Emacs

Table of Contents

Competition: Stability & Performance - The Need for Emacs

- apps should be fast, survive outages, survive kernel crashes, and be understandable
  - to solve all of these problems, we must use persistent storage, along with a cache of what is in the persistent storage, or a buffer of what is in the storage
- emacs is useful as 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** and **latency**, and terminal interfaces can deliver because they are simple and fast
  - GUI's constantly have to be reconfigured and do not have the same level of flexibility as terminal use such as EMACs

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 which extends emacs.

```
 +------------------+
| Lisp code        |
+------------------+
| Lisp interpreter |
+------------------+
| C interpreter    |
+------------------+
```

- 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 file system in one fell swoop.
- 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.

Buffer-related Key Binds

```
 (ctrl)-x (ctrl)-b                 # list buffers
(ctrl)-x b <buffer name>          # switch to a buffer


(ctrl)-x 1  # sees only the buffer that is present


(ctrl)-x 2  # splits the current buffer into two views


(ctrl)-x 5  # splits the window horizontally
```

Columns of Emacs

1. Directory
2. Hard Link Count
3. Owner
4. Group
5. Number of Bytes
6. Last Modified
7. Last Modified
8. Last Modified
9. Name or Symbolic Link Contents

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.

File Types in Emacs

```
 ”-”     # regular file
”d”     # directory file
”l”     # symbolic link (treated as a name that serves as a shortcut to another file
```

Basic Emacs Commands

Fixing Mistakes

```
 C-/             # Undo the last command (only if text was modified)
C-g             # Quit current command or exit command minibuffer
ESC ESC ESC     # Universal "get out" - quit command, window, etc.
C-x C-s         # Save current file
C-x C-f         # Open/return to a file, creating it if necessary
C-x C-c         # Exit Emacs
C-z             # Temporarily suspend Emacs (enter fg to re-enter)
```

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                          # bashlist key bindings
C-h k KEYSTROKE(S)             # bashlist one binding (what happens in keystroke)
C-h a <regex> RET              # bashsearch for a command
M-x apropos RET <query> RET    # search for a command
```

Shell within Emacs

Run a command as a separate, one-off shell process:

```
 M-! <command> RET
M-! foo bar | tr a b | sort
```

Same thing but taking input from a buffer and then piping it to the shell command (takes all characters in **current region** and pipes them to the command as its stdin, and then takes the output of the command and pipes it to the *shell command output buffer* like normal:

```
 M-| <command> RET
```

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-@                 # set marker
C-s eggert RET      # search for "eggert"
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.

Text Manipulation

Emacs distinguishes **killing** from **deleting**. When you **kill** text, it is actually saved in a special buffer called the **kill ring**, and content in here can be reinserted (aka **yanked**) at the point. Most commands that perform bulk removal of text actually *kill* the text, not *delete* it.

```
 C-k       # kill from point to end of line
C-w       # kill current selection
M-w       # copy current selection
C-y       # yank most recent kill
M-y       # cycle the text to yank through kill history
```

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.

You can also open a shell subprocess (**Shell** mode) or a view of a directory (**dired**, directory-editing mode). The mode you are in affects the keys you input. You can see the name of the major mode you are currently in with the **mode bar** just above the minibuffer.

```
C-x d <dirname> RET     enter dired mode for directory
```