The Author
Li Mingming

The Full Book of

Emacs

How I use Emacs as my main editor

Sep 29, 2020

Dedication

The first text editor I systematically learn is actually **vim**. As the default text editor on Linux, I use it to write shell script and edit configuration files on Linux OS. Once, I saw the debate between Emacs and Vim on Internet. At that time, I had never used Emacs, so I couldn't say which one is more suitable to me. I learned Emacs at all my spare time and was facinated by the functionality and power of it. I dicide to use Emacs as my main text editor expecially having read "Hacker and Painter".

The book I write here is with Emacs's LaTeXmode (AUCTex acutally).

Contents

List of Tables

List of Figures

Chapter 1

Emacs Basics

1.1 What is Emacs

Emacs is an text editor.

> Learning to use an editor is basically a matter of learning finger habit. Good finger habits can make you an incredibly fast typist. Intellectually, it's possible to absorb a lot from one reading, but you can form only a few new habit each day. Don't feel obliged to learn them all at once; pick something, practice it, and move on to the next topic. Time spent developing good habits is time well spent.

1.2 Why I You Use Emacs

Efficiency: There are many commands that can move cursor without a mouse.

Function: It can do many things like writting LaTeX, Python.

Extensibility: If there is some function not meeting my need, I can programm to implement it using Emacs Lisp language.

1.3 Files and Buffers

You don't really edit files. Instead, Emacs copies the content of a file into a temporary buffer and you edit that. The file on disk doesn't change until you save the buffer.

1.4 Modes

Emacs becomes sensitive to the task at hand. **Modes** allows Emacs to the kind of editor you want for different tasks. A buffer can be in only one **major mode** at a time. Minor modes defines a particular aspect of Emacs's behavior and can be turned on and off within a major mode.

If you are good at Lisp programming, you can add your own modes. Emacs is almost infinitely extensible.

1.5 Commands

You issue commands to instruct Emacs to do what you want to. Each **command** has a formal name, which is the name of a Lisp routine (Emacs is written with Emacs Lisp language). Some command names are quite long. As a result, we need some way to abbreviate commands. Emacs ties a command name to a short sequence of keystrokes. This tying of commands to keystrokes is known as **binding**.

The author of Emacs try to bind the most frequently used commands to the key sequences that are the easiest to reach (C: Ctrl, M: Meta):

1. The most commonly used commands are bound to C-n(where n is any character).
2. Slightly less commonly used commands are bound to M-n.
3. Other commonly used commands are bound to C-x something.
4. Some specialized commands are bound to C-c something. These commands often relate to one of the more specialized modes, such as Java or HTML mode.
5. typing M-x long-command-name Enter.(This works for any command really)

Chapter 2

Get Help

C-h C-h Visit the overall get help tutorial.

C-h k Get help on the key bounding you typed.

C-h f Get help on the function.

C-h v Get help on the variable.

C-h a Show commands whose name matches the PATTEN.

M-x version Get the current Emacs version.

F10 Move focus onto the top menu bar.

C-h m Get help on current major mode and minor mode.

Chapter 3

Commands

3.1 Basic Commands

C-x C-f Open or create a file.

C-x i Insert a file's content into the current cursor.

C-x C-s Save the current buffer.

C-x C-w Write current buffer into the file.

C-x C-c Exit Emacs.

C-/ Undo the previous command.

C-q useful for inserting control character

M-n `n` stands for numbers

C-u Begin a numberic argument.

M– Begin a negative numeric argument.

C-l Move current buffer line to the specified window line.

C-g Quit.

C-x Esc Esc Edit and re-evaluate last complex command from last.

C-x z Repeat most recently executed command.

M-/ Expand previous word "dynamically".

M-m Move point to the first non-whitespace character on this line.

C-M-i Complete the current symbol.

3.2 Movement

C-f forward-char

C-b backward-char

M-f forward-word

M-b backward-word

C-n next-line

C-p previous-line

C-a Move to the beginning of the current line.

C-e Move to the end of the current line.

M-a backward-sentence

M-e forward-sentence

M-} forward-paragraph

M-{ backward-paragraph

C-x] forward-page

C-x [backward-page

C-v Move down one screen.

M-v Move up one screen.

M-> Move to the end of the buffer.

M-< Move to the beginning of the buffer.

M-g M-g Go to the specified line.

C-M-v Scroll other window one screen.

3.3 Deleting Text

Del Delete backward one character.

C-d Delete one character.

M-Del Delete backward one word.

M-d Delete one word.

C-k Kill one line from the current position.

M-k Kill one sentence.

C-x Del Delete one sentence backward.

3.4 Yank

In Emacs, killing is not fatal, but in fact, quite the opposite. Text that has been killed is not gone forever but is hidden in an area called the kill ring. The kill ring is an internal storage area where Emacs put things you've copied or deleted.

C-y Yank the most recently delete text.

M-y Yank pop.

3.5 Marking Text

C-SPC Set mark.

C-@ Set mark.

C-x C-x exchange point and mark.

M-h Mark the current paragraph.

C-x h Mark the whole buffer.

C-w Kill the marked region.

M-w Copy the marked region.

3.6 Editing

C-t Transpose character.

M-t Transpose word.

C-x C-t Transpose lines.

M-c Capitilize the word from current position.

M-u Upcase the word from current position.

M-l Downcase the word from current position.

3.7 Search

C-s Search incrementally.

C-r Search incrementally backward.

Enter exit incremental search.

C-s C-w Search the word under the cursor.

C-s C-s Repeat search forward.

C-r C-r Repeat search backward.

M-% Query and replace.

C-M-s Search regular expression forward.

C-M-r Search regular expression backward.

C-M-% Search regular expression replace

Chapter 4

Using Buffers, Windows, and Frames

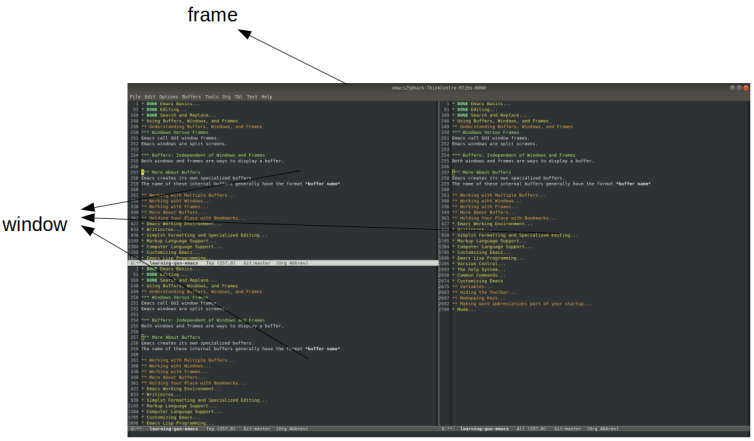

Figure 4.1: Frame and Window

Emacs call GUI window frames. Emacs windows are split screens.

Both windows and frames are ways to display a buffer. Emacs creates its own specialized buffers. The name of these internal buffers generally have the format **buffer name**.

4.1 Buffers

C-x b Switch to the most recently opened buffer.

C-x C-b List all buffers.

C-x k Kill the current buffer.

C-x s Save all buffers.

C-x C-q toggle read only mode

C-x 4 r open a file as read-only in a new window

C-x 5 r open a file as read-only in a new frame

4.2 Windows

C-x o Switch to other window.

C-x 0 Delete the current window.

C-x 1 Delete other windows.

C-x 2 Split window vertically.

C-x 3 Split window right.

C-x ^ Enlarge window.

C-x } Enlarge window horizontally.

C-x { Shrink window horizontally.

C-x - Shrink window if larger than buffer.

C-x + Balance window.

4.3 Frames

C-x 5 o other-frame

C-x 5 0 delete-frame

C-x 5 1 delete-other-frame

C-x 5 2 make-frame-command

Chapter 5

Dired

Entering Dired mode with `C-x C-f` adding a directory.

5.1 Executing Commands in Shell Buffers

M-! Execute command string in inferior shell; display output, if any. With `C-u` prefix, insert the output into the current buffer.

M-x shell Run an inferior shell.

M-| Run shell command on region.

You can have multiple shell buffers running at once; just use the command `M-x rename-uniquely` to rename your shell buffer. Then type `M-x shell` to open a new one.

5.2 Operations

v View file read only, q to quit.

RET Find the file, `C-x b` to get back.

x Delete the deleted-flagged files.

D Do delete.

C Do copy.

R Do rename.

Z Compress or uncompress marked or next ARG files.

= Run the `diff` command on the given two files.

! Run shell commond.

g Replace current buffer text with the text of the visited file on disk.

+ Create a directory.

Q Replace matches of FROM with TO, in all marked files.

A Find all matches for REGEXP in all marked files.

5.3 Marking

m Mark the current file.

3m Mark the current file and the next 2 files.

M-Del Unmark all files.

U Unmark all files.

t Toggle marks.

*** *** Mark excutable files.

*** @** Mark symlinks.

*** /** Mark directories.

% m Mark files matching REGEXP.

% g Mark files containing REGEXP.

d Flag file as deletion.

Flag autosave files.

~ Flag backup files.

u Unmark the current file.

Del Unmark backward.

5.4 Navigating

^ Up directory.

> Next directory line.

< Previous directory lien.

i Insert this subdirectory into the same dired buffer.

Chapter 6

Macro

In Emacs, a macro is simply a group of recorded keystrokes you can play back over and over again. Macros are a great way to save yourself repetitive work.

6.1 Commands

C-x (Start a macro definition.

C-x) End a macro definition.

C-x e Call the last defined macro.

C-x C-k e Edit a keyboard macro.

C-x C-k C-d Delete current macro from keyboard macro ring.

C-x C-k C-t Swap first two elements on keyboard macro ring.

C-x C-k C-p Move to previous keyboard macro in keyboard macro ring.

C-x C-k C-n Move to next keyboard macro in keyboard macro ring.

C-x C-k b Offer to bind last macro to a key.

C-x C-k n Assign a name to the last keyboard macro defined.

C-x C-k r Apply last keyboard macro to all lines in the region.

C-x q Query user during kbd macro execution.

C-M-c Exit from the innermost recursive edit or minibuffer.

6.2 Writting Interactive Macro

1. Use `C-x (` to start a macro definition.
2. Use `C-x q` to insert a query during kbd macro.
3. Use `C-r` to start a recursive edit during executing the macro.
4. Use `C-M-c` to exit the recursive edit.
5. Use `Y` to continue to the next interactive edit or others.

6.3 Tips for Creating Good Macros

Good macros work in all situations. Here are some tips:

1. Use commands that are absolute rather than relative.
2. Type the search argument (C-s) rather than using the command to repeat the last search (C-s C-s).
3. Add extra commands (typically C-a and C-e) that aren't strictly necessary, just to make sure that you're positioned correctly on the line.

6.4 Naming, Saving, and Executing Your Macros

1. Define a macro.
2. name it with `C-x C-k n`.
3. Open a file.
4. `M-x insert-kbd-macro RET <macroname> RET`.
5. add `(load-file "<your macro file>")` to .emacs.
6. add `(global-set-key "\C-x\C-k<your key>" '<your macro name>)` to .emacs.

Chapter 7

Rectangle Editing

C-x r k Delete the region-rectangle and save it as the last killed one.

C-x r y Yank the last killed rectangle with upper left corner at point.

C-x r c Blank out the region-rectangle.

C-x r o Blank out the region-rectangle, shift text right.

C-x r r Copy rectangular region into register REGISTER.

C-x r i Insert contents of register REGISTER. (REGISTER is a character.)

Chapter 8

Cusomization

Three ways to customize Emacs:

1. Options
2. Custom
3. Lisp code

No matter what method you use, though, the .emacs startup file is modified. Custom modifies it for you when you save settings through that interface. The Options menu invokes Custom behind the scenes; when you choose Save Options, Custom again modifies .emacs.

8.1 Hide Tool Bar

Refer to the Figure: 8.1.
After the hiding, remember to press the `Save Options`.

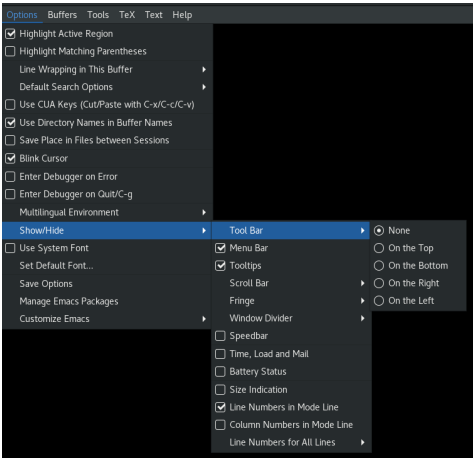

Figure 8.1: Hide tool bar

8.2 Theme

Refer to the Figure: 8.2.

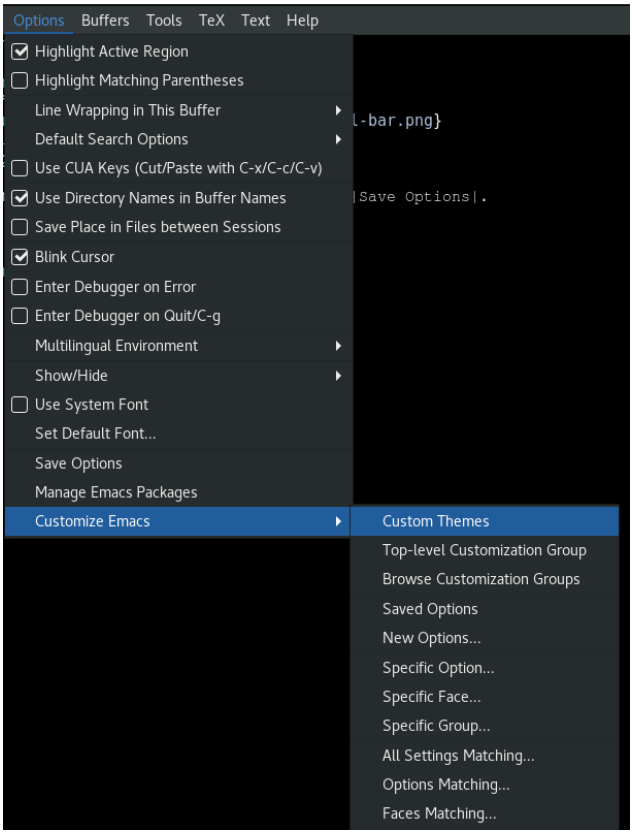

Figure 8.2: Theme

After the hiding, remember to press the `Save Options`.

8.3 Package Repository

MELPA (Milkypostman's Emacs Lisp Package Archive) is a emacs lisp package repository.

Add the following Emacs Lisp code into you `init.el` or `.emacs` file.

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

8.4 Some Customization

```
1  ;; install packages
2  (defvar my-packages
3    '(better-defaults   ; set up some better Emacs defaults
4      material-theme   ; Theme
5      )
6    )
7
8  ;; scans the list in my-packages
9  ;; if the package listed is not already installed, install it
10 (mapc #'(lambda (package)
11       (unless (package-installed-p package)
12         (package-install package)))
13       my-packages)
14
15
16 ;; basic cusomization
17 (setq inhibit-startup-message t) ; hide the startup message
18 (load-theme 'material t)  ; load material theme
19 (global-linum-mode t)   ; enable line numbers globally
```

8.5 Customize Default Directory

```
(setq default-directory "~/")
```

8.6 Enter Full Screen on Startup

```
(toggle-frame-fullscreen) ; enter fullscreen on startup
```

8.7 Set Default Font

The formal configuration is:

1. Options → Set Default Font
2. After you select you font, press Options → Save Options

However this configuration does not work on my MacOS. Here's the oprations that work on MacOS:

1. Options → Set Default Font
2. M-x desribe-font
3. Copy the name (Here is my first line "name (opened by): -*-Menlo-normal-normal-normal-*-9-*-*-*-m-0-iso10646-1")
4. add the the Elisp code "(add-to-list 'default-frame-alist '(font . "-*-Menlo-normal-normal-normal-*-9-*-*-*-m-0-iso10646-1"))"

Chapter 9

Org Mode

When you open a file that ends with `.org`, the major mode is Org. You can alse enter Org mode with the command `M-x org-mode`.

9.1 TODO Items

9.1.1 TODO Basics

Any headline becomes a TODO item when it starts with the word "TODO", for example:

```
*** TODO This is a todo headline
```

The most important commands to work with TODO entries are:

Bounding	Command	Explaination
C-c C-t	org-todo	rotate the TODO state
S-←	org-shiftleft	select the preceeding TODO state
S-→	org-shiftright	select the following TODO state
C-c /	org-sparse-tree	create a sparse tree
S-M-RET	org-insert-todo-heading	insert a new TODO entry below the current one

Table 9.1: TODO commands

9.1.2 Multi-state Workflow

```
(setq org-todo-keywords
          '((sequence "TODO" "|" "DONE" "DELEGATED")))
```

The vertical bar separates the "TODO" keywords (states that **need action**) from the "DONE" states (which need **no further action**). If you do not provide the separator bar, the last state is used as the "DONE" state.

Sometimes you may want to use different sets of TODO keywords in parallel.

```
(setq org-todo-keywords
'((sequence "TODO(t)" "|" "DONE(d)")
            (sequence "REPORT(r)" "BUG(b)" "KNOWNCAUSE(k)" "|"
```

For state logging, Org mode expects configuration on a per-keyword basis. This is achieved by adding special markers '!' (for a timestamp) and '@' (for a note) in parentheses after each keyword. For example:

```
#+TODO: TODO(t) WAIT(w@/!) | DONE(d!) CANCELED(c@)
```

defines TODO keywords and fast access keys, and also request that a time is recorded when the entry is set to 'DONE', and that a note is recorded when switching to 'WAIT' or 'CANCELED'. The same syntax works also when setting `org-todo-keywords`.

To define TODO keywords that are valid only in a single file, use the following text anywhere in the file.

```
#+TODO: TODO(t) | DONE(d)
#+TODO: REPORT(r) BUG(b) KNOWNCAUSE(k) | FIXED(f)
#+TODO: | CANCELED(c)
```

9.1.3 Progress Logging

To record a timestamp and a note when changing a TODO state, call the command org-todo with a prefix argument.

```
C-u C-c C-t
```

Prompt for a note and record a the time of the TODO state change.

9.1.4 Closing Items

The most basic logging is to keep track of when a certain TODO item was marked as done. This can be achieved with[1]

```
(setq org-log-done 'time)
```

Then each time you turn an entry from a TODO (not-done) state into any of the DONE states, a line "CLOSED: [timestamp]" is inserted just after the headline.

If you want to record a note along with the timestamp, use[2]

[1] The corresponding in-buffer setting is #+STARTUP: logdone.
[2] The corresponding in-buffer setting is #+STARTUP: logenotedone

```
(setq org-log-done 'note)
```

You are then be prompted for a note, and that note is stored below the entry with a "Closing Note" heading.

9.1.5 Drawer

You might want to keep track of TODO state changes. You can either record just a timestamp, or a time-stamped note for a change. These records are inserted after the headline as an itemized list. When taking a lot of notes, you might want to get the notes out of the way into a drawer. Customize the variable `org-log-into-drawer` to get this behavior.

9.1.6 Priorities

Prioritizing can be done by placing a **priority cookie** into the heading of a TODO item, like this

```
*** TODO [#A] This is a high priority task
```

Org mode supports three priorities: 'A', 'B', and 'C'. 'A' is the highest, 'B' the default if none is given.

9.1.7 Breaking Tasks Down into Subtasks

It is often advisable to break down large tasks into smaller, manageable subtasks. You can do this by creating an outline tree below a TODO item, with detailed subtasks on the tree. To keep an

Binding	Command	Explaination
C-c ,	org-priority	change the priority of an item
S-*uparrow*	org-priority-up	
increase the priority of the current headline		
S-*downarrow*	org-priority-down	
decrease the priority of the current headline		

Table 9.2: Priorities

overview of the fraction of subtasks that have already been marked as done, insert either '[/]' or '[%]' anywhere in the headline. These cookies are updated each time the TODO status of a child changes, or when pressing C-c C-c on the cookie. For example:

```
* Organize Party [33%]
** TODO Call people [1/2]
*** TODO Peter
*** DONE Sarah
** TODO Buy food
** DONE Talk to neighbor
```

9.2 Dates and Times

9.2.1 Timestamps

A timestamp is a specification of a date. Its presence causes entries to be shown on specific dates in the agenda.

Plain timestamps `2006-11-01 Wed 19:15`

Timestamp with repeater interval `2007-05-16 Wed 12:30 +1w`

Diary-style expression entries `<%%(diary-float t 4 2)>`

Time/Date range `<2004-08-23 Mon>--<2004-08-26 Thu>`

Inactive timestamp `[2006-11-01 Wed]` These timestamps are inactive in the sense that they do not trigger an entry to show up in the agenda.

9.2.2 Creating Timestamps

C-c . Prompt for a date and insert a corresponding timestamp

C-c ! insert an inactive timestamp

S-←

S-→ Change date at point by one day

S-↑ On the beginning or enclosing bracket of a timestamp, change its type. Within a timestamp, change the item under point.

S-↓

9.2.3 Deadlines and Scheduling

A timestamp may be preceded by special keywords to facilitate planning:

C-c C-d Insert 'DEADLINE' keyword along with a time stamp, in the line following the headline.

C-c C-s Insert 'SCHEDULED' keyword along with a stamp, in the line following the head- line.

9.2.4 Clocking Work Time

C-c C-x C-i Start the clock on the current item (clock-in). When called with a C-u prefix argument, select the task from a list of recently clocked tasks.

C-c C-x C-o Stop the clock (clock-out).

C-c C-x C-e Update the effort estimate for the current clock task.

C-c C-x C-q Cancel the current clock.

C-c C-x C-j Jump to the headline of the currently clocked in task. With a C-u prefix argument, select the target task from a list of recently clocked tasks.

9.3 Agenda Views

Due to the way Org works, TODO items, time-stamped items, and tagged headlines can be scattered throughout a file or even a number of files. To get an overview of open action items, or of events that are important for a particular date, this information must be collected, sorted and displayed in an organized way.

The extracted information is displayed in a special agenda buffer. This buffer is read-only, but provides commands to visit the corresponding locations in the original Org files, and even to edit these files remotely.

9.3.1 Agenda Files

The information to be shown is normally collected from all agenda files, the files listed in the variable `org-agenda-files`.

C-c [Add the current file to the list of the agenda files.

C-c] Remove the current file from the list of agenda files.

C-,

C-' Cycle through agenda file list, visiting one file after the other.

9.3.2 The Agenda Dispatcher

The views are created through a dispatcher, accessible with `M-x org-agenda`. It displays a menu from which an additional letter is required to execute a command.

```
Press key for an agenda command:
--------------------------------        <   Buffer, subtree/region restriction
a   Agenda for current week or day      >   Remove restriction
t   List of all TODO entries            e   Export agenda views
m   Match a TAGS/PROP/TODO query        T   Entries with special TODO kwd
s   Search for keywords                 M   Like m, but only TODO entries
/   Multi-occur                         S   Like s, but only TODO entries
?   Find :FLAGGED: entries              C   Configure custom agenda commands
*   Toggle sticky agenda views          #   List stuck projects (!=configure)

n   Agenda and all TODOs: set of 2 commands
```

Figure 9.1: Org agenda

9.3.3 Commands in the Agenda Buffer

Motion

n next line

p previous line

View/Go to Org File

SPC Display the original location of the item in another window.

TAB Go to the original location of the item in another window.

RET Go to the original location of the item and delete other windows.

Change Display

o Delete other windows.

v d or short d Switch to day view.

v w or short w Switch to week view.

f Go forward in time to display the span following the current one.

b Go backward in time to display earlier dates.

. Go to today.

j Prompt for a date and go there.

v l or short l Toggle Logbook mode.

r

g Recreate the agenda buffer.

s Save all Org buffers in the current Eamcs session, and also the location of IDs.

Remote Editing

0–9 Digit argument.

t Change to the TODO state of the item, both in the agenda and in the original Org file.

C-k Delete the current agenda item along with the entire subtree belonging to it in the original Org file.

C-c C-s Schedule this item. With a prefix argument, remove the scheduling timestamp.

C-c C-d Set a deadline for this item. With a prefix argument, remove the deadline.

S-→ Change the timestamp associated with the current line by one day into the future.

S-← Change the timestamp associated with the current line by one day into the past.

I Start the clock on the current item.

O Stop the previously started clock.

X Cancel the currently running clock.

J Jump to the running clock in another window.

Quit and Exit

q Quit agenda, remove the agenda buffer.

x Exit agenda, remove the agenda buffer and all buffers loaded by Emacs for the compilation of the agenda.

9.3.4 Integration of Agenda and Calendar

In order to include entries from the Emacs diary into Org mode's agenda, you only need to customize the variable

```
1 (setq org-agenda-include-diary t)
```

Chapter 10

Python IDE

After adding the MELPA into your `.emacs`, install the following packages.

10.1 elpy

The **elpy** package (Emacs Lisp Python Environment) provides a near-complete set of Python IDE features, including:

- Automatic indentation
- Syntax highlighting
- Auto completion
- Syntax checking
- Python REPL integration
- Virtual environment support

To install and enable elpy, you add the package to your Emacs configuration.

```
;; install packages
(defvar my-packages
  '(better-defaults   ; set up some better Emacs defaults
    material-theme   ; Theme
    elpy    ; Emacs Lisp Python Environment
    )
  )

;; enable elpy
(elpy-enable)
```

You can configurate the `elpy` with `M-x elpy-config`.

> If there is a bug: `melpa.org/packages/company-20200725.2348.tar not i`
> Try to run the command `package-refresh-contents` to refresh your package database.

10.2 Syntax Checking

By default, `elpy` uses a syntax-checking package called `flymake`. While `flymake` is built into Emacs, it only has native support four languages, and it requires significant effort to be able to support new languages.

The syntax-checking package `flycheck` supports real-time syntax checking in over 50 languages and is designed to be quickly configured for new languages.

```
;; install packages
(defvar my-packages
  '(better-defaults   ; set up some better Emacs defaults
    material-theme   ; Theme
    elpy    ; Emacs Lisp Python Environment
    flycheck    ; on the fly syntax checking
    )
  )

;; enable flycheck
(when (require 'flycheck nil t)
  (setq elpy-modules (delq 'elpy-module-flymake elpy-modules))
```

```
13   (add-hook 'elpy-mode-hook 'flycheck-mode))
```

10.3 Code Fomatting

```
1  ;; install packages
2  (defvar my-packages
3    '(better-defaults   ; set up some better Emacs defaults
4      material-theme   ; Theme
5      elpy    ; Emacs Lisp Python Environment
6      flycheck    ; on the fly syntax checking
7      py-autopep8    ; run autopep8 on save
8      blacken    ; black formatting on save
9      )
10   )
11
12 ;; enable autopep8
13 (require 'py-autopep8)
14 (add-hook 'elpy-mode-hook 'py-autopep8-enable-on-save)
```

At this end, when you save your code Emacs will format your code automately.

10.4 Python Interpreter

10.5 Itegration With Jypyter and IPython

```
1  ;; Use IPython for REPL
2  (setq python-shell-interpreter "jupyter"
3        python-shell-interpreter-args "console␣--simple-prompt"
4        python-shell-prompt-detect-failure-warning nil)
5  (add-to-list 'python-shell-completion-native-disabled-interpreters
6               "jupyter")
```

This will update Emacs to use IPython rather than the standard Python REPL.

The `ein` package enables IPython Notebook in Emacs.

```
1
2  ;; install packages
```

```
(defvar my-packages
  '(better-defaults   ; set up some better Emacs defaults
    material-theme   ; Theme
    elpy    ; Emacs Lisp Python Environment
    flycheck    ; on the fly syntax checking
    py-autopep8    ; run autopep8 on save
    blacken    ; black formatting on save
    ein    ; Emacs IPython Notebook
    )
  )
```

To start the server, use the command `M-x ein:jupyter-server-start`.

10.6 Debugging

The built-in **python-mode** allows you to use Emacs for Python code debugging with `pdb`.

> While the `pdb` executable may exist in some Python distribution, it doesn't exist in all of them. You can set the variable `gud-pdb-command-name` to define the command used to launch pdb. Add `(setq gud-pdb-command-name "python -m pdb")` to `.emacs` file.

Use `M-x pdb` to start the Python debugger.

You can step through code in `pdb` using one of two keys:

s steps into other functions.

n steps over other functions.

10.7 Git Support

In Emacs, source control support is provided by the `magit` package.

```
1
2  ;; install packages
3  (defvar my-packages
4    '(better-defaults   ; set up some better Emacs defaults
5      material-theme   ; Theme
6      elpy    ; Emacs Lisp Python Environment
7      flycheck    ; on the fly syntax checking
8      py-autopep8    ; run autopep8 on save
9      blacken    ; black formatting on save
10     ein    ; Emacs IPython Notebook
11     magit    ; Git integration
12     )
13   )
```

`M-x magit-status`.

Chapter 11

TeX Mode

Emacs comes with a package for editing TeX and LaTeX files. However, this package is extremely limited in its functionality. A far better package called **AUC TeX** can help you write your papers dfficiently.

11.1 Installation

The modern and stronly recommended way of installing AUCTeX is by using the Emacs package integrated in Emacs and greater(ELPA). Simply do `M-x list-packages RET`, mark the auctex package for installation with `i` and hit `x` to execute the installation procedure. That's all.

But on my CentOS 8, there is no **auctex** on MELPA repository, So I compile it from source.

1. Download the source file. auctex
2. Unzip the source file.
3. change to the unzipped directory
4. `./configure`

5. `make`

6. `make install`

To uninstall AUCTeX, `make uninstall`.

Note: AUCTeX is more strict on the LaTeX! There is a error saying `! Undefined control sequence. l.1218 \ExplSyntaxOn`. I comment **ExplSyntexOn** out to solve the problem Temperately

11.2 Common Commands

C-c { Make a pair of braces.

C-c C-s Insert a sectioning command.

C-c C-e Insert a environment.

C-M-a Move point to the `\begin` of the current environment.

C-M-e Move point to the `\end` of the current environment.

M-RET Insert an item.

C-M-i Complete TeX symbol before point.

C-c C-m Prompt for the name of a TeX macro.

C-c * Mark the current section.

C-c . Mark the current environment.

C-c % Add or remove '%' from the beginning of each line in the current paragraph.

C-c C-q C-p or M-q Fill and indent the current paragraph.

C-c C-q C-e Fill and indent the current environment.

C-c C-q C-s Fill and indent the current section.

11.3 Folding Macros and Environments

A popular complaint about markup languages like TeX and LaTeX is that there is too much clutter in the source text and that one cannot focus well on the content. With AUCTeX's folding functionality you can collapse those items and replace them by a fixed string, the content of one of their arguments, or a mixture of both. In order to use this feature, you have to activate `TeX-fold-mode` which will activate the auto-reveal feature and the necessary commands to hide and show macros and environments. You can activate the mode in a certain buffer by typing the command `M-x TeX-fold-mode RET` or using the keyboard shortcut `C-c C-o C-f`. If you want to use it every time you edit a LaTeX document, add it to a hook:

```
1 (add-hook 'LaTeX-mode-hook (lambda () (TeX-fold-mode 1)))
```

If it should be activated in all AUCTeX modes, use `TeX-mode-hook` instead of `LaTeX-mode-hook`.

Once the mode is active there are several commands available to hide and show macros, environments and comments:

C-c C-o C-b Hide all foldable items in the current buffer according to the setting of `TeX-fold-type-list`.

C-c C-o C-r Hide all configured macros in the marked region.

C-c C-o C-p Hide all configured macros in the pararaph containing point.

C-c C-o C-e Hide the current environment.

C-c C-o C-c Hide the comment point is located on.

C-c C-o b Permanently unfold all macros and environments in the current buffer.

C-c C-o r Permanently unfold all macros and environments in the marked region.

C-c C-o p Permanently unfold all macros and environments in the pararaph containing point.

C-c C-o i Permanently show the macros or environment on which point currently is located.

C-c C-o C-o Hide or show items according to the current context.

11.4 Narrowing

C-x n g Make text outside current group invisible.

C-x n e Make text outside current environment invisible.

C-x n w Unnarrow.

11.5 Documentation about Macros and Packages

C-c ? Get documentation about the packages installed on your system, using 'texdoc' to find the manuals.

Chapter 12

Version Control

A version control system gives you automated help at keeping a change history for a file or group files. It allows you to recover an stage in that directory, and it makes getting reports on the different between versions easy.

12.1 How VC Helps with Basic Operations

Historically, you had to know **three or four** different shell commands to do the basic operations of version control (registration, check in, check out, and revert). This procedure was complicated and annoying, or at best a distraction from the flow of working on your code and changes.

VC's interface is much simpler. The simplicity comes from noticing that whatever state your version-controlled file is in, there is normally just **one** logical thing to do next. Thus, VC mode has just one basic command: `C-x v v` (for `vc-next-action`), which you can think of as "do the next logical thing to this file".

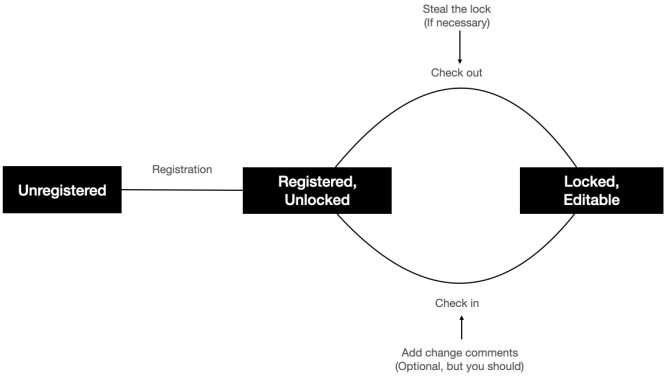

Figure 12.1: VC status

12.2 Editing Comment Buffers

In VC mode, three operations typically pop up a buffer to accept comment or notification text:

- check in
- lock stealing
- file registration

In each case, the operation is on hold until you type `C-c C-c` to commit the comment buffer.

The comment buffer is a plain-text buffer. However, each time you commit a comment buffer, the contents are saved to a new slot in a ring of comment buffers. You can cycle backwards in the ring with

`M-p` and forward with `M-n` , or you can search for text backwards in the ring with `M-r` and forward with `M-s` . By design, these are the same keys you can use to navigate an Emacs minibuffer command history.

12.3 VC Command Summary

C-x v v Go to the next logical version control state.

C-x v d Show all files beneath a directory.

C-x v = Display diffs between file revisions.

C-x v u Revert working copies of the selected fileset to their repository contents.

C-x v Visit revision REV of the current file in another window.

C-x v l Display a file's change comments and history.

C-x v i Register a file for version control.