Emacs

`Emacs` is a highly customizable and extensible **text editor** with powerful editing features and rich functionality, which is popular among programmers and writers.
Emacs is a software developed as part of the **GNU**.

`GNU` is an **operating system** and a **collection of computer software**.
GNU is an acronym(首字母缩写) for "GNU's Not **Unix** !" because GNU's design is Unix-like but differs from Unix by being free software and containing no Unix code.

`Unix` is a family of multitasking, multi-user computer operating systems that derive from the original AT&T Unix.
Unix has played a pivotal role in the development of modern operating systems, providing key concepts and a design philosophy that continues to influence the field of computing today.

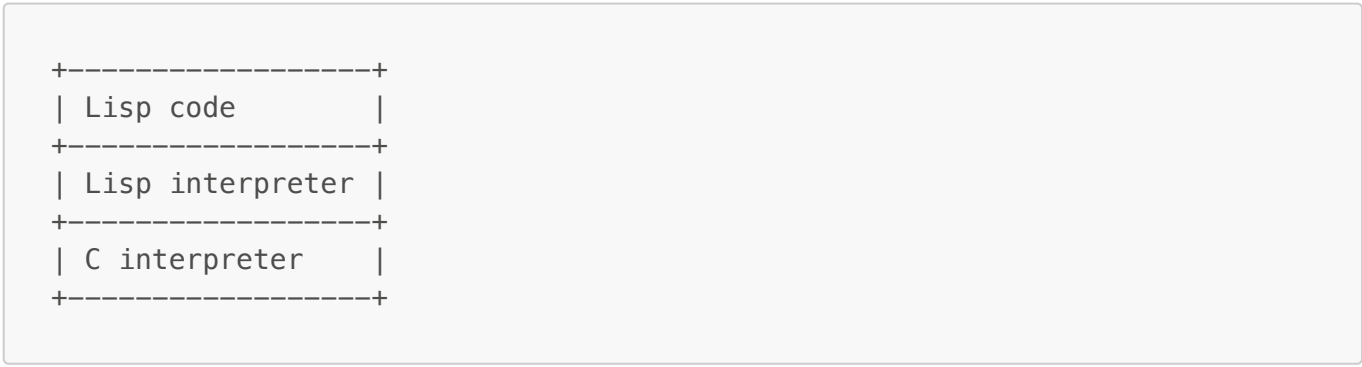
`Linux` refers to the Linux kernel(not an OS) in the strictest sense, most systems that people refer to as "Linux" are actually **GNU/Linux** distributions.

`GNU/Linux` is an OS combines the Linux kernel with GNU utilities and applications. Emacs, being one of the many applications developed under the GNU Project, is included in almost all GNU/Linux distributions as a powerful text editor.

> **Why use Emacs for things like cloud computing even in the modern day?**
>
> Such applications are at the mercy of networking, which are orders of magnitude slower than operations on a local file system. Thus, they care a lot about **throughput(吞吐量)** and **latency(延迟)**, and terminal interfaces can deliver because they are simple and fast.

Implementation

Emacs is built in "layers": it has a C interpreter inside it, and atop it is a Lisp interpreter. The bulk of Emacs is written in Lisp.



The I/O devices like mouse, keyboard, and display communicate with the application through the Lisp code.

Programs typically have their own interpreters embedded in their own executables. For example, Chromium executables come with a JavaScript interpreter included in them.

Concept of Buffers

- Emacs makes use of **buffers** to be *fast*. Buffers are just **a bunch of text that live in RAM**.
- Emacs (and editors in general) makes a clear distinction between what's *persistent* and what's not to balance speed and reliability. For work that is rapidly changing like when you're typing a sentence, we have very performant buffers. Only when work is ready to be saved, the application can flush the buffer to the filesystem in one fell swoop(一次性).

Auto-generated Files

- By convention, file names starting with `.#` indicate a symbolic link to a file that is currently being edited by another program.
- When editing a file `F` in Emacs:
  - Emacs creates a symlink `.#F` that signals other programs that the file is being edited.
  - Emacs creates a file `#F#`, a copy of the unsaved buffer for `F` as part of its auto-save feature, a safety mechanism for in case Emacs or the computer crashes. This file disappears on write.

Editor Navigation: Moving the Point

NOTE: arrow keys (and their Shift and Ctrl variants) work too, but many of the movement keys are actually universal across Unix, which could be useful to know. Also, it's faster to use key binds that are in the main keyboard area instead of having to reach for arrow keys.

The current cursor position is called the **point**.

From **anywhere**:

```
M-g g NUM RET     Go to line number NUM
```

At any point, you can use `C-l` (that's a lowercase L) to vertically center the point if possible. Use `C-l` again to bring the current line to the top of the viewport.

Help System

The help system makes Emacs a "self-documenting" system. `C-h` is the designated **help key**, which prefixes commands related to viewing documentation.

```
C-h b                          list key bindings
C-h f                          describe a function
C-h k KEYSTROKE(S)             list one binding
C-h a <regex> RET              search for a command
M-x apropos RET <query> RET    search for a command
M-x apropos-command RET        like the one above, used to search certain
command
```

Extended Commands

`C-x` is the designated **eXtended command key**, which starts keystroke chords for common commands, like `C-x C-f` to open a file.

`M-x` is the designated **eXtended named command key**, which focuses a command **minibuffer** at the bottom of the terminal, below the mode bar. The minibuffer is itself a buffer supporting your familiar navigation keys, and in it you can write the full name of commands.

After attempting to use the full name for a command that has a `C-x` shortcut, Emacs will tell you what the shortcut is for future use.

There is a special command that allows you to **send a numerical argument to another command**:

```
C-u NUM KEY
```

For many commands, this simply repeats the actions, but some commands may have slightly different behavior. For example:

```
C-u 2 C-k       delete content of two lines and their newlines
C-k C-k         delete content of a line, and then its newline
```

This behavior is determined by how they're implemented - whether they take that optional numeric argument in the first place and what they do with it.

Window and Buffer Navigation

Opening another window for the same buffer does not duplicate the buffer; any edits in one buffer will affect the other(s). You can still use this when you want to reference some other part of the buffer while typing in another region.

Shell within Emacs

> **Q:** What happens if you do emacs within the shell, then do `M-x shell` to start a shell within emacs, and then type emacs within the emacs shell? Does that work, and if so, for how many times?

> **A:** It won't even really work for one iteration because the emacs shell does not emulate(模仿) many of the terminal features that emacs needs to run smoothly.

Selection Manipulation

Concept of the "current region (of current buffer)":

- You can save pointers called **marks** at arbitrary positions within a buffer.
- The **current region** is all the characters between the mark and the point.

You can **set a mark** at the current point with `C-SPC` or `C-@`. An example of selecting a region of text:

```
M-<             go to start of buffer
C-SPC           set marker
C-s eggert RET  search for "eggert" forward
C-r             search backward
```

```
C-M-s           regex search forward
C-M-r           regex seach backward
M-|             pipe buffer into a shell command
```

You can find out where your mark is with `C-x C-x`, which exchanges point and mark (selects the text between them). You can `C-g` to cancel the selection.

Modes

Emacs is a **modeful** editor. That means the current state of Emacs not only includes the contents of the current files being edited but also what way you intend to be using the editor next. A **mode** is like a method of interacting with the editor.

- **Upside:** more efficient for experts
- **Downside:** confusing/tricky for non-experts

```
M-x MODENAME            switch to mode
```

`C-h m` brings up a buffer that describes what mode you are in. The default "editor" mode is called **Fundamental** mode, and there is also **Text** mode in which navigating among words with certain characters like apostrophes is slightly different.

Other M-x functions

```
M-x replace-string                          replace string systematicl,
forward
M-x flush-lines                             delete all lines containing
certain strings
M-x compile RET make -k hello ./hello.c     compiles the C++ file using
cmake (can be gcc if wanted)
M-x load RET file_name RET                  load the file E.x. file.el
M-SPC                                       cycling the indicated space
between 1, 0, original

M-x delete-all-space
M-: (delete-all-space)

M-x: delete-horizontal-space
M-: (delete-horizontal-space)

M-x delete-region
M-: (delete-region START END)

M-x goto-char RET POSITION
M-: (goto-char POSITION)

M-x: insert-char                             (or string)
M-: (insert YOUR_STRING)                     insert
```

```
M-: (elt SEQUENCE INDEX)                    get the element at passed
index
M-: (message “%s” this-command)             prints the name of current
command
M-: (message "%s" last-command)             same as above
M-: (plist-get PLIST PROPERTY)              prints the corresponding value
of passed in
                                            property in the passed in
plist
M-: (prefix-numeric-value &optional ARG)
M-: (functionp object)                      checks if passed in object is
a function
M-: (funcall function &rest arguments)      this call function with
specific arguments.
```

Midterm Question

> You are running Emacs, type `C-x 4 d RET`, and see a new buffer on your screen containing:
>
> `#midterm-answers.txt#`
> This file is likely an **auto-save** file. Emacs periodically saves the content you're editing into an auto-save file. This is not usually the main file but a backup to prevent data loss in case of a crash.
>
> `.#midterm-answers.txt`
> This file appears to be a **lock** file. Emacs uses lock files to indicate that a file is being edited, to prevent concurrent editing sessions on the same file which could lead to conflicts or data loss.
>
> `midterm-answers.txt`
> This is likely the main file that contains the content for the midterm answers.
>
> `midterm-answers.txt~`
> This file is likely a **backup** file. Emacs (and many other text editors) creates a backup of the original file when you start editing, appending a tilde ~ to the end of the filename.

Regular expressions(regex)

Regex are used for matching patterns within strings, and they can vary slightly in syntax and capabilities across different programming languages and tools.

```
Special Characters (without`[]`):
.               any single char except newline  a.c   abc
\               escape a special char    \.\*\+\?\$\^/\\  .*+?$^/\
^               start of string or start of line   ^abc.*   abcksfafkjas
$               end of string or end of line   askf$   ~askf
+               1 or more occurrence     file\w\w+  file9aaaa
*               0 or more times
?               0 or 1 time
]               this is special when it ends a char alternative [ … ]
-               this is special when it is inside of a char alternative
```

```
[a-z]

Character Classes (with `[]`):
[ … ]           one of the chars in the brackets  [aeiou]   a or e or i …
[x-y]           one of the chars in the range from x to y [a-z]   letter
a-z
[^x]            one char that is not x
[^x-y]          one char that is not in the range  [^a-z]   A ? ./
[:ascii:]       POSIX class of all ascii chars
[:alpha:]       POSIX class of all alphabetical chars
[:digit:]       POSIX class of all numerical chars

p1\|p2          any match for p1 or p2
\(p\)           only match for p
[]a-z]          char that is ‘]’ or ‘a’ to ‘z’
[^]a-z]         char that is not ‘]’ or not ‘a’ to ‘z’
[a-z[]          char that is ‘a’ to ‘z’ or ‘[‘
[^^-]           char that is not ‘^’ or  not ‘-‘
[^^]            char that is not ‘^’
[^-]            char that is not ‘-‘

^\d{9}$ == ^[0-9]{9}$
```

How can Character Class (with `[]`) handle special characters?

In a regular expression character class (the part within the square brackets `[...]`), some **special characters** lose their special meaning if they are placed immediately after the opening `[` or just before the closing `]` or escape it with a backslash `\`.

1. `[` - opening bracket

When you want to include a literal opening bracket `[` within a character class, you must escape it with a backslash `\`.

```
[a-z\[]     # valid
[a-z\[1-9]  # valid
[[a-c]      # invalid
[a-c[]      # invalid
```

2. `]` - closing bracket

When including a literal closing bracket `]`, you can either place it immediately after the opening square bracket `[` or using a backslash `\` as escaption.

```
[]a-z]      # valid
[a-z\]1-9]  # valid
[a-z]]      # invalid
```

3. `-` - hyphen

The hyphen `-` has a special meaning in character classes in regular expressions; it's used to specify a range of characters.

However, if you want to include a hyphen as a literal character in a character class, you can 1. place it immediately after the opening square bracket `[` or 2. place it immediately before the opening square bracket `]` or 3. using a backslash `\` as escaption.

```
[-a-c]    # valid
[a-c-]    # valid
[a\-c]    # invalid
```

4. `^` - caret

When placed at the start of a character class, it negates the class, matching anything not specified within the brackets (e.g., [^a-z] matches anything that is not a lowercase letter).

Elsewhere, it's treated as a literal character.

5. `\` - backslash

Used to escape other special characters or signify special character sequences.

Within a character class, it can escape another backslash (e.g., `[\\]` to match a single backslash), a hyphen, or a closing bracket.

6. Special Sequences

Certain character classes are represented by shorthand sequences: `\d` matches any digit (equivalent to [0-9]). `\D` matches any character that is not a digit (equivalent to [^0-9]). `\w` matches any word character (equivalent to [a-zA-Z0-9_]). `\W` matches any character that is not a word character (equivalent to [^a-zA-Z0-9_]). `\s` matches any whitespace character (e.g., space, tab, newline). `\S` matches any character that is not a whitespace character.

NOTE: The underscore `_` is considered a word character in regular expressions because of historical reasons and its common usage in programming and writing. (e.g., user_name vs. username) and allows for names that resemble natural language more closely without spaces.