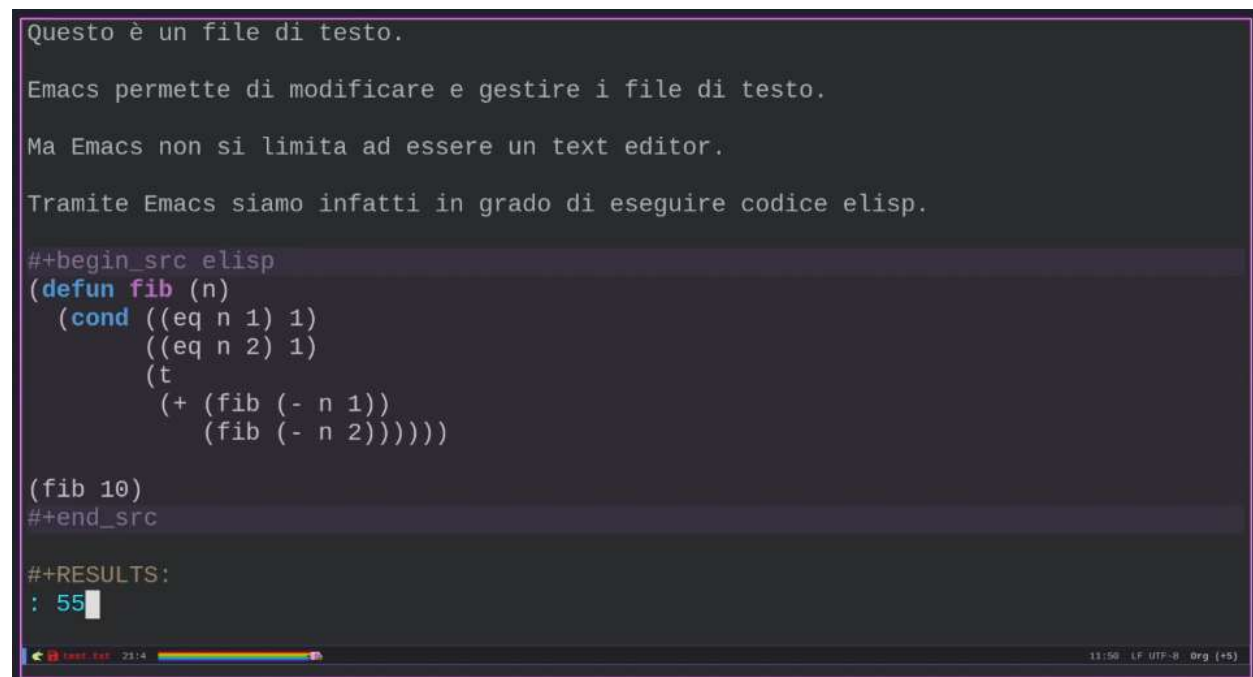
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

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

$$F_1 = F_2 = 1$$

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

Ecco un esempio pratico di utilizzo di Emacs:



```
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.

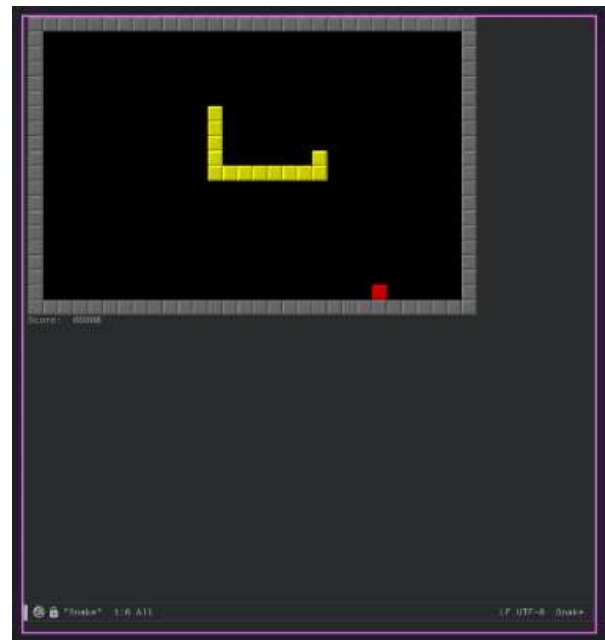
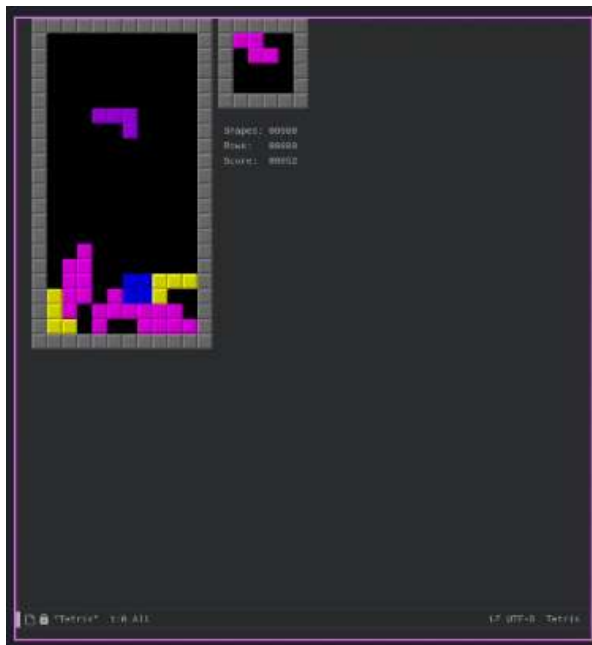
#+begin_src elisp
(defun fib (n)
  (cond ((eq n 1) 1)
        ((eq n 2) 1)
        (t
         (+ (fib (- n 1))
            (fib (- n 2))))))

(fib 10)
#+end_src

#+RESULTS:
: 55
```

Abbiamo un **buffer** (vedremo più avanti il significato di questo) con del testo, una sezione con il codice elisp e in fine un'ulteriore sezione con il risultato del codice precedente. Dunque come possiamo notare Emacs non è semplice editor di testo, ma è un qualcosa di più potente.

È proprio l'abilità di eseguire codice elisp (linguaggio **Turing-Completo**) ciò che rende **Emacs** estremamente flessibile e potente. Elisp viene definito un linguaggio **Turing-Completo** perchè con esso è possibile teoricamente implementare qualsiasi cosa che un computer può calcolare.



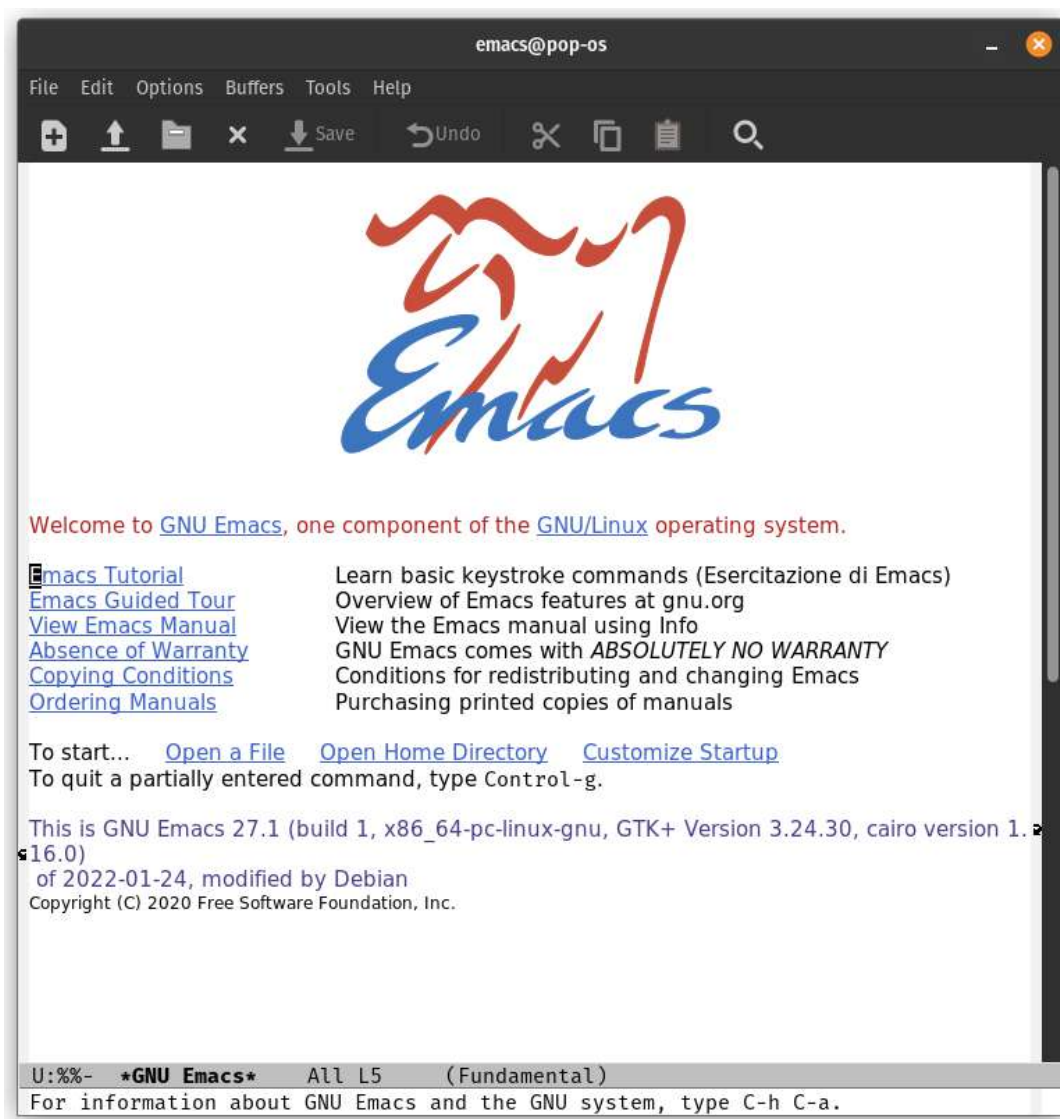
Qui vediamo del codice **Elisp** che implementa rispettivamente il gioco *Tetris* e *Snake*, ed è possibile eseguire questo codice nello stesso **Emacs**.

Imparare bene **Emacs** significa principalmente capire come configurarlo in modo da ottenere un comportamento che si avvicini il più possibile a quello desiderato. Cosa vuol dire questo? Coloro che hanno in origine scritto Emacs, hanno implementato determinate funzioni, col passare degli anni, gli utenti che usano questo software possono trovarsi davanti a problemi che il software non sa risolvere. Ma ciò non è una limitazione anzi, grazie al fatto che **Emacs** è configurabile

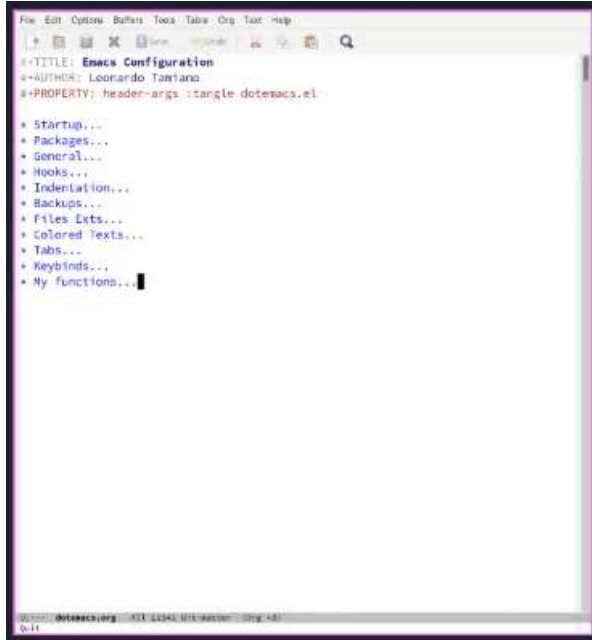
ed esteso con **Elisp** l'utente finale può scrivere il proprio codice Elisp che implementa il modo per risolvere i propri problemi.

Possiamo sia scrivere noi il nostro codice, o in caso non lo sapessimo fare, possiamo utilizzare del codice elisp scritto da altre persone e disponibile in archivi pubblici, come ad esempio l'archivio [MELPA](#).

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



Investendo abbastanza tempo ed impegno però è possibile trasformarlo in ciò che si vuole. Ecco due immagini del **file di configurazione** iniziale di **Emacs**:



Andiamo a vedere ora 3 modi di utilizzo di Emacs.

Interfacciamento OS

Qui vedremo tutte quelle estensioni/pacchetti per la gestione e ottimizzazioni del sistema operativo. **Emacs** offre le tipiche funzionalità di un **Terminal Emulator** tramite vari pacchetti, come:

- vterm
- term-mode
- shell-mode
- eshell

È molto potente questa funzionalità perchè ci permette di ad esempio di eseguire un certo comando, ed essendo in un text editor salvare in un file l'output di tale comando.

Tramite il pacchetto *dired* che è un **File Explorer** invece è possibile spostare, creare, modificare, eliminare e rinominare file e cartelle.

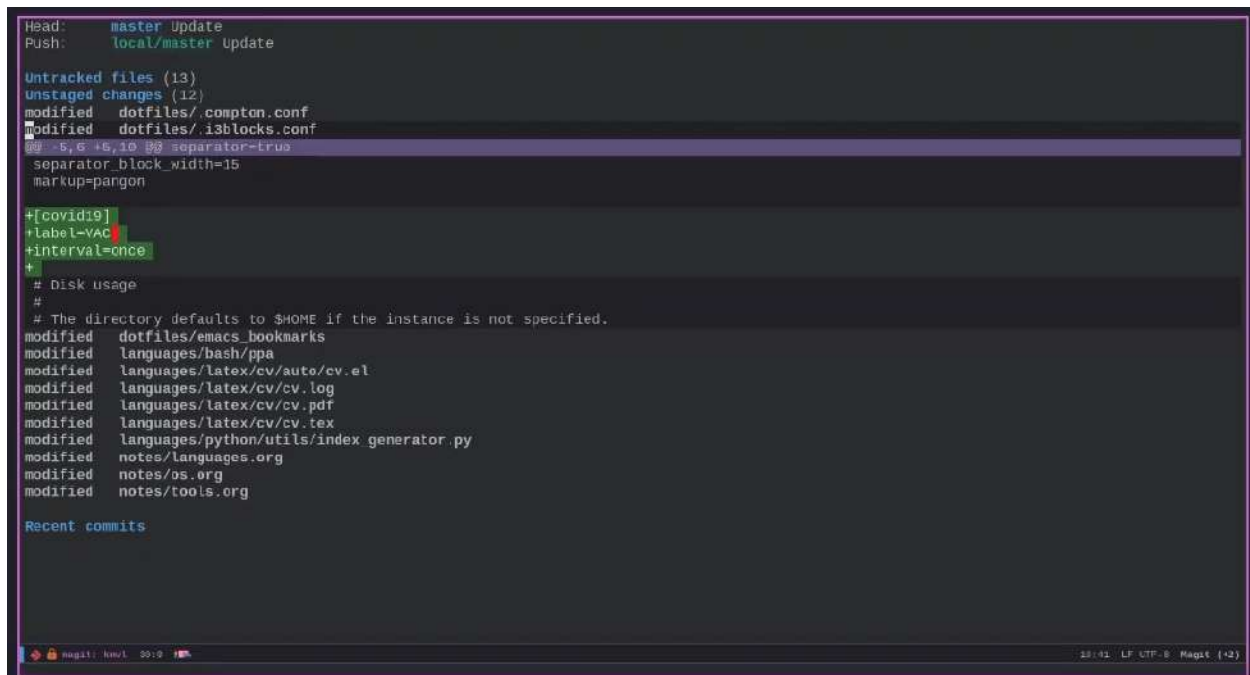
Inoltre è possibile avere anche un Email Manager che gestisce le email, questo mediante i pacchetti:

- mu4e
- gnus
- notmich

Programmazione

La flessibilità di Emacs offre svariate occasioni per supportare e facilitare le tipiche task svolte da un programmatore.

Il pacchetto **magit**, ad esempio, offre una comoda ed intuitiva interfaccia a *git*, il famoso software di versionamento distribuito:



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

Untracked files (13)
Unstaged changes (12)
modified dotfiles/compton.conf
modified dotfiles/isblocks.conf
@@ -5,6 +5,10 @@ separator=Erwa
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
```

Ogni linguaggio di programmazione poi ha la sua particolare modalità.

Org-Mode

Tra tutte le funzionalità di Emacs, **org-mode** è una delle migliori, in quanto esistono svariati use-cases in cui utilizzare i file orgs.

I file scritti in **org-mode** sono semplici file di testo che vengono processati in modo dinamico da 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
```

```
#+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...
```

In generale le potenzialità offerte da **Emacs** per aumentare la propria produttività sono tantissime, e sono tanti gli insegnamenti che è possibile estrarre dall'utilizzo di queste tecnologie.

Dato che **Emacs** è un interprete del linguaggio Elisp per imparare bene **Emacs** dobbiamo anche imparare la famiglia di linguaggi LISP che presentano molte caratteristiche tremendamente affascinanti.

Più è difficile una determinata cosa, e più imparare quella cosa ci fa crescere, direttamente e indirettamente. Sotto questo punto di vista, imparare Emacs apre la mente alle vere potenzialità offerte dal software.

Imparare le basi di Emacs

Emacs è un software complesso ed è difficile, se non del tutto impossibile, descrivere *"il modo migliore"* per imparare **Emacs**.

Per tutti coloro che sono veramente intenzionati ad imparare **Emacs** è infatti condizione necessaria assumersi la propria responsabilità e crearsi il proprio personale percorso di apprendimento.

Ogni cosa complessa è ottenuta combinando tra loro molte cose semplici.

Installazione

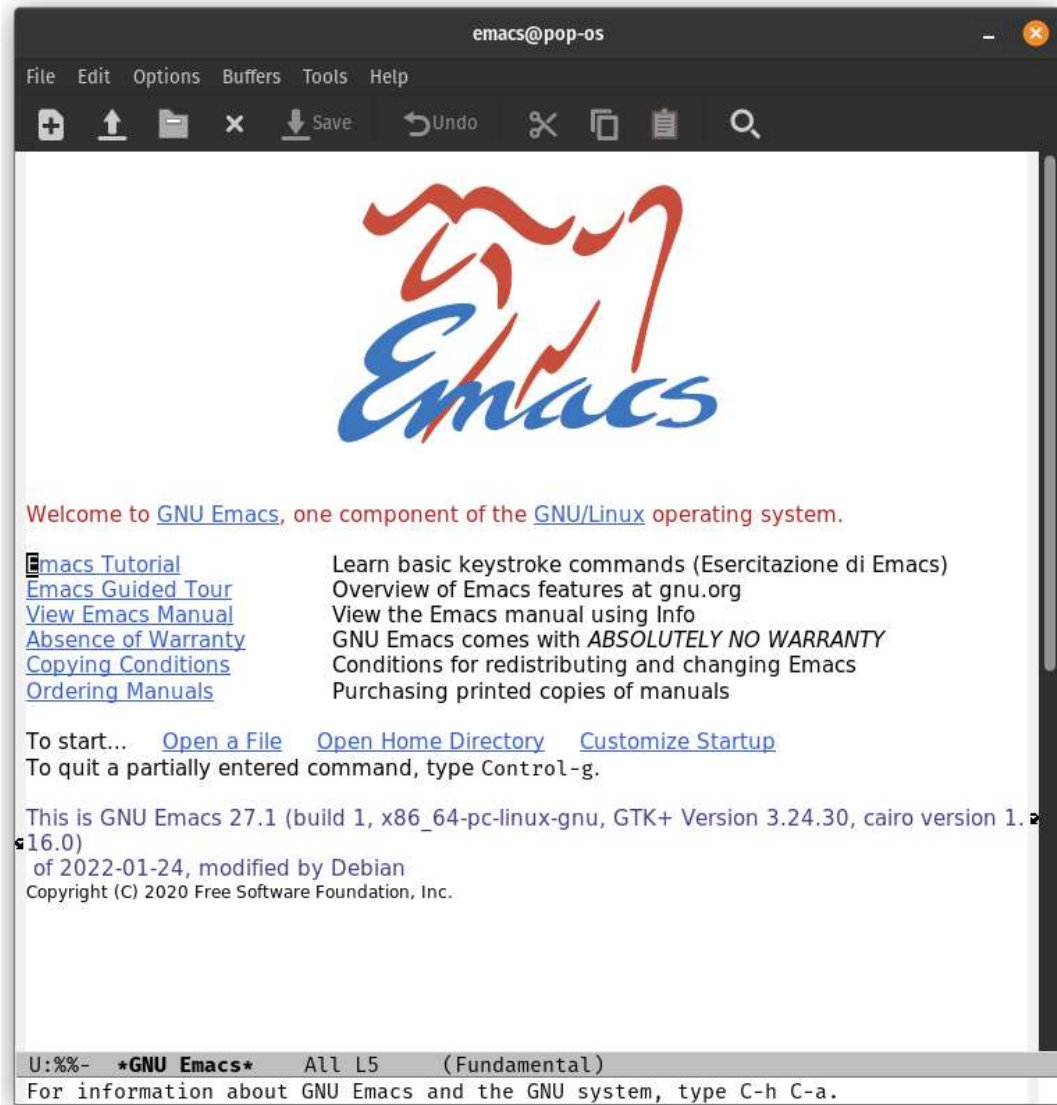
Per installare **Emacs** su sistemi operativi Ubuntu-like useremo il package manager **apt** come segue:

```
sudo apt update
sudo apt install emacs
```

Anche se è possibile usare **Emacs** su **Windows**, è più difficile raggiungere dei buoni livelli di integrazione con il sottostante sistema operativo. Per questo motivo e per altre ragioni, è consigliato l'utilizzo di un sistema operativo *UNIX-like* per imparare **Emacs**.

A prescindere da dove lo installiamo, Emacs può essere fatto partire in due modi diversi:

1) Chiamando normalmente il comando "**emacs**" è possibile aprire una finestra grafica contenete emacs (o anche usando la scorciatoia che si crea in fase di installazione):

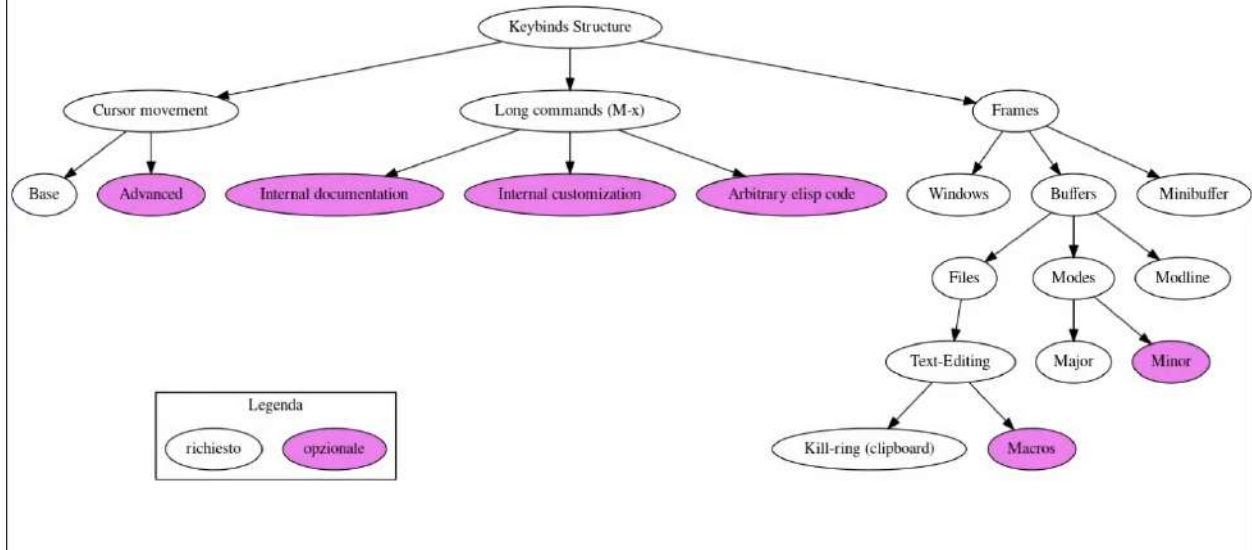


2) Tramite il comando “**emacs -nw**” dentro un terminale chiamiamo la versione *cli* di **Emacs**.

È consigliata la GUI.

Di seguito vedremo uno schema introduttivo che schematizza i concetti fondamentali per approcciarsi all’apprendimento di **Emacs**, tale schema è stato creato soggettivamente, dall’esperienza personale di *Leonardo Tamiano*, infatti lo schema non è concepito per essere nè completo nè generale. È un punto di partenza:

Emacs Baseline Concepts



Questa struttura ad albero ha diversi nodi che rappresentano i diversi concetti all'interno di **Emacs**, le frecce indicano una propedeuticità tra gli argomenti. Ovviamente va letto dall'alto verso il basso. Andiamo ora a fare una panoramica di questi nodi, poi a tempo debito li tratteremo tutti nello specifico.

In cima all'albero troviamo i **Keybinds Structure**, ossia la struttura dei keybinds, dei tasti da premere per eseguire i vari comandi. Emacs è pieno di keybinds e non li tratteremo tutti, ma solo quelli più importanti. Supponiamo di voler aprire un file sul nostro desktop, un semplice file `.txt` con del testo, per farlo usiamo la combinazioni di tasti `Ctrl + x Ctrl + f`:

```
U:%%- *GNU Emacs* All L5 (Fundamental)
Find file: ~/Scrivania/
```

Si aprirà questo finder e qui possiamo scrivere il nome del nostro file, ad esempio `test.txt`, premendo `ENTER` dopo aver inserito il nome del nostro file ci verrà aperto:



Nota: è possibile usare col tasto *TAB* anche l'autocompletamento in fase di inserimento del nome del file.

La notazione in Emacs per descrivere la combinazione *Ctrl + x Ctrl + f* è la seguente: *C-x C-f*.

Il **Cursor movement** indica il movimento del cursore, esso può essere:

- Base
- Avanzato

Per movimento base si indicano i movimenti: alto, basso, sinistra e destra. In Emacs per effettuare questi movimenti ci sono i seguenti **keybinds base**:

- *Ctrl - p / C-p* : vai in alto
- *Ctrl - n / C-n* : vai in basso
- *Ctrl - b / C-b* : vai a sinistra
- *Ctrl - f / C-f* : vai a destra

I **keybinds avanzati** ci permettono di muoverci in modo più flessibile sono:

- `Ctrl - e / C-e` : vai a fine riga
- `Ctrl - a / C-a` : vai a inizio riga
- `Alt - f / M-f` : spostati avanti di una parola
- `Alt - b / M-b` : spostati indietro di una parola
- `Alt - e / M-e` : spostati a fine paragrafo
- `Alt - a / M-a` : spostati a inizio paragrafo

Alcuni comandi non sono assegnati a nessuno keybinds, essi prendono il nome di **Long Commands**, ed è possibile digitarli premendo `Alt - x / M-x`.

I **Long Commands** ci permettono di digitare dei comandi nel minibuffer e aprono la strada a diverse vie, come possiamo notare dallo schema. Tra le tre vie che possiamo intraprendere, quella più interessante è la **Arbitrary Elisp Code**, questo ci permette di scrivere la nostra funzione Elisp e possiamo renderla disponibile al sistema dei Long Commands così da poterla usare quando vogliamo.

Nota: Usando i **Long Commands**, e digitando `quit`, andremo alla Window precedente. Mentre con `quit-window` chiuderemo la Window attuale.

Un altro componente utile è la **Internal documentation**, essa ci permette di leggere la documentazione di un certo comando laddove non lo conosciamo, per aprirlo digitiamo `Ctrl + h / C-h`, digitiamo poi il comando da voler ricercare, e una volta premuto ENTER ci comparirà la documentazione interna (in questo caso di **find-file**):

```
U:--- test.txt All L3 (Text)
```

```
find-file is an interactive compiled Lisp function in 'files.el'.
```

```
It is bound to <open>, C-x C-f, <menu-bar> <file> <new-file>.
```

```
(find-file FILENAME &optional WILDCARDS)
```

```
Probably introduced at or before Emacs version 1.5.
```

```
Edit file FILENAME.
```

```
Switch to a buffer visiting file FILENAME,  
creating one if none already exists.
```

```
Interactively, the default if you just type RET is the current directory,  
but the visited file name is available through the minibuffer history:
```

```
type M-x next-history-element to pull it into the minibuffer.
```

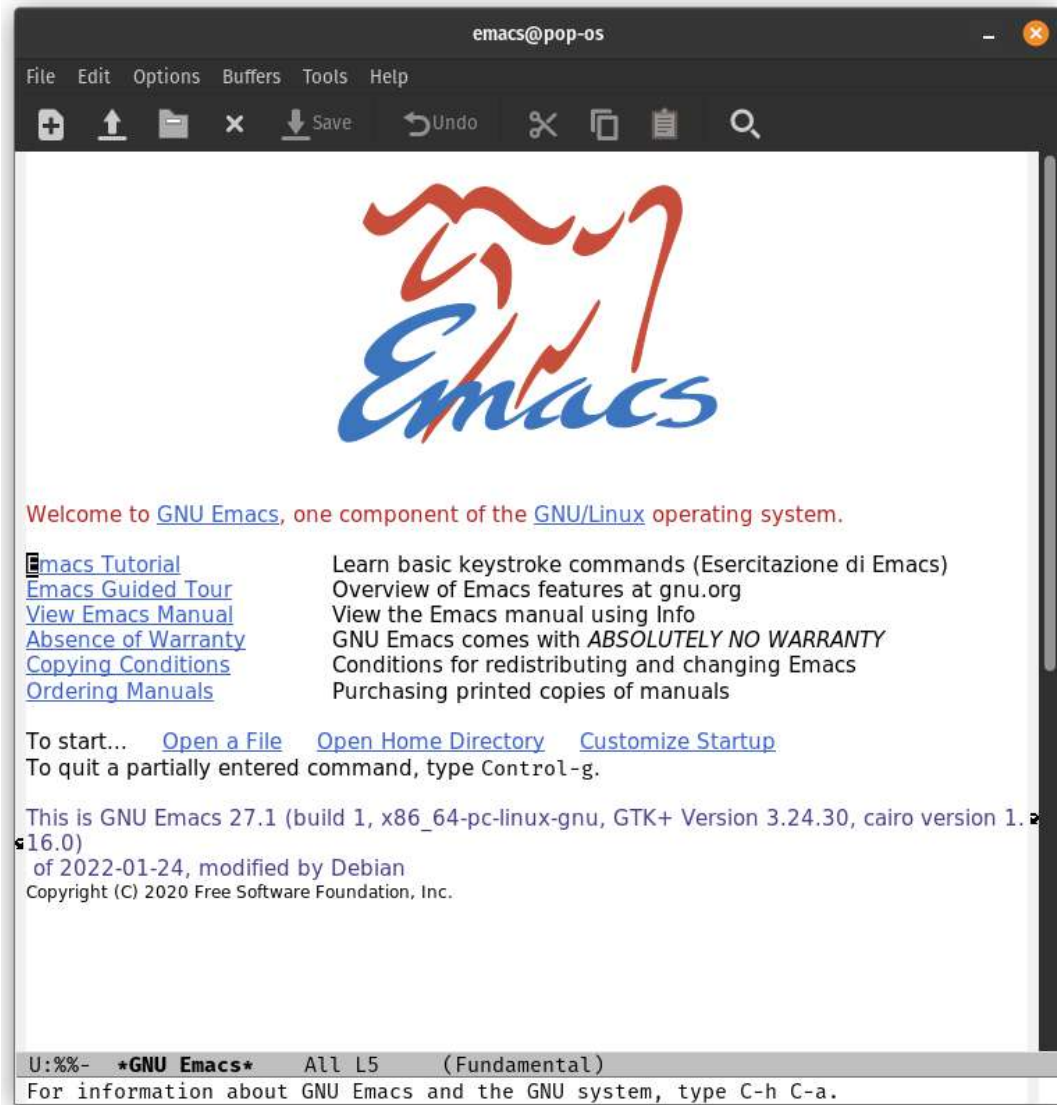
```
The first time M-x next-history-element is used after Emacs prompts for  
the file name, the result is affected by 'file-name-at-point-functions',  
which by default try to guess the file name by looking at point in the  
current buffer. Customize the value of 'file-name-at-point-functions'  
or set it to nil, if you want only the visited file name and the  
current directory to be available on first M-x next-history-element
```

```
U:%%- *Help* Top L1 (Help)
```

Ci mostra una descrizione e anche i keybinds da usare richiamarlo (in questo caso Ctrl - x Ctrl - f).

In fine, per finire questo ramo, vi è **l'Internal customization** che ci permette di personalizzare il nostro Emacs, per iniziare è ok ma vedremo più avanti che scrivere il nostro codice Elisp è più ottimale.

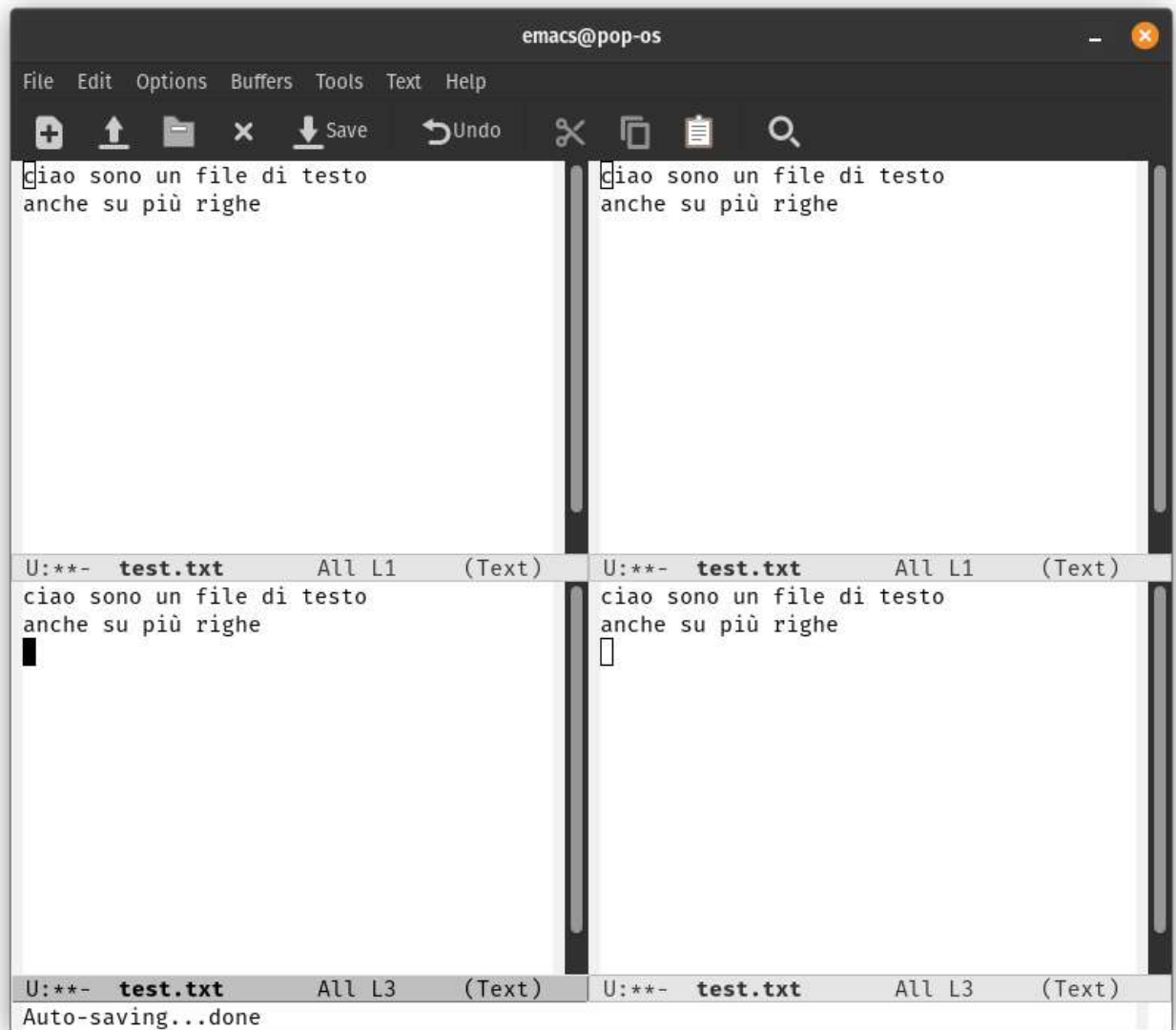
Passiamo ora all'altro ramo, partiamo con il concetto di **Frame**, esso è semplice, è il componente che contiene tutti i buffer di Emacs, in modo brutale, la finestra di Emacs:



Un singolo **Frame**, può essere spezzettato in più **Windows**, questo mediante i keybinds:

- Per spezzare il Frame orizzontalmente: `Ctrl - x - 2`
- Per spezzare il Frame verticalmente : `Ctrl - x - 3`
- Per tornare ad un Frame: `Ctrl - x - 1`

Iterando questi split possiamo avere quante **Windows** voglio. Questo è utile per fare in modo che abbiamo Windows diverse che possono mostrare cose diverse:



In questo caso abbiamo 4 Windows che mostrano lo stesso **Buffer**.

Nota: Un buffer può essere ingrandito o rimpicciolito facendo *C-Scroll Up or Down* del mouse.

Un **buffer** è il contenuto di una particolare entità mostrato in una Window, o meglio possiamo dire che: Un **buffer** è un oggetto Lisp contenente del testo da modificare. I buffer vengono utilizzati per contenere il contenuto dei file che vengono visitati; potrebbero esserci anche buffer che non stanno visitando i file. Il **Minibuffer** è un particolare buffer che **Emacs** usa per comunicare con noi, come abbiamo visto in precedenza:

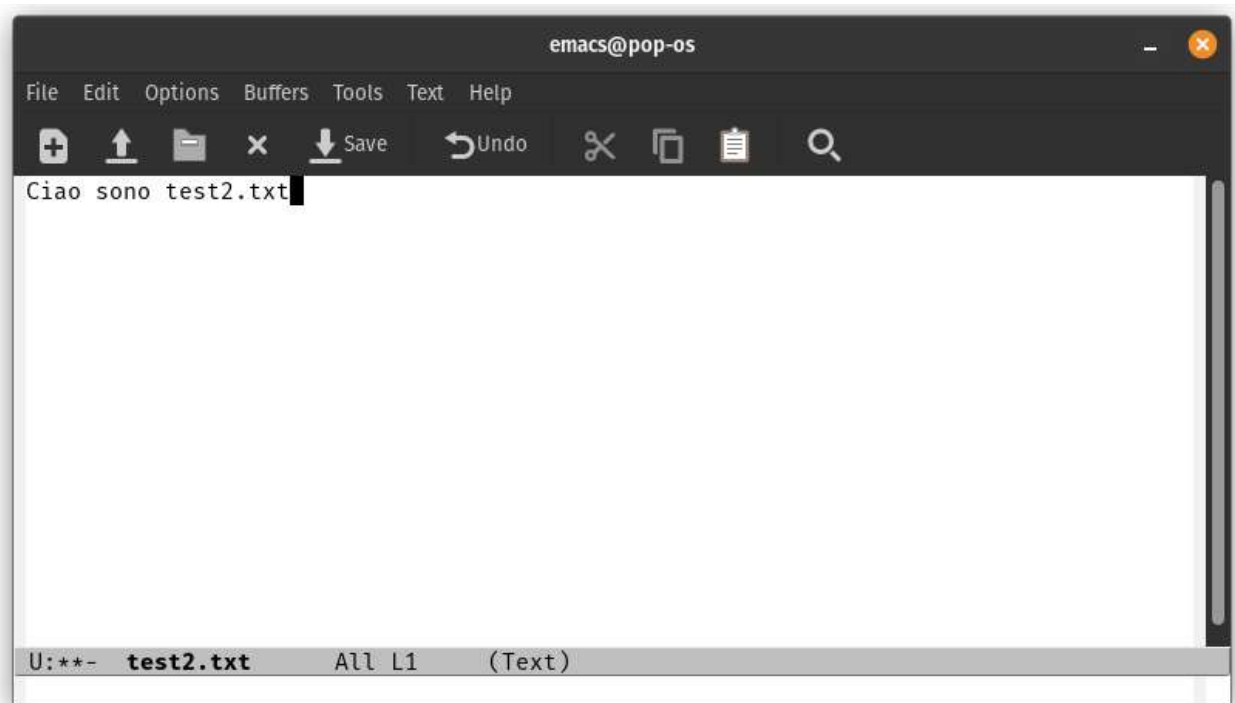
```
U:**- test.txt All L3 (Text)
Find file: ~/Scrivania/test.txt
```

Questo è un esempio con il **find-file**, questo piccolo riquadro è il **Minibuffer**.

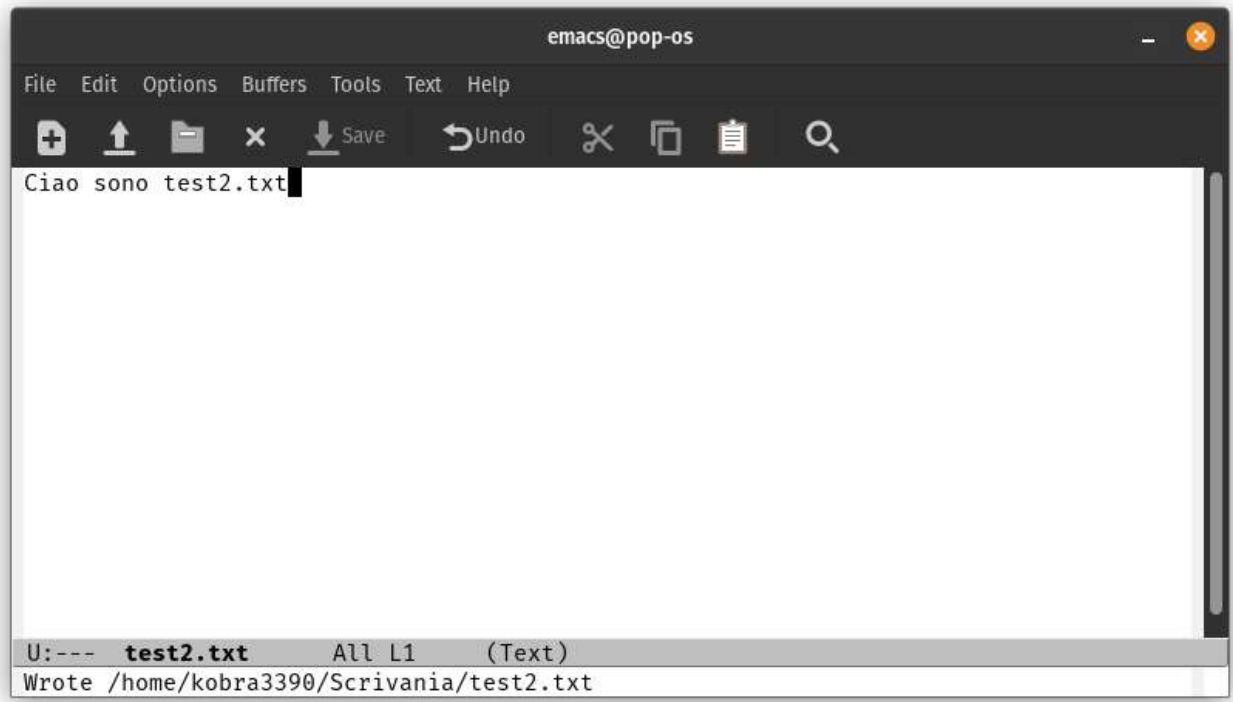
Un **Buffer** è associato ad una serie di informazioni, quali:

- **Files**
- **Modes**
- **Modline**

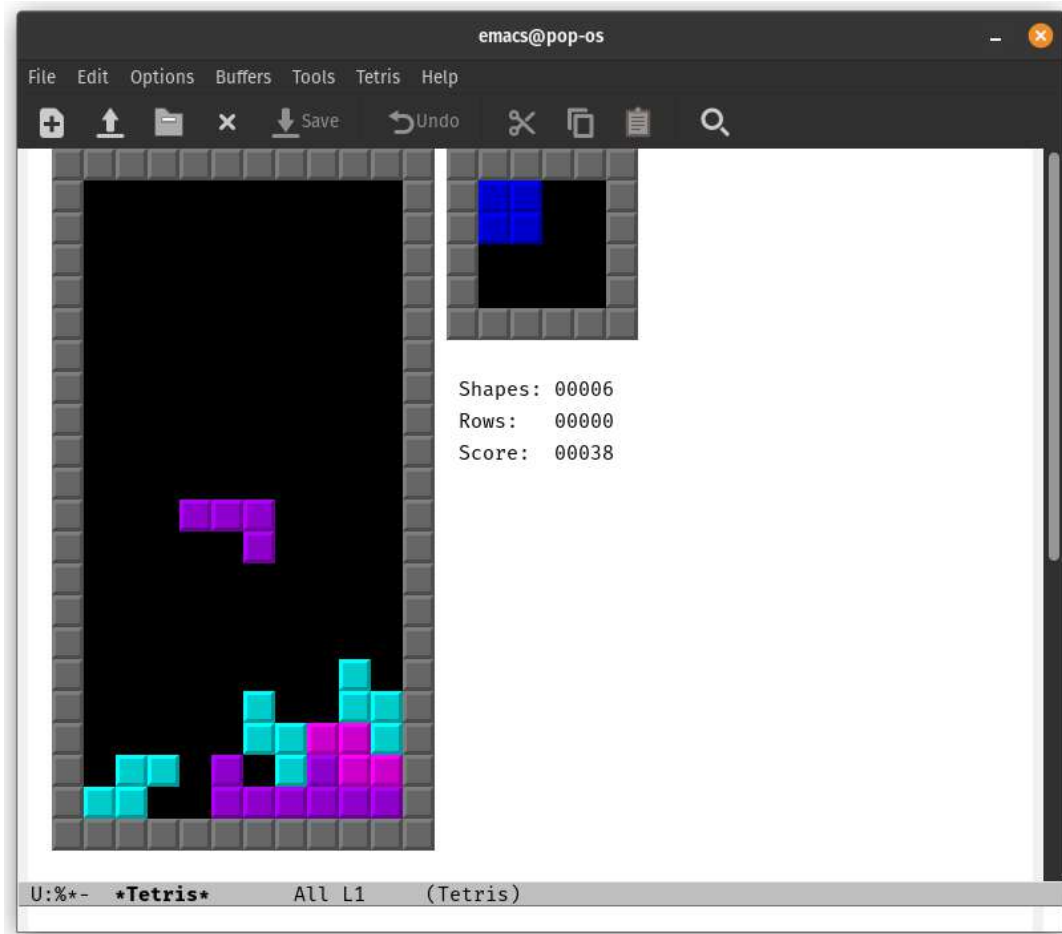
Un Buffer può essere associato ad un file, ma non necessariamente, che vuol dire? Supponiamo mediante il `find-file` cerchiamo il file `test2.txt` (non esistendo, lo crea), e scriviamo qualcosa dentro:



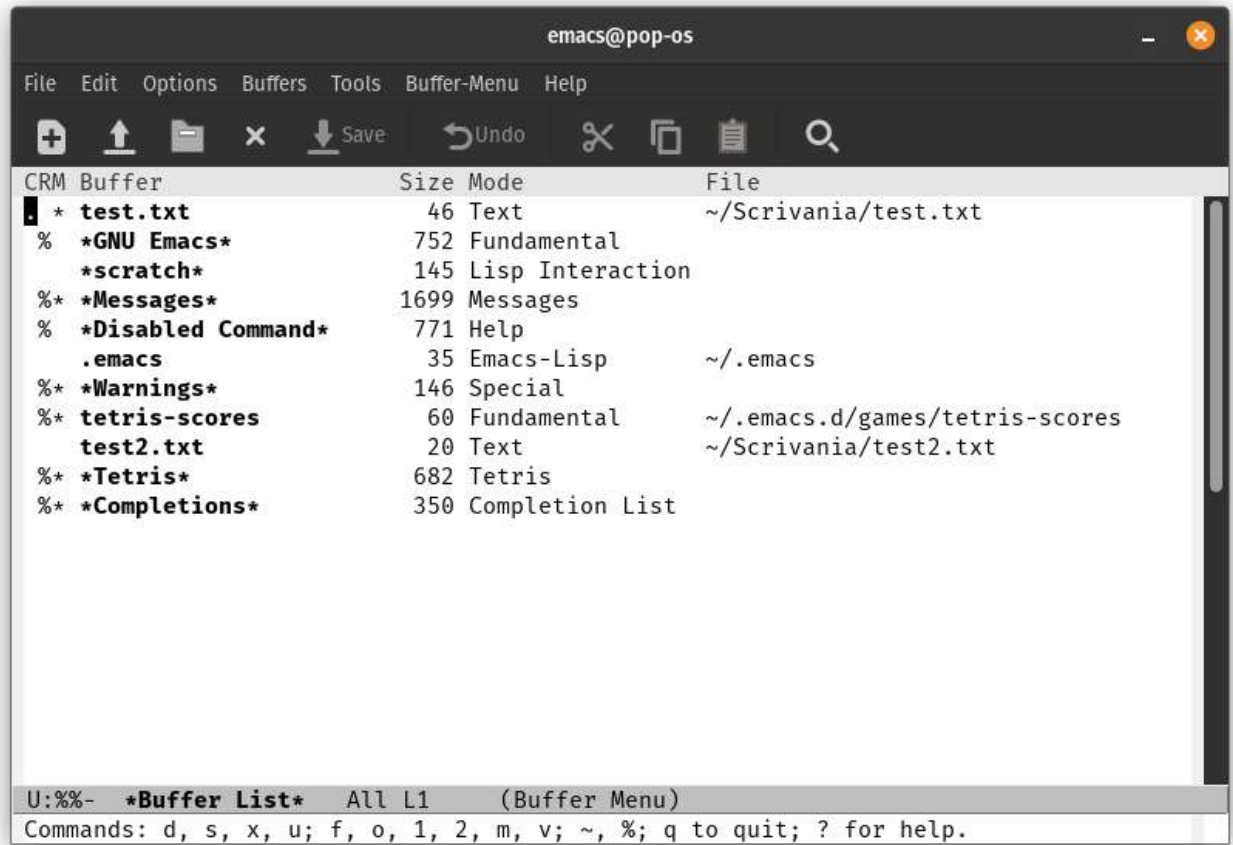
Questo contenuto non si trova ora nel file `test2.txt`, ma stiamo scrivendo nel **buffer**, possiamo confermare questo dai due asterischi vicino alla lettera U in basso a sinistra (U**), che indicano proprio che il contenuto si trova solo nel buffer attualmente. Per salvare il contenuto del buffer, nel file, digitiamo: `Ctrl - x - s`:



Ora al posto degli asterischi abbiamo dei trattini. Come detto in precedenza non tutti i **Buffer** sono associati a dei file, ad esempio, apriamo di nuovo il **Long Commands** e digitiamo, *tetris*:

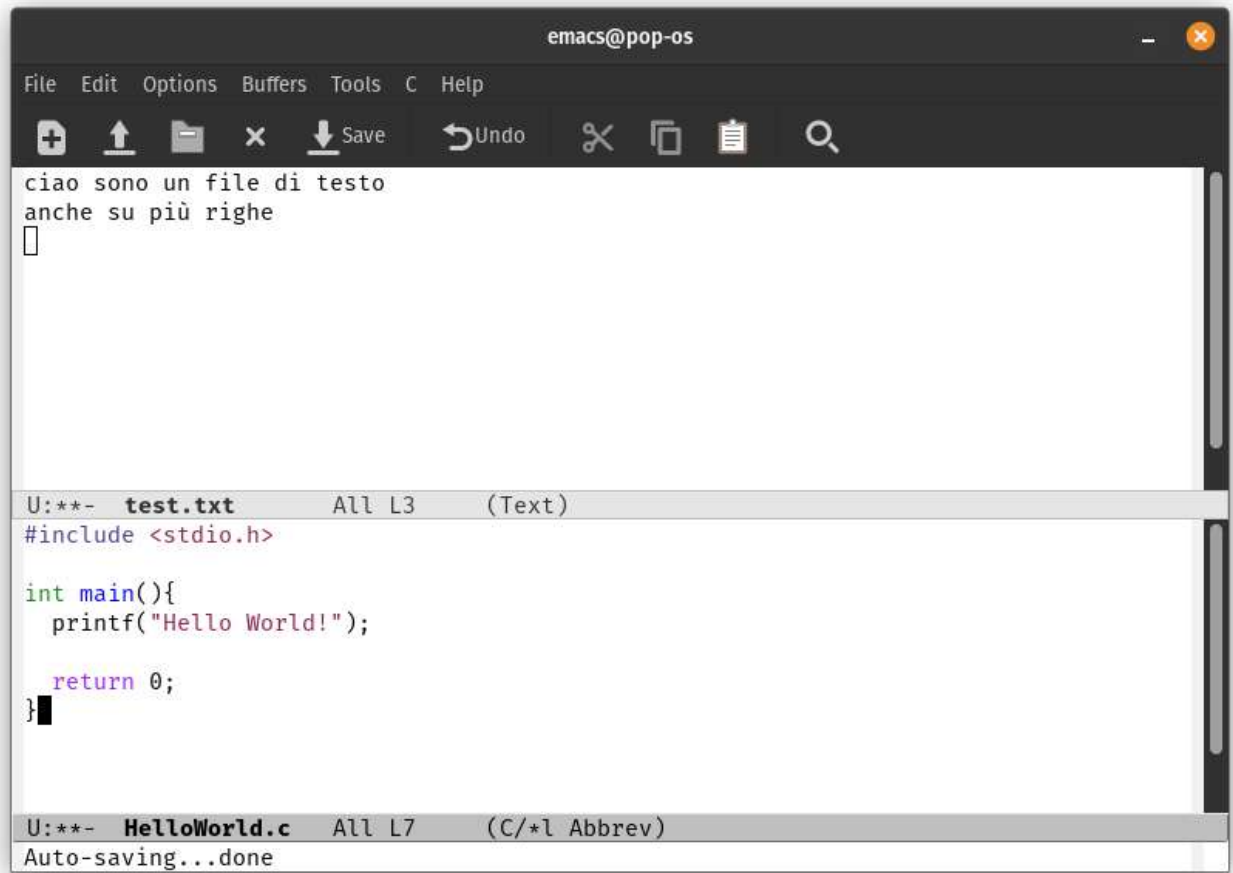


Con il **Long Command**: **buffer-menu** possiamo vedere tutti i buffer aperti nella nostra sessione di Emacs, e andando col cursore sulla riga di un buffer e premendo ENTER possiamo entrarvi:



Il buffer **messages** è un buffer particolare che **Emacs** usa per darci una serie di informazioni, come i file caricati, i messaggi etc.

Ogni **buffer** inoltre ha una **modes**, una modalità, ed ha anche una **modline** che ne indica il comportamento, per esempio, un buffer di testo sarà diverso da un buffer con un codice **.c**:



Le modalità in cui si può trovare ciascun buffer possono essere:

1. Major
2. Minor

Le **Major Mode** sono le modalità principali che specificano una serie di comportamenti che Emacs ha quando editor un buffer in quella particolare mode. Ad esempio, nell'immagine qui sopra riportata, quando apro un file .c, Emacs riconosce l'estensione e la apre in **C Mode**, che è una Major Mode, ed indica una modalità di programmazione. In Emacs esistono 3 Major Mode principali:

- Fundamental Mode
- Programming Mode
- Special Mode

La **Programming Mode** contiene tutte le modalità di programmazione per i vari linguaggi, e laddove non ve ne sia una, è possibile

aggiungerla mediante codice Elisp.

Un esempio di **Special Mode** è il gioco tetris visto in precedenza.

Nel caso di un editing di un file di testo, esso si troverà nella *text mode*, che deriva dalla **Fundamental Mode**.

Ogni **buffer** può stare in un'unica **major mode** alla volta ma può avere più **minor mode** attivate, queste ultime indicano delle piccole modifiche in quella particolare modalità.

Passiamo ora all'ultimo ramo, i **Files**, come abbiamo accennato prima, quando Emacs apro/crea un nuovo file, non lavora con il file direttamente, ma bensì con un buffer, il contenuto del file viene infatti trasferito nel buffer. Una volta salvato il buffer quest'ultimo sovrascrive il file.

Le **macros** permettono di registrare più keybinds mediante una macro appunto, e in fare di editing possiamo usare quest'ultima per essere più produttivi.

In fine, per accedere al tutorial in italiano basta eseguire i seguenti steps all'interno di Emacs: *Help > Emacs Tutorial (choose language) > Italian (questo nel minibuffer)*.

Emacs Frameworks

I frameworks di Emacs sono file di configurazione altamente modificabili e complessi che velocizzano l'utilizzo di Emacs per l'utente iniziale. In VScode siamo abituati ad avere già un file di configurare con le funzioni essenziali, ebbene, in Emacs dobbiamo configurarlo con il nostro codice **Elisp**. Se vogliamo un pò sorvolare questo step possiamo usare dei frameworks per renderci il lavoro più semplice e non dover scrivere codice Elisp. Riescono in questo obiettivo in quanto creano un livello di astrazione tra l'utente e il sottostante file di configurazione in modo tale che anche l'utente che non conosce elisp può ottenere con poco sforzo delle funzionalità avanzate.

I frameworks sono vari, possiamo consigliare:

- DOOM Emacs
- Spacemacs

È consigliato usarli a chi è interessato a configurare Emacs a modo proprio. Anche se il livello di astrazione che creano facilita la configurazione iniziale, in realtà rende più difficile la comprensione di quello che si sta facendo a livello di codice elisp, e dunque allontanano l'utente dal software che sta utilizzando.

Keybinds utili in Emacs

Avvio di Emacs

Da riga di comando digitare `emacs` o `emacs nome file`

Uscire da Emacs

Digitare: **C-x C-c**

Significa: tieni premuto control poi premi x, smetti di premere control e poi tieni premuto control poi premi c

Spostare il cursore nell'ambiente EMACS

Keybinds	Descrizione
M-b	Sposta il cursore all'inizio della parola alla sua sinistra.
M-f	Sposta il cursore alla fine della parola alla sua destra.
M-a	Sposta il cursore all'inizio della frase corrente.
M-e	Sposta il cursore alla fine della frase corrente.
C-n	Sposta il cursore sulla riga seguente.
C-p	Sposta il cursore sulla riga precedente.
C-a	Sposta il cursore all'inizio della riga.
C-e	Sposta il cursore alla fine della riga.
C-v	Sposta quanto visualizzato di un'intera schermata verso il basso.
M-v	Sposta quanto visualizzato di un'intera schermata verso l'alto.
M->	Sposta il cursore alla fine del file.
M-<	Sposta il cursore all'inizio del file.

Nota: che **M-b** significa tieni premuto alt poi premi b.

Se Emacs viene avviato in modalità C, ad esempio aprendo un file con estensione .c, possiederà funzionalità mancanti alla modalità predefinita, la Lisp Interaction.

Keybinds	Descrizione
M-C-a	Sposta il cursore all'inizio della funzione corrente
M-C-e	Sposta il cursore alla fine della funzione corrente
M-a	Sposta il cursore all'inizio dell'istruzione C più interna
M-e	Sposta il cursore alla fine dell'istruzione C più interna

Cancellazione di testo

Keybinds	Descrizione
C-d	Cancella il testo in corrispondenza del cursore
C-k	Cancella il testo dalla posizione del cursore fino alla fine della riga

Per cancellare un'intera area di testo:

- Spostare il cursore sul primo carattere dell'area.
- Digitare **C-SPACE** per marcarne l'inizio.
- Spostare il cursore alla fine dell'area.
- Digitare **C-w** per cancellare l'area selezionata

Se si vuole ripristinare il testo cancellato: C-x u

Ricerca di testo in Emacs

Digitare **C-s** e successivamente inserire la stringa da ricercare.

Salvare file in Emacs

Digitare **C-x C-s** per salvare semplicemente. Digitare **C-x C-w** per salvare con nome. Digitare **C-x C-f** per aprire un file all'interno del buffer corrente.

Dividere l'ambiente in più finestre

Digitare **C-x 2** per dividere l'ambiente in due finestre e avere la possibilità di lavorare su due file diversi. Digitare **C-x 1** Cancella

tutte le finestre ad eccezione di quella corrente. Digitare **C-x o** Si sposta sull'altra finestra.

Funzioni di supporto alla programmazione

Emacs supporta le modalità di un'ampia gamma di linguaggi di programmazione; I linguaggi supportati includono diverse varietà di Lisp, C, C++, Pascal, Perl, Tcl.. Per passare a una determinata modalità

di linguaggio, digitare:

M-x [language]-mode

Per passare al C digitare:

M-x c-mode

Compilazione con Emacs

Per compilare utilizzare il comando:

M-x compile

Poi scrivere il comando di compilazione.

Muoversi tra i buffer

C-x C-f è sempre seguito da un nome file. Il comando per spostarsi tra i buffer, **C-x b**, è seguito da un nome del buffer. Il nome del buffer e il nome del file, se presente, sono gli stessi a meno che non vengano modificati (consultare la sezione "Rinominare i buffer", più avanti in questo capitolo). Per spostarsi tra i buffer, digitare **C-x b**. Emacs ti mostra un nome di buffer predefinito. Premi Invio se è il buffer che desideri, oppure digita i primi caratteri del nome del buffer corretto e premi Tab. **Emacs** completa il resto del nome. Ora premi Invio per passare al buffer.

Se digiti C-x b seguito da:	Spiegazione
Un nuovo nome del buffer	Crea un nuovo buffer che non è connesso a un file e si sposta lì.
Il nome di un buffer esistente	Ti sposta nel buffer (non importa se il buffer è collegato a un file o meno).

Copia e incolla in Emacs

Per copiare o eliminare del testo in Emacs, devi prima selezionare il testo. Questo viene fatto usando il comando di selezione `Ctrl + Spazio` e poi spostati con le frecce o i relativi keybinds:

- `Ctrl - p / C-p` : vai in alto
- `Ctrl - n / C-n` : vai in basso
- `Ctrl - b / C-b` : vai a sinistra
- `Ctrl - f / C-f` : vai a destra

Se desideri copiare solo la regione selezionata, puoi farlo premendo `Alt + w`. Per tagliare o eliminare il testo, puoi utilizzare i tasti `Ctrl + k` per terminare una riga particolare, o il comando `Ctrl + w` per terminare l'intera regione selezionata. Per incollare, o strappare, il testo, premere i tasti `Ctrl + y`. Questo incolla l'ultimo oggetto ucciso dal kill ring. Emacs ti consente anche di scorrere l'elenco dei kill-ring usando il comando `Alt + y`.

Per fare Taglia e Incolla:

1. Selezionare il testo con il cursore del mouse
2. Premere `C-w`
3. Spostarsi dove si desidera
4. Premere `C-y`

Per fare il Copia e Incolla:

1. Selezionare il testo con il cursore del mouse
2. Premere `M-w`
3. Spostarsi dove si desidera
4. Premere `C-y`

Keybinds in Emacs

In questa sezione tratteremo i Keybinds in Emacs, iniziamo con due concetti molto utili, che sono quello di **modificatore** e di **notazione**.

I **modificatori** sono dei tasti sulla nostra tastiera che modificano il comportamento degli altri tasti. Un tipo modificatore è il tasto

SHIFT, che ci permette di scrivere le lettere in maiuscolo.

In Emacs i modificatori più comunemente utilizzati sono i seguenti:

- **Control:** abbreviato con CTRL o C
- **Meta:** abbreviato con M
- **Esc:** può essere utilizzato per simulare il meta premendolo e poi rilasciandolo
- **Shift**

Tipicamente però bastano e avanzano i primi due. Il nome *meta* potrebbe confondere, dato che nelle tastiere moderne non appare nessun tasto con questo nome. Questo nome, come altri nomi in Emacs, è stato ereditato dalle tastiere del passato, che contenevano anche un tasto meta:



In ogni caso, il tasto meta non è altro che il moderno tasto **alt** (quello a sinistra).

Parliamo ora della definizione di **notazione**, i **modificatori** sono combinati assieme ad altri tasti della tastiera per formare delle sequenze particolari di tasti. Per descrivere queste sequenze viene utilizzata la seguente notazione:

```
C-<char>, per dire:
```

```
-----
```

```
  tieni premuto il modificatore CONTROL,  
  e premi il tasto <char>.
```

```
M-<char>, per dire:
```

```
-----
```

```
  tieni premuto il modificatore META,  
  e premi il tasto <char>.
```

Ad esempio se premo *C-x* (Ctrl-x) nel minibuffer Emacs è in attesa che io premi altri keybinds, questo perchè questo particolare keybind che è *C-x* può essere terminato in diversi modi, con appunti diversi keybind.

La cosa è molto simile anche con *M-x* ad esempio (Alt-x), che permette nel **minibuffer** di usare i **long commands**.

Ecco altri esempi:

```
C-x ---> Tieni premuto CTRL e premi x  
C-c ---> Tieni premuto CTRL e premi c  
M-h ---> Tieni premuto META e premi h
```

Certe volte possiamo combinare queste sequenze atomiche tra loro per ottenere sequenze più complesse:

```
<modifier>-<char> <remaining>, per dire:
```

```
-----
```

```
  tieni premuto <modifier>, premi il tasto <char>,  
  rilascia <modifier> e premi i restanti tasti in <remaining>.
```

Con **sequenze atomiche** intendiamo: *C-x*, *C-c* etc... Con **remaining** intendiamo altre coppie. Ecco un esempio di quanto appena detto:

```
C-x M-n ---> Tieni premuto CTRL, premi x, rilascia CTRL  
              tieni premuto Meta, premi n
```

```
C-h M-x ---> Tieni premuto CTRL, premi h, rilascia CTRL  
              tieni premuto Meta, premi x
```

Nel caso in cui la coppia `<modifier>-<char>` successiva condivide lo stesso modificatore della coppia precedente, non dobbiamo rilasciare il modificatore, ma possiamo direttamente premere il prossimo carattere.

Nota: Laddove ci sono dei keybinds che ripetono lo stesso modificatore, ad esempio il keybind per modificare il buffer, che è `Ctrl x Ctrl s` (abbreviato `C-x C-s`) la notazione di esso sarà: `C-x-s`. Questo è un modo compatto per descrivere gli stessi keybinds che hanno lo stesso modificatore. Ne esistono svariati così di keybind.

Ad alcune sequenze di tasti in Emacs possiamo associare un particolare comando, identificato tramite il nome simbolico della rispettiva funzione elisp che implementa tale comando. Questa associazione viene effettuata tramite l'operazione di **binding**.

Nota: Ogni keybind è associato ad un comando, e ogni comando è associato ad un keybind, non ha tutti i comandi sono associati dei keybind, ma sicuramente a tutti i keybind è associato un comando.

Un keybind è quindi un tasto (o una particolare sequenza di tasti) a cui è stato associato un comando. Quando premiamo correttamente il keybind, Emacs eseguirà il comando associato al keybind tramite il **binding** (quest'ultimo significa proprio *legare*, in inglese):



Il **binding** può essere fatto con **Elisp**. Andiamo adesso a vedere i keybinds associati ai comandi più importanti di Emacs.

Nota: tutti i comandi associati ai keybind che stiamo andando a vedere possono usati tramite il **long commands**.

Keybinds Generali

Keybind	Comando	Descrizione
C-x C-c	Save-buffers-kill-terminal	Salva tutti i buffer e distruggi/chiudi la sessione di Emacs corrente
C-g	Keyboard-quit	Permette di uscire/annullare i comandi che si stavano impartendo nel minibuffer. Se ad esempio stiamo inserendo un file mediante questo possiamo annullare l'operazione

In caso abbiamo buffer non salvati (dunque con degli asterischi), nel minibuffer ci verrà chiesto se vogliamo uscire salvando o meno.

Keybinds di movimento

Keybind	Comando	Descrizione
C-p	previous-line	Vai in su
C-n	next-line	Vai in giù
C-b	backward-char	Vai a sinistra
C-f	forward-char	Vai a destra

Keybinds di text editing

Keybind	Comando	Descrizione
C-x C-f	find-file	Permette di cercare o creare un file
C-x C-v	find-alternative-file	Permette di cercare un file laddove ne abbiamo aperto già uno
C-x C-s	save-buffer	Permette di salvare il contenuto del buffer corrente nel relativo file
C-x C-w	write-file	Ci permette di copiare un file in un altro file così da poterlo modificare. Può essere utile nei file read-only (In Emacs indicato con il simbolo %)

Keybind	Comando	Descrizione
C-x i	insert-file	Specificato il nome di un file, ci permette di aggiungere nel buffer corrente il contenuto del file specificato

Keybinds della Documentazione Interna

Questi keybinds sono utili per chi è alle prime armi con questo Text Editor, perchè permettono di imparare Emacs al suo interno, ed era proprio il punto di *Stallman* che voleva creare un software self-documenting, ossia che contiene la sua stessa documentazione e non deve usare altre fonti.

Keybind	Comando	Descrizione
C-h k	describe-key	Permette di inserire un keybind e mostra la sua relativa documentazione
C-h f	describe-function	Permette di conoscere cosa fa una funzione che non è associata a nessun keybind ma non solo
C-h v	describe-variable	Permette di conoscere cosa fa una variabile d'ambiente di Emacs. Utile quando dobbiamo scrivere il nostro file di configurazione

Global Set Key

Con il codice Elisp è possibile effettuare il global set key, cos'è questa operazione? La risposta è molto semplice, si va ad effettuare un mapping, ossia associare certi keybind a certe funzioni/comandi, questo con due modi:

1. Associare nuovi keybind a funzionalità di Emacs laddove i keybind associati a questi comandi non ci aggradano
2. Associare dei keybind a funzioni che scriviamo noi con Elisp

Questo è possibile mediante del codice Elisp che vediamo di seguito (**Nota:** tale codice va inserito nel file `.emacs`, esso sarà trattato in un paragrafo più avanti nel dettaglio):

```
;; Versione Generica del comando
(global-set-key (kbd keybind) 'funzione)

;; Esempi Pratici
(global-set-key (kbd "C-x t") 'mu4e)
(global-set-key (kbd "C-c q") 'query-replace)
(global-set-key (kbd "C-c v") 'my/vterm-window-split)
```

global-set-key è una **funzione**, che accetta due parametri: il keybind, è la funzione che deve associare al rispettivo keybind. **Kbd** è un **formato per specificare i keybind**.

Muoversi in Emacs

Per quanto riguarda le definizioni legate al muoversi in Emacs, ci interessano due definizioni, quelle di **punto** e di **cursore**.

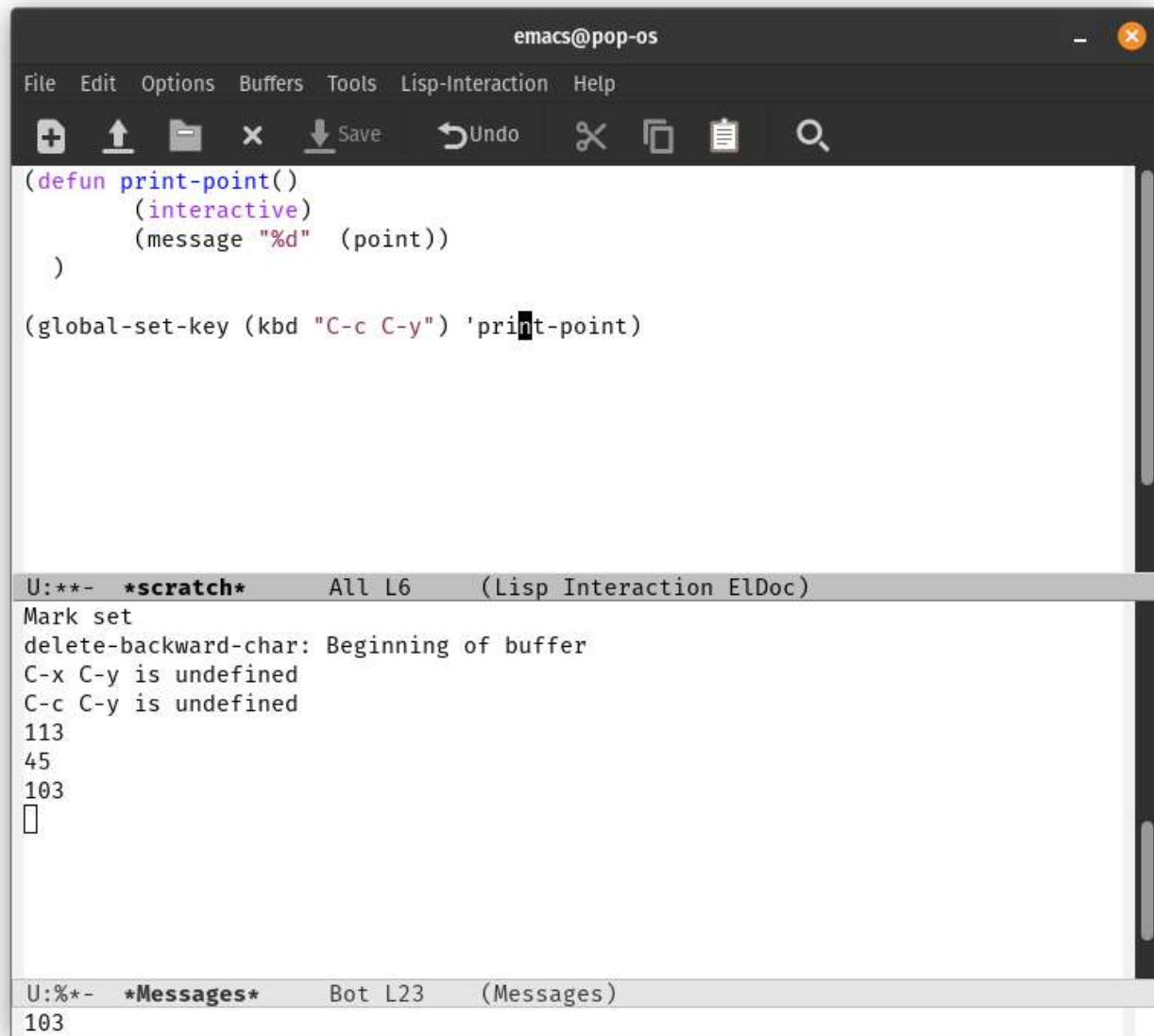
Il **punto** (in inglese *point*) è il luogo in cui Emacs inserisce il prossimo carattere che scriviamo, ogni buffer ha associato il proprio punto. In **elisp** il punto è rappresentato come un intero, la **funzione point** ritorna l'intero corrispondente al punto nel buffer corrente.

Per testare il valore del point possiamo eseguire il seguente codice elisp, andandolo a copiare in un buffer vuoto e chiamando il comando *eval-buffer*:

```
(defun print-point()
  (interactive)
  (message "%d" (point))
  )

(global-set-key (kbd "C-c C-y") 'print-point)
```

Senza entrare nel dettaglio, tale codice definisce una **funzione print-point()** che stampa il valore del punto nel buffer in cui ci troviamo. Tale funzione viene bindata al keybind: *C-c C-y* (questo grazie ad *interactive* che permette di usare il minibuffer in modalità long commands):



```
emacs@pop-os
File Edit Options Buffers Tools Lisp-Interaction Help
+ ↑ 📁 × ↓ Save ↶ Undo ✂ 📄 🗑️ 🔍

(defun print-point()
  (interactive)
  (message "%d" (point))
)

(global-set-key (kbd "C-c C-y") 'print-point)

U:*** *scratch* All L6 (Lisp Interaction ElDoc)
Mark set
delete-backward-char: Beginning of buffer
C-x C-y is undefined
C-c C-y is undefined
113
45
103
█

U:%*- *Messages* Bot L23 (Messages)
103
```

Ho copiato il codice visto in precedenza nel **buffer scratch** (ho digitato come comando: *C-t b* per cambiare buffer), ho digitato come long command: *eval-buffer*, in seguito ho splittato orizzontalmente la window e sotto ho aperto il buffer Messages. Da qui in avanti se eseguo questa funzione con il keybind *C-c C-y* nel **buffer Messenger** verrà mostrato un valore intero che indica la posizione del cursore. Il **cursore** (in inglese *cursor*) invece rappresenta il "punto grafico", ovvero ciò che possiamo vedere visivamente mentre scriviamo. In pratica il cursore in Emacs è quel piccolo rettangolo che lampeggia. Possiamo cambiare il tipo di cursore utilizzato tramite la variabile *cursor-type*. I valori possibili sono:

Valore del cursore	Descrizione
t	Use the cursor specified for the frame
nil	Don't display a cursor
box	Display a filled box cursor
hollow	Display a hollow box cursor
bar	Display a vertical bar cursor with default width
(bar . WIDTH)	Display a vertical bar cursor with width WIDTH
hbar	Display a horizontal bar cursor with default height
(hbar . WIDTH)	Display a horizontal bar cursor with height HEIGHT
ANYTHING ELSE	Display a hollow box cursor

Per impostare un nuovo cursore, ad esempio *bar*, dobbiamo digitare `M-x`, inseriamo `set-variable`, in seguito scriviamo `cursor-type` e inseriamo il nostro valore, ossia `bar`.

La differenza tra questi due cursori non è solo visivo ma anche logico, questo perchè, quando utilizziamo un **cursore di tipo box**, ovvero quando il cursore appare sopra un carattere, il punto è immediatamente prima del carattere. Questo significa che la successiva lettera sarà scritta alla sinistra di quella coperta dal cursore. Se invece utilizziamo un **cursore di tipo bar**, allora il punto è nella stessa posizione del cursore. Vi è una modalità di scrittura peculiare che è la **overwrite mode**, che si attiva digitando come long command: `overwrite-mode`, per disabilitarla scriviamo di nuovo `overwrite-mode`.

Essa permette di sovrascrivere un carattere, con uno nuovo, se è coperto la cursore di tipo box. Col cursore di tipo bar e in `overwrite-mode` il punto è a destra del cursore.

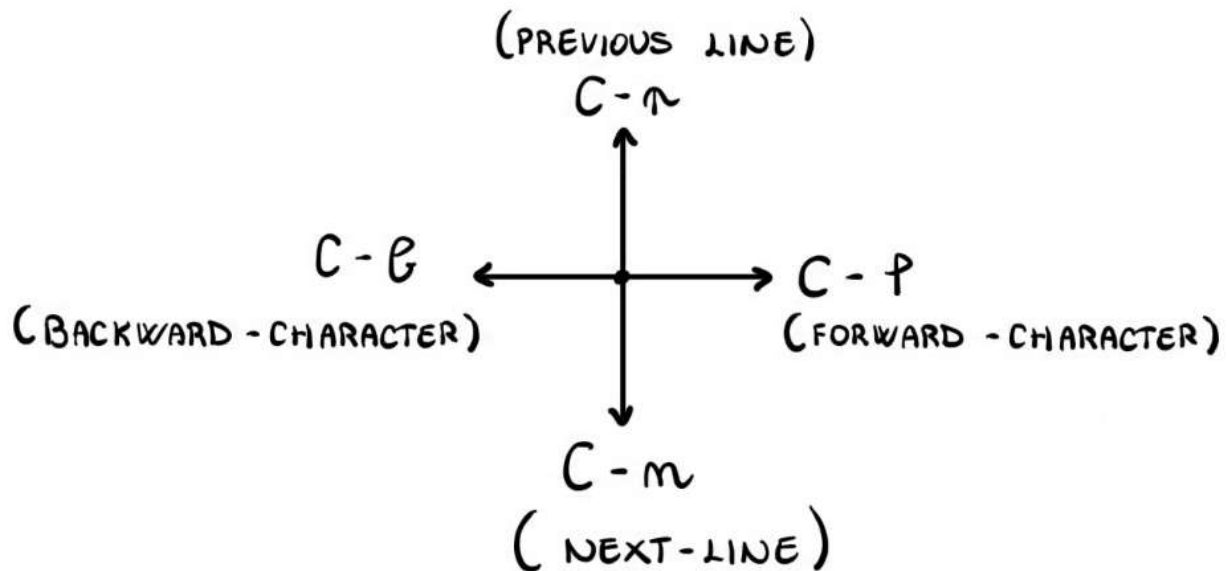
Una funzionalità molto utile di **Emacs** è il fatto che esso ricorda l'ultima posizione del cursore nel caso stiamo lavorando su più buffer.

I movimenti base in Emacs sono stati già trattati, ma facciamo una piccola menzione:

Keybind	Comando	Descrizione
---------	---------	-------------

Keybind	Comando	Descrizione
C-p	previous-line	Sposta il cursore in alto
C-n	next-line	Sposta il cursore in basso
C-b	backward-char	Sposta il cursore a sinistra
C-f	forward-char	Sposta il cursore a destra

Graficamente la situazione è:



Inizialmente i **keybinds** per muoversi in **Emacs** potrebbero confondere, specialmente a chi non è abituato a questa tipologia di software (Emacs/Vim/Unix). Detto questo, c'è una chiara e semplice motivazione dietro a questi keybinds: non allontanare la posizione delle mani dal centro della tastiera per massimizzare la produttività. Infatti, il tempo impiegato per muoversi dal centro della tastiera al mouse, per quanto piccolo possa essere, diventa significativo quando consideriamo il numero di volte in cui dobbiamo farlo.

Utilizzando i keybinds di **Emacs** siamo in grado di restare sempre al centro della tastiera, in modo da avere accesso alla maggior parte dei tasti senza dover spostare la mano.

Oltre ai movimenti base, che ci spostano di un carattere alla volta, ci sono altri keybinds che ci permettono di effettuare gli spostamenti più ampi. Ad esempio:

Keybind	Comando	Descrizione
M-f	forward-word	Avanti di una parola
M-b	backward-word	Indietro di una parola
C-a	beginning-of-visual-line	Vai a inizio riga
C-e	end-of-visual-line	Vai a fine riga
M-a	backward-sentence	Vai a inizio della frase
M-e	forward-sentence	Vai a fine della frase
M-{	backward-paragraph	Vai a inizio paragrafo
M-}	forward-paragraph	Vai a fine paragrafo

Nota: Emacs per frase intende un pezzo di testo che termina da un punto seguito da due spazi.

Oltre a muovere il cursore, è possibile anche muovere lo schermo, ovvero la sezione del buffer visibile. Questa tipologia di movimenti sono molto utili quando abbiamo a che fare con file molto grandi. Il movimento dello schermo può essere effettuato sia con il mouse che automaticamente muovendo il cursore.

Ecco i keybind:

Keybind	Comando	Descrizione
C-v	Scroll-down-command	Scorre in avanti di una pagina
M-v	Scroll-up-command	Scorre in dietro di una pagina
M-<	Beginning-of-buffer	Va all'inizio del buffer
M->	End-of-buffer	Va alla fine del buffer
C-l	recenter-top-bottom	Centra lo schermo in base alla posizione del cursore

Nota: Per movimento dello schermo, in Emacs, si intende il contenuto del buffer.

Infine, troviamo i seguenti comandi:

Keybind	Comando	Descrizione
-	goto-line	Vai alla linea specificata
-	goto-char	Vai al carattere specificato

Esse sono molto utili quando dobbiamo, ad esempio, capire un errore di compilazione. Essi sono comandi da digitare come long commands perchè non sono associati a nessun keybind.

Esercizi Base in Emacs

Gli esercizi che seguono sono degli esercizi molto banali ma efficaci per iniziare a capire le basi di Emacs.

Esercizio #1

Effettuare i seguenti task:

- Aprire Emacs
- Aprire un nuovo file
- Copiare il testo che segue
- Salvare il file
- Chiudere Emacs

Il testo è questo che segue:

```
Amalia invece non si è limitata a sopportare la pena, lei aveva anche
l'intelligenza per capirla fino in fondo; noi scorgevamo solo le
conseguenze, lei anche la causa, noi speravamo in qualche mezzuccio, lei
sapeva che tutto era ormai deciso, noi potevamo bisbigliare fra di noi, lei
non poteva che tacere: lei si trovava faccia a faccia con la verità, eppure
ha vissuto e supportato questa vita, oggi come allora.
Noi tutti, pur nella nostra miseria, stavamo meglio di lei.
```

Aprire Emacs: da terminale digitiamo *Emacs*

Aprire un nuovo file: premiamo la keybinds C-x C-f e diamogli un nome, ad esempio *testo.txt*

Copiare il testo: Copiare a mano il testo fornito, nel buffer appena aperto

Salvare il file: Per salvare il testo premiamo C-x C-s

Chiudere Emacs: Per chiudere il programma premiamo C-x C-c

Esercizio #2

Effettuare i seguenti task:

- Aprire Emacs
- Aprire il file `/etc/passwd`
- Salvare una copia di tale file in `~/my_passwd` (la tilde indica la home directory dell'utente)
- Modificare il file a piacimento
- Salvare il file
- Chiudere Emacs

Aprire Emacs: da terminale digitiamo *Emacs*

Aprire il file `/etc/passwd`: digito C-x C-f e ricerco `/etc/passwd`

Salvare una copia di tale file in `~/my_passwd`: essendo il file `passwd` un file read-only, per modificarlo dobbiamo creare una copia, per farlo usiamo C-x C-w

Modificare il file a piacimento: qui la scelta è personale

Salvare il file: Per salvare il testo premiamo C-x C-s

Chiudere Emacs: Per chiudere il programma premiamo C-x C-c

Esercizio #3

Effettuare i seguenti task:

- Aprire Emacs
- Cercare un file nella nostra home directory
- Annullare il comando precedente prima di aprire il buffer
- Chiudere Emacs

Aprire Emacs: da terminale digitiamo *Emacs*

Cercare un file nella nostra home directory: digito C-x C-f e ricerco il file di interesse

Annullare il comando precedente prima di aprire il buffer: per annullare la ricerca di questo file dal minibuffer (più nello specifico nella long commands) digito C-g

Chiudere Emacs: Per chiudere il programma premiamo C-x C-c

Esercizio #4

Effettuare i seguenti task:

- Aprire Emacs
- Leggere la documentazione della funzione `insert-file`
- Leggere la documentazione della keybind `C-h t`
- Leggere la documentazione della variabile `inhibit-startup-screen`

Aprire Emacs: da terminale digitiamo `Emacs`

Leggere la documentazione della funzione `insert-file`: digito `C-h f` per entrare nel sistema della documentazione di Emacs, in seguito inserirò la funzione da ricercare, in questo caso **`insert-file`**

Leggere la documentazione della keybind `C-h t`: digito `C-h k` per entrare nel sistema della documentazione di Emacs, in seguito inserirò il keybind da ricercare, in questo caso **`C-h t`**

Leggere la documentazione della variabile `inhibit-startup-screen`: digito `C-h v` per entrare nel sistema della documentazione di Emacs, in seguito inserirò la variabile da ricercare, in questo caso **`inhibit-startup-screen`**

Esercizio #5

Effettuare i seguenti task:

- Assegnare il comando `goto-line` ad un keybind a scelta e utilizzarlo

Assegnare il comando `goto-line` ad un keybind a scelta e utilizzarlo: Andiamo nel nostro file di configurazione `.emacs` e inseriamo questa riga:

```
(global-set-key (kbd "C-c C-p") 'goto-line)
```

E riavviamo `Emacs`.

Esercizio #6

Effettuare i seguenti task:

- Aprire un file e muoversi nel file utilizzando solamente i keybinds per il movimento base, senza arrows nè mouse

Esercizio #7

Effettuare i seguenti task:

- Cambiare lo stile del cursore al valore *hbar*

Cambiare lo stile del cursore al valore hbar: premiamo *M-x* e digitiamo **set-variables** e scriviamo **cursor-type** seguita dal nostro valore che è *hbar*.

Frames, Windows e Buffers in Emacs

Nel linguaggio di **Emacs**, per **frame** si intende la finestra GUI che contiene il programma in un ambiente grafico come X11.

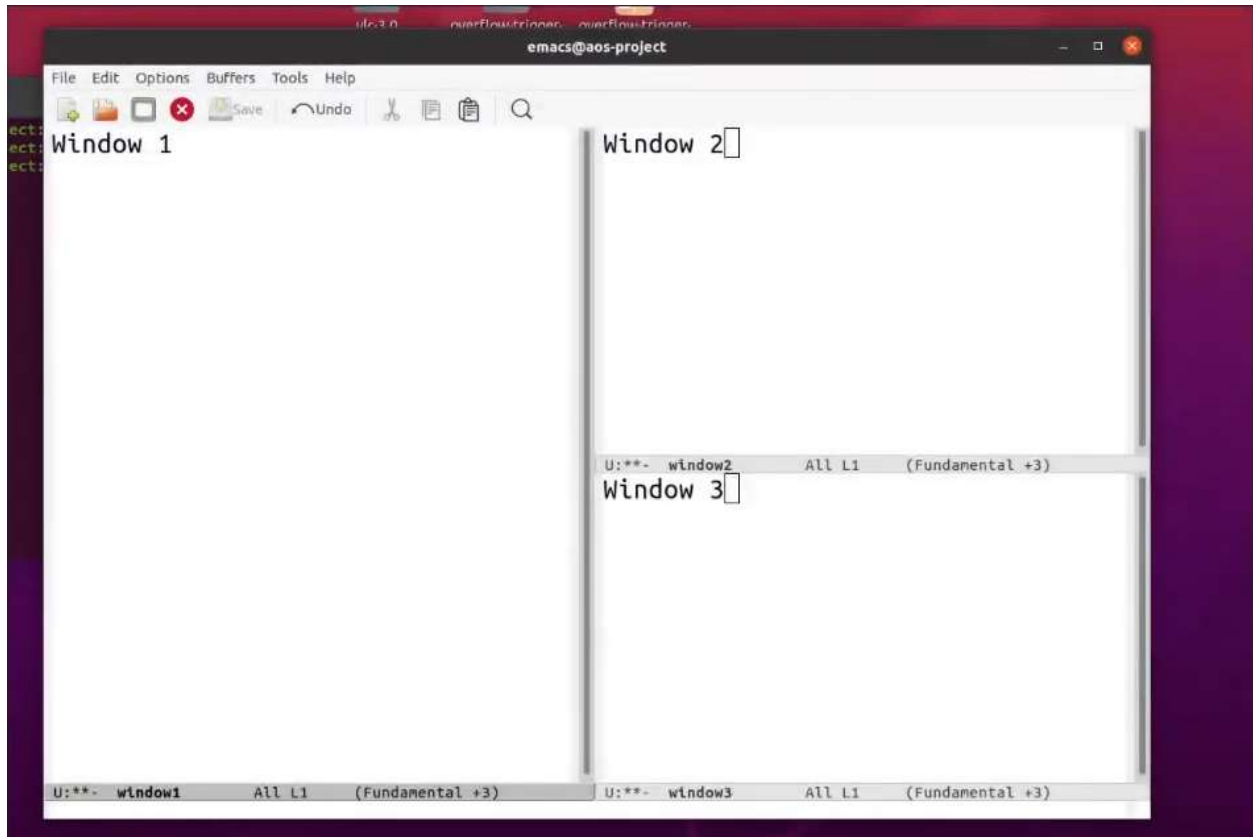
È possibile modificare le dimensioni del primo frame aperto con il seguente codice **Elisp**:

```
(setq initial-frame-alist '((top . 50)
                            (left . 150)
                            (width . 70)
                            (height . 120)))
```

Dei keybind curiosi ma non fondamentali legati ai frames sono:

Keybind	Comando	Descrizione
C-x 5 2	make-frame	Crea un nuovo frame
C-x 5 o	other-frame	Laddove ce ne sia già uno, si sposta al frame successivo

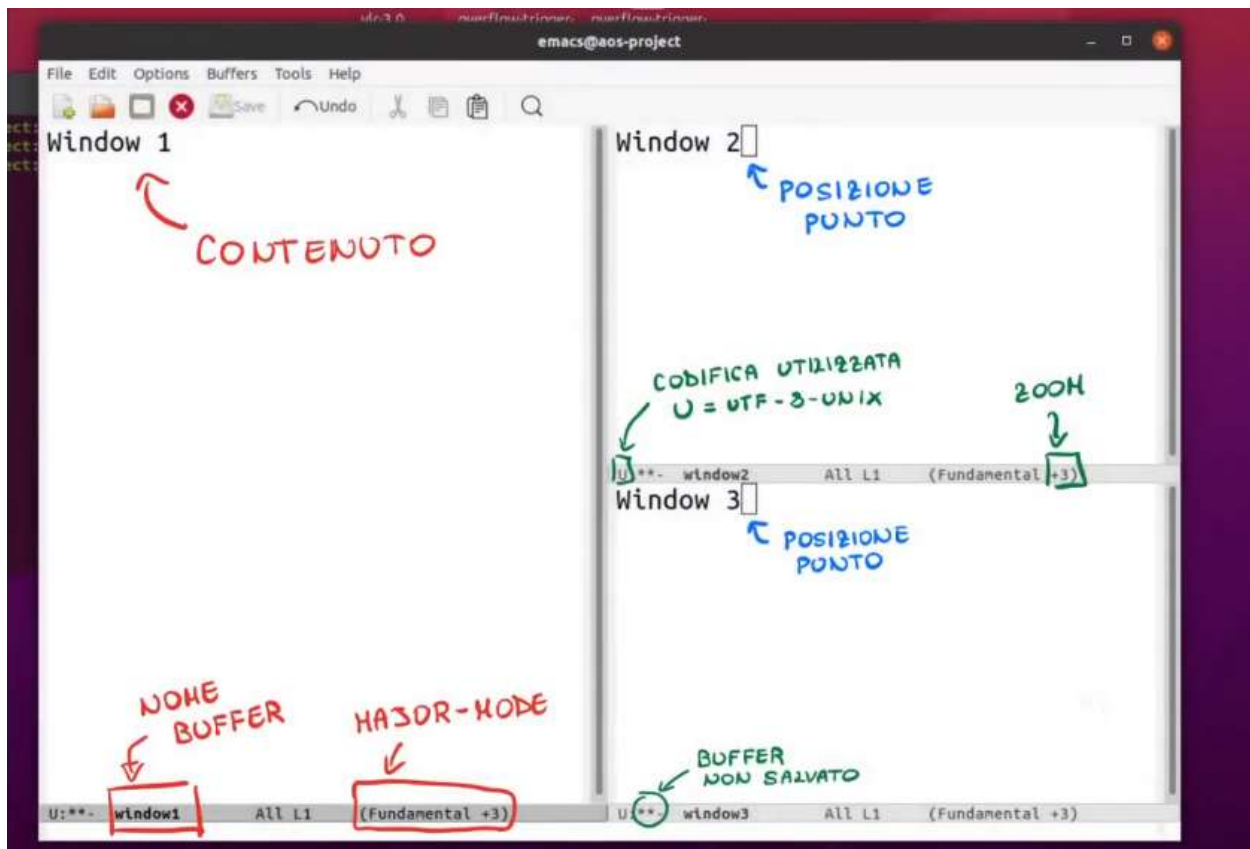
Per **Emacs** una **window** (finestra) è una porzione dello schermo visibile. Un frame di Emacs può essere diviso in più finestre (**windows**):



Ciascuna finestra mostra una porzione di un buffer. Oltre al contenuto del buffer (tipicamente testo), a ciascuna finestra sono associate le seguenti informazioni:

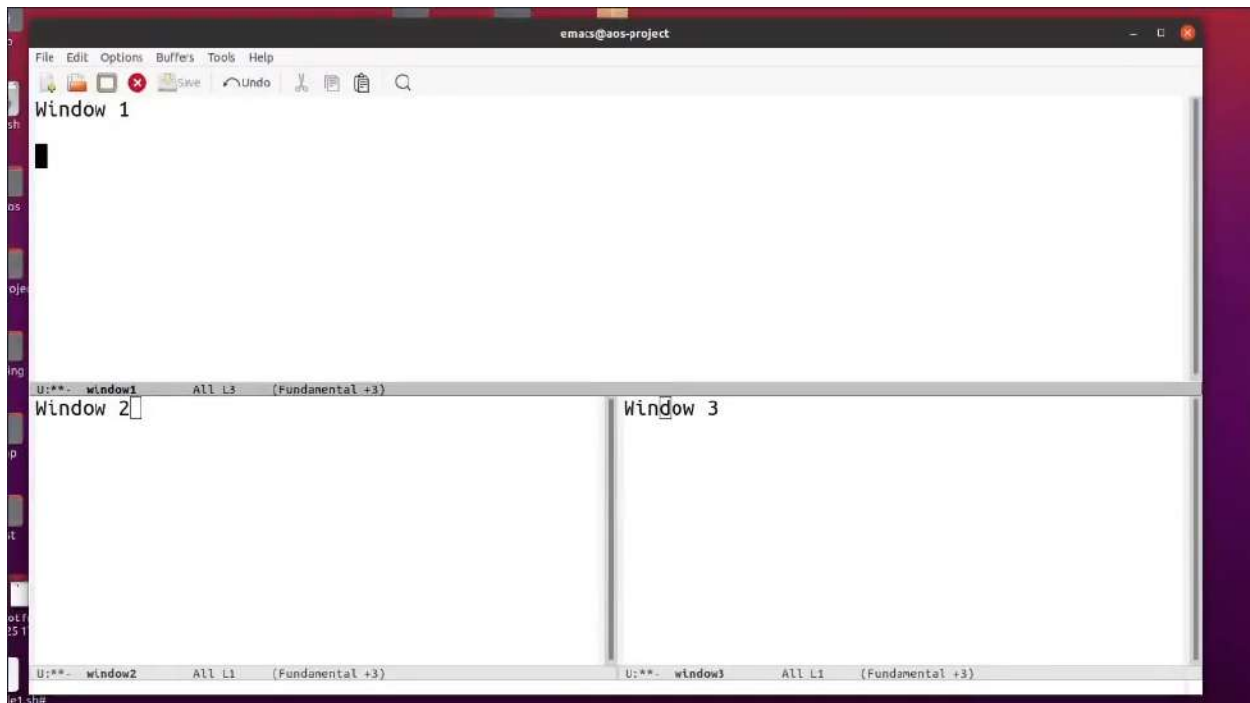
- Modeline
- Major Mode / Minor Modes attive
- Posizione del point nel buffer

Vediamo un'immagine per capire meglio questi tre punti:



Penso che l'immagine sia abbastanza autoesplicativa. In generale è possibile avere più windows contenenti lo stesso buffer. Ogni windows ha una **modline** (la barra orizzontale grigia) che fornisce delle info sulla finestra attiva.

In ogni momento è presente un solo cursore, che rappresenta la **finestra attiva** (active window) su cui si sta lavorando:



In questo caso la finestra attiva è quella col cursore nero, ossia Window 1. I keybind più importati per gestire le finestre sono:

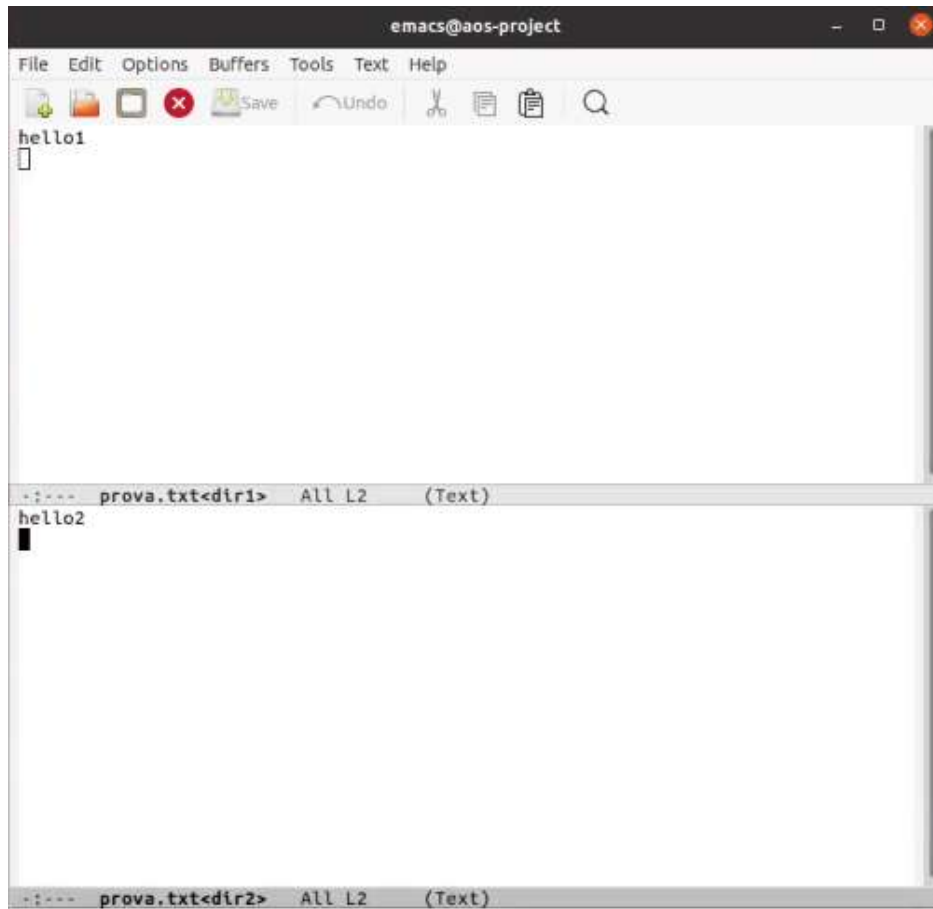
Keybind	Comando	Descrizione
C-x 2	split-window-vertically	Divide la finestra verticalmente
C-x 3	split-window-horizontally	Dividere la finestra orizzontalmente
C-x o	other-window	Permette di cambiare finestra (seguendo l'ordine da sinistra-destra alto-basso)
C-x 0	delete-window	Cancella la finestra attiva (ossia quella con il cursore)
C-x 1	delete-other-window	Cancella tutte le finestre facendone rimanere solo una
-	delete-windows-on	Cancella tutte le finestre su uno specifico buffer
C-x ^	enlarge-window	Allarga in altezza la finestra
-	shrink-window	Diminuisce in altezza la finestra
C-x +	balance-windows	Bilancia automaticamente la grandezza di tutte le finestre

I **buffer** sono i contenitori di contenuto. In altre parole, sono i luoghi in cui lavoriamo sempre mentre utilizziamo **Emacs**. Le finestre (e i frame) sono i modi per mostrare il contenuto di uno o più *buffers*. Alcuni buffer sono "*speciali*", in quanto vengono creati e gestiti internamente da **Emacs**. Tra questi troviamo il buffer *Messages* e il buffer *scratch*. Il **buffer scratch** verrà approfondito più avanti, limitiamoci a dire che permette di eseguire codice Lisp. Mentre il **buffer Messages** mostra dei messaggi di servizio di cosa sta facendo internamente Emacs.

Riconosciamo che un buffer è speciale, perchè nella modline viene mostrato il nome del buffer tra asterischi, ad esempio: ***Messages***, ***scratch***.

Come sappiamo il **keybind** *C-x C-f* ci permette di aprire un file o crearne uno laddove forniamo un nome che Emacs non trova. Bene, quando apriamo un file, non stiamo lavorando direttamente con esso, ma bensì con il suo buffer. Finchè il buffer non verrà salvato, il contenuto originale del file non verrà cambiato/sovrascritto.

Il nome del nuovo buffer corrisponderà al nome del file, a meno di ononimi, nel senso che, se abbiamo due file con lo stesso nome, in directory diverse in **Emacs** oltre al nome del file (che in questa circostanza sarà uguale) tra le parentesi angolari vedremo anche le rispettive directory dei file:



In questo esempio abbiamo due file (*prova.txt*) entrambi con lo stesso nome, ma in directory diverse (*/dir1* e */dir2*), in Emacs vedremo queste due directory.

Tramite il **keybind** `C-x b`, che chiama il comando **switch-to-buffer**, siamo in grado di:

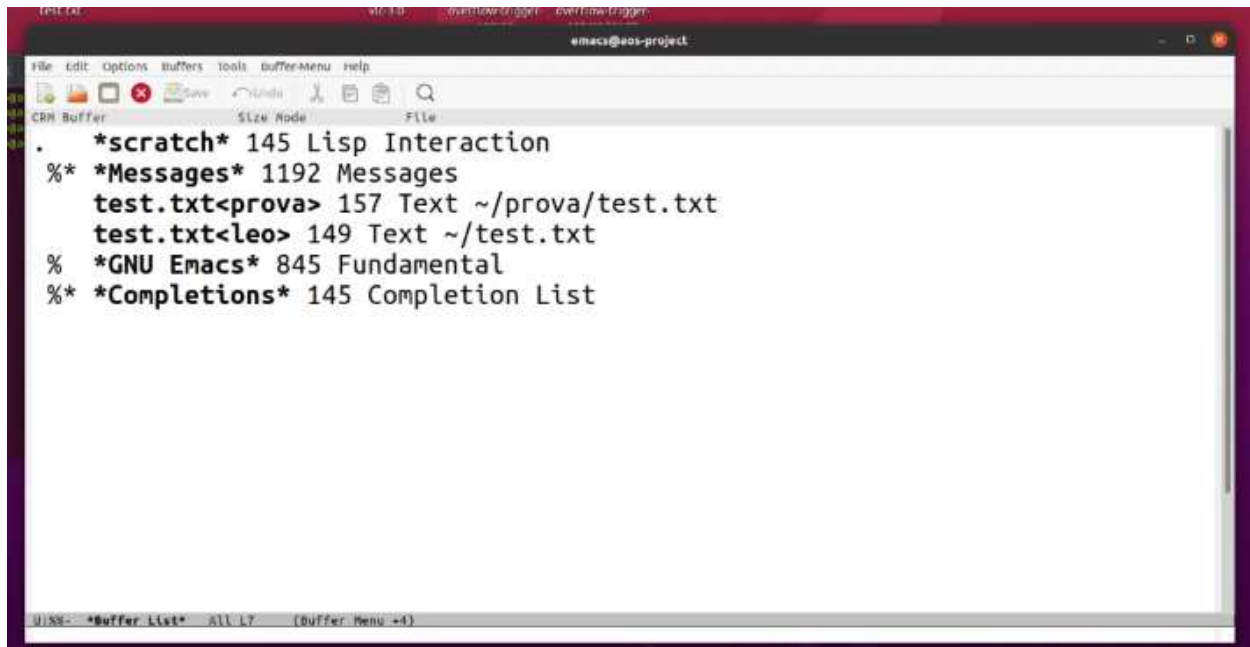
- Cambiare il buffer attuale
- Creare un nuovo buffer senza associarlo ad alcun file

In ogni dato momento vediamo solo una piccola parte dei buffer attivi, per vedere tutti i buffer attivi ci sono tre modi diversi:

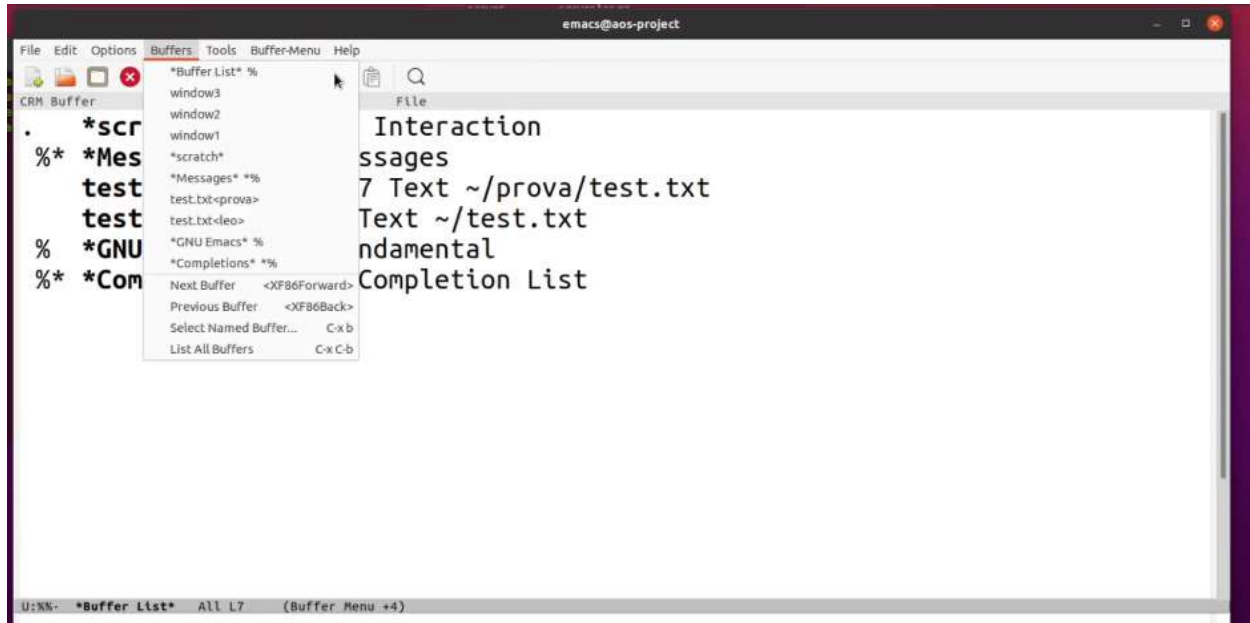
1. Tramite la **buffer list**.
2. Tramite il **buffer menu**.
3. Tramite il **buffer pop-up menu**.

Per accedere alla **buffer list** (o meglio, alla lista dei buffer associati a dei file) possiamo utilizzare il **keybind** `C-x C-b` o

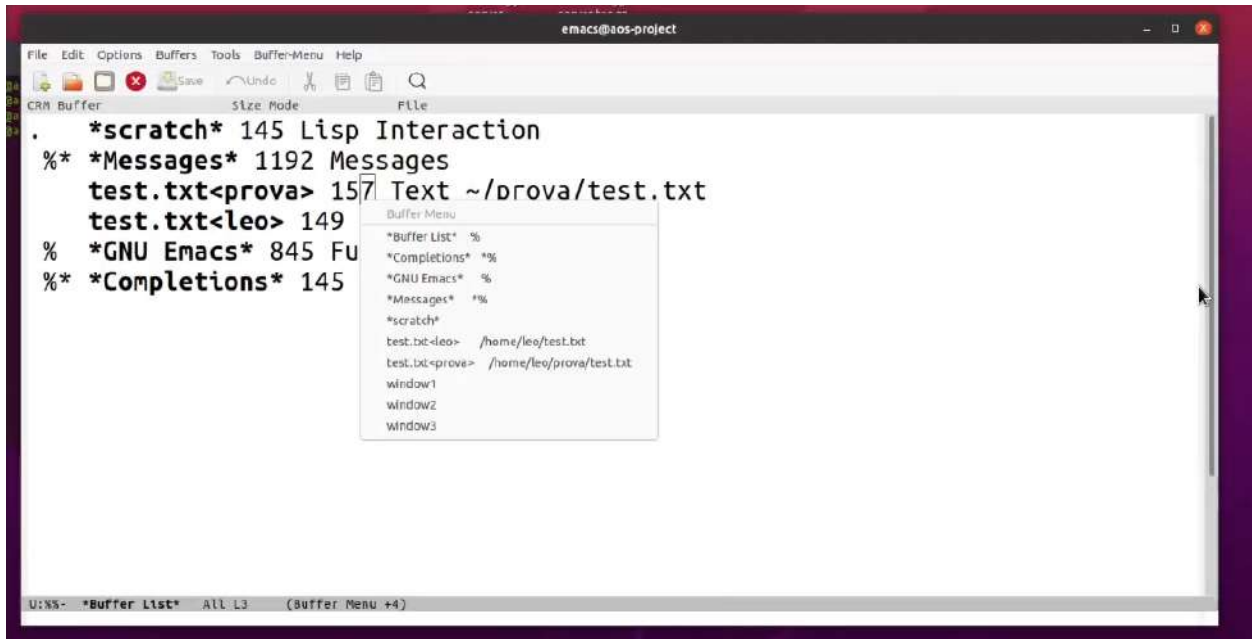
equivalentemente il comando `list-buffers`:



Il **buffers menu** invece fa parte della GUI di Emacs:



Infine, il **buffer pop-up menu** fa sempre parte della GUI di Emacs, e può essere attivato premendo `CTRL + left mouse click`:



Una volta aperta la buffer list, possiamo eseguire vari comandi, tra cui:

Keybind	Descrizione
1	Mostra/apri in full screen il buffer puntato
f	Mostra buffer puntato su finestra attuale
o	Mostra buffer puntato su altra finestra
d, k	Mark per deletion
s	Salva il buffer
u	Unmark
x	Esegui comandi sui buffer marcati
q	quit

In generale, per gestire i buffers al di fuori della **buffer list** i seguenti keybinds sono utili:

Keybind	Comando	Descrizione
C-x b	switch-to-buffer	Cambia il buffer
C-x C-b	list-buffers	Apri la lista dei buffer attivi su un file
C-x k	kill-buffer	Elimina un particolare buffer
-	rename-buffer	Permette di rinominare un buffer

Keybind	Comando	Descrizione
C-x C-q	read-only-mode	Rende un buffer read-only
C-x s	save-some-buffers	Itera l'azione di salvataggio su tutti i buffer
-	kill-some-buffers	Itera l'azione di chiusura su tutti i buffer

Il buffer Scratch

Quando Emacs si avvia, contiene un buffer chiamato *scratch*, che viene fornito per valutare le espressioni Emacs Lisp in modo interattivo. La sua modalità principale è la modalità di interazione Lisp. Si può anche abilitare la modalità Lisp Interaction digitando M-x **lisp-interaction-mode**. Nel buffer *scratch* e in altri buffer della modalità di interazione Lisp, C-j (eval-print-last-sexp) valuta l'espressione Lisp prima del punto e inserisce il valore nel punto. Pertanto, mentre si digitano le espressioni nel buffer seguite da C-j dopo ogni espressione, il buffer registra una trascrizione delle espressioni valutate e dei relativi valori. Tutti gli altri comandi in modalità Lisp Interaction sono gli stessi della modalità Emacs Lisp.

All'avvio, il buffer *scratch* contiene un breve messaggio, sotto forma di un commento Lisp, che spiega a cosa serve. Questo messaggio è controllato dalla variabile `initial-scratch-message`, che dovrebbe essere una stringa di documentazione o nil (che significa sopprimere il messaggio).

Comando Term

Qual'ora non vogliamo usare pacchetti per emulare il terminale di Linux, possiamo usare il comando `term` come long command. Per eseguire una subshell in un emulatore di terminale di testo, utilizzare il termine M-x. Questo crea (o riutilizza) un buffer chiamato *terminal*, ed esegue una subshell con l'input proveniente dalla tastiera e l'output va a quel buffer. Quando si usa `term` ci viene chiesta anche la patch della shell di sistema.

L'emulatore di terminale utilizza la modalità Term, che ha due modalità di input. In modalità linea, Term agisce sostanzialmente come la modalità Shell. In modalità char, ogni carattere viene inviato direttamente alla subshell, come input del terminale; l'unica eccezione è il carattere di escape del terminale, che per impostazione predefinita è C-c. Qualsiasi eco del tuo input è responsabilità della subshell; qualsiasi output del terminale dalla subshell va nel buffer, avanzando nel punto

File di configurazione .emacs

Quando Emacs viene avviato, normalmente prova a caricare un programma Lisp da un file di inizializzazione, o init file in breve. Questo file, se esiste, specifica come inizializzare Emacs.

Tradizionalmente, il file `~/.emacs` viene utilizzato come file init, sebbene Emacs guardi anche `~/.emacs.el`, `~/.emacs.d/init.el`, `~/.config/emacs/init.el` o altre posizioni. Tale file si trova in genere nella home directory dell'utente

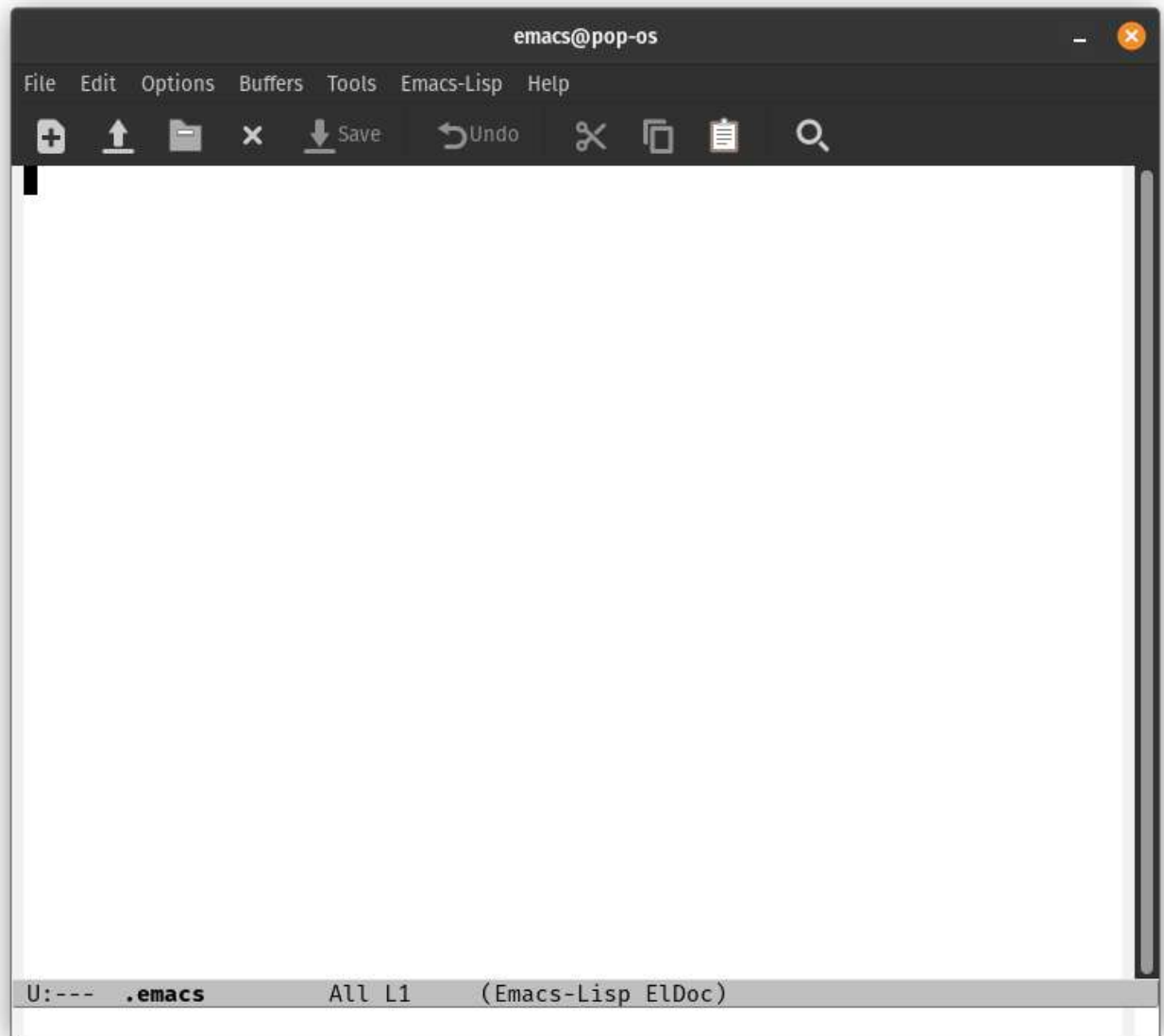
Capito dunque cos'è tale file, andiamo a modificare per personalizzare il nostro Emacs come vogliamo.

Nota: nel momento in cui andiamo ad aprire il file `.emacs`, se vi è del contenuto, a meno che non abbiamo noi scritto in precedenza su tale file, possiamo cancellare il suo contenuto.

Nota: per modificare questo file esistono diversi approcci, io per semplicità editerò direttamente il file `.emacs`, senza usare Org-Mode. È possibile usarlo per avere una gestione migliore del file di configurazione, tutto dipende da quanto siamo intenzionati a scrivere su tale file.

Nota: laddove non vi è questo file di configurazione possiamo crearlo con: **touch .emacs**

Apriamo su Emacs il nostro file di configurazione:



Per eliminare la **menu-bar** scriviamo il seguente codice Elisp:

```
(menu-bar-mode -1)
```

Per eliminare la **toolbar** scriviamo il seguente codice Elisp:

```
(tool-bar-mode -1)
```

Per eliminare la **scroll bar** scriviamo il seguente codice Elisp:

```
(scroll-bar-mode -1)
```

Possiamo andare ad eliminare la schermata Home di **Emacs** qual'ora non ci aggrada, una volta tolta Emacs si aprirà di default nello scratch buffer:

```
(setq inhibit-startup-screen t)
```

Per inserire il numero della riga corrente, utile se si sta programmando, usiamo il seguente codice Elisp:

```
(global-linum-mode 1)
```

Repository Melpa

All'inizio del documento avevamo accennato al repository Melpa, che ci permette di installare svariati pacchetti, quest'ultimi non sono altro che codice Elisp che noi scarichiamo e usiamo per dare molteplici funzionalità al nostro Emacs.

Copiamo le seguenti righe nel nostro file `.emacs`:

```
(require 'package)
(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)
;; Comment/uncomment this line to enable MELPA Stable if desired. See `package-archive-priorities`
;; and `package-pinned-packages`. Most users will not need or want to do this.
;;(add-to-list 'package-archives '("melpa-stable" . "https://stable.melpa.org/packages/") t)
(package-initialize)
```

Questi righe sono disponibili all'indirizzo:

MELPA

The largest and most up-to-date repository of Emacs packages.

 <https://melpa.org/#/getting-started>

Installazione Use-Package

Use-package è un tool molto potente e facile che ci permette di installare dei pacchetti esterni da Melpa in Emacs in un modo “dichiarativo”, un pò come se stessi importando una libreria in un file sorgente. Per usare `use-package` usiamo il seguente codice Elisp:

```
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))
```

Ora vediamo una carrellata di pacchetti, quelli a parer mio più interessanti in termini di estetica e di utilità.

Doom-Modeline

Ecco come installare la doom-modeline:

```
(use-package doom-modeline
  :ensure t
  :init (doom-modeline-mode 1))
```

Per essa ci sono dei piccoli problemi sui simboli, questo perchè dobbiamo installare un pacchetto di font, per farlo usiamo il seguente codice Elisp:

```
(use-package all-the-icons
  :ensure t)
```

Salviamo e riapriamo Emacs, premiamo poi M-x e digitiamo: *all-the-icons-install-fonts* poi digitiamo *yes*.

Neotree

È un file manager che ci permette di dare una visione dei file più carina in Emacs, per installarlo scriviamo:

```
(use-package neotree
  :ensure
  :config
  ;; config to open/close neotree tab and refresh to current folder's files
  (define-key global-map (kbd "C-x C-n") 'neotree-toggle)
)
```

Installazione del tema Spacemacs

Un tema molto interessante è il tema Spacemacs usabile mediante il seguente codice Emacs:

```
(use-package spacemacs-theme
  :ensure t
  :defer t
  :init
  (load-theme 'spacemacs-dark t)
  (setq spacemacs-theme-bg nil))
```

Company-mode

Esso è un pacchetto molto utile perchè permette l'autocompletamento sia di parole che in fase di programmazione, per installarlo:

```
(use-package company
  :ensure t
  :defer t
  :config

  (setq company-idle-delay 0)
  (setq company-minimum-prefix-length 3)
  (setq company-selection-wrap-around t)
  (company-tng-configure-default)
  (setq company-quickhelp-color-background "#4F4F4F")
  (setq company-quickhelp-color-foreground "#DCDCCC")

  (global-company-mode 1))
```

La mia configurazione finale sarà:

```
;; Rimuovi gli elementi grafici
(menu-bar-mode -1)
(tool-bar-mode -1)

;; Inserisci il numero di riga
(global-linum-mode 1)

;; Collegamento a MELPA
(require 'package)
(add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/") t)
(package-initialize)

;; Use-Package
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))

;; Doom-Modline
(use-package doom-modline
  :ensure t
  :init (doom-modline-mode 1))
```

```

;; All-The-Icons
(use-package all-the-icons
  :ensure t)
(custom-set-variables
  ;; custom-set-variables was added by Custom.
  ;; If you edit it by hand, you could mess it up, so be careful.
  ;; Your init file should contain only one such instance.
  ;; If there is more than one, they won't work right.
  '(package-selected-packages
    '(nyan-mode company all-the-icons doom-modeline use-package cmake-mode)))
(custom-set-faces
  ;; custom-set-faces was added by Custom.
  ;; If you edit it by hand, you could mess it up, so be careful.
  ;; Your init file should contain only one such instance.
  ;; If there is more than one, they won't work right.
  )

;; Neotree
(use-package neotree
  :ensure
  :config
  ;; config to open/close neotree tab and refresh to current folder's files
  (define-key global-map (kbd "C-x C-n") 'neotree-toggle)
)

;; Spacemacs-theme
(use-package spacemacs-theme
  :ensure t
  :defer t
  :init
  (load-theme 'spacemacs-dark t)
  (setq spacemacs-theme-bg nil))

;; Company-Mode
(use-package company
  :ensure t
  :defer t
  :config

  (setq company-idle-delay 0)
  (setq company-minimum-prefix-length 3)
  (setq company-selection-wrap-around t)
  (company-tng-configure-default)
  (setq company-quickhelp-color-background "#4F4F4F")
  (setq company-quickhelp-color-foreground "#DCDCCC")

  (global-company-mode 1))

```