Emacs

Mingming Li

First Created: September 29, 2020
Last Modified: February 26, 2023

Contents

List of Figures

List of Tables

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 You Should Use Emacs

There are following reasons to use Emacs:

- **Efficiency**: There are many commands that can move cursor without a mouse. There are many commands to help you saving typing.
- **Powerful**: It provides many major and minor modes for special situation. I often use the latex mode.
- **Extensibility**: If there is some function not meeting your need, you can programm to implement it using Emacs Lisp language.

1.3 Files and Buffers

In Emacs, you don't really edit **files** (stored on disk). 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.

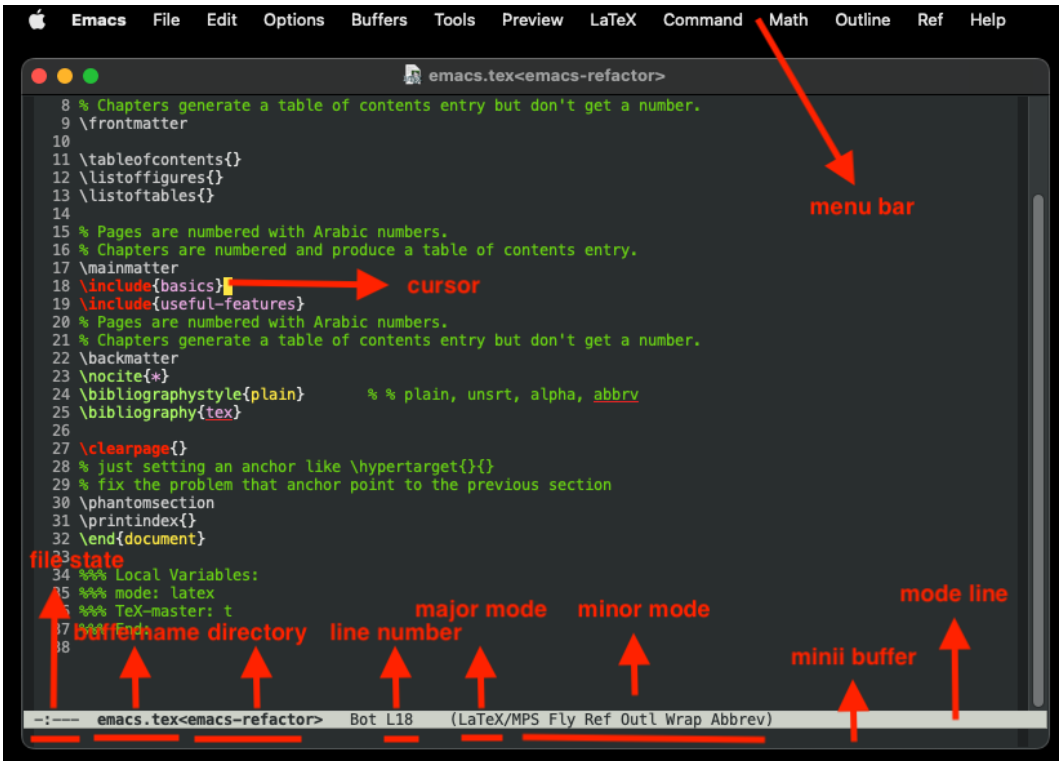
1.4 Modes

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 Display

The display is shown in Figure 1.1.



Figures 1.1: Display

Here I have configured Emacs to hide the tool bar. The operation is: Options ⟶ Show/Hide ⟶ Toolbar. After that, you will see the following file in `/.emacs`:

```
(custom-set-variables
 '(tool-bar-mode nil))
```

The **mode line** is the place to some useful information. The **file state** shows the state of the current buffer. For example, it is read only or not, altered or not.

1.6 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**):

- ♠ The most commonly used commands are bound to **C-n**(where **n** is any character).
- ♠ Slightly less commonly used commands are bound to **M-n**.
- ♠ Other commonly used commands are bound to **C-x keystrokes**.

- ♠ Some specialized commands are bound to **C-c keystrokes**. These commands often relate to one of the more specialized modes, such as Java or HTML mode.
- ♠ To use commands that is binded, use **M-x command-name Enter**. (This works for any command really)

1.7 Kill Ring

Emacs use **kill** commands to delete text, for example **kill-region, kill-word**. However, 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 puts things you've copied or deleted.

What exactly goes into the kill ring? Only the text delete by **Del** and **C-d** (without numeric argument) does not go into kill ring. All the else will go into kill ring.

1.8 Pointer and Cursor

A **cursor** is on the top the char. A **pointer** is the position between the cursor and one char before the cursor.

1.9 Backup Files

A backup file is a copy of the old contents of a file you are editing. Emacs makes a backup file the first time you save a buffer into its visited file. Thus, normally, the backup file contains the contents of the file as it was before the current editing session. The contents of the backup file normally remain unchanged once it exists. Thus the backup file keeps a copy of the original file. During the editing, you may save the buffer several times and the content of the original file will change accordingly.

The name of the backup file is the same as the name of the file you're editing, with a tilde (~) added. For example, if you are editing the file `text`, the backup file is `text~`

1.10 Auto Save Files

Emacs will auto save you files from time to time into `auto-save files`. The name of an auto-save file is the same as the name of the file you are editing, with a sharp (#) added to the beginning and the end. For example, if you are editing the file `text`, its auto-save file is `#text#`.

The save files are used save the content of current buffer if you do not save the buffer and the Emacs or system crashed.

Chapter 2

Start With Emacs

2.1 Start Emacs

There are two ways to start Emacs:

- Click the Emacs icon  .
- Use the command `emacs` in your terminal.

```
# To get emacs usage.
emacs --help

# To simply start the Emacs
emacs
```

2.2 Exit Emacs

In Emacs, use **C-x C-c** (x: execute; c: clear) to exit.

2.3 Working With Files

Binding	Meaning
C-x C-f (f: file)	Open or create a file.
C-x C-v	Read an alternate file, replacing the one read with **C-x C-f**.
C-x i (i: insert)	Insert file at cursor position.
C-x C-s (s: save)	Save buffer.
C-x C-w (w: write)	Save buffer as another file.

Tables 2.1: Working with files

Chapter 3

Editing

3.1 Moving Cursor

Group	Binding	Meaning
char	**C-f (f: forward)**	forward one char
	C-b (b: backward)	backward one char
	C-n (n: next line)	downward one char
	C-p (p: previous line)	upward one char
word	**M-f**	forward one word
	M-b	backward one word
line	**C-a**	beginning of line
	C-e	end of line
sentence	**M-a**	start of sentence
	M-e	end of sentence
paragraph	**M-{**	start of paragraph
	M-}	end of paragraph
screen	**C-v**	downward one screen
	M-v	upward one screen
	C-l	scroll the window so that current line is in the middle of the window, type again, cursor on top, again, bottom
	C-M-v	scroll other window
buffer	**M-<**	start of buffer
	M->	end of buffer
	M-m	first non-whitespace character on this line
	F10	start key navigation of the menu bar

Tables 3.1: Moving Cursor

Keep in mind that: **Meta** bindings move larger than **Ctrl** bindings. Table 3.1 show commands for moving cursor.

3.2 Marking

To define a region using the keyboard, you use a secondary pointer called a **mark**. You set the mark, move the cursor to somewhere to define a region between the mark and the cursor.

Table 3.2 show commands for marking.

Binding	Meaning
C-Space or C-@	set the mark
C-x C-x	exchange point and mark
M-h	mark paragraph
C-x h	mark buffer

Tables 3.2: Marking

C-Space sets mark and highlights the **region**. The region is still there even if you can not see it. Because the region is define between mark and point. The mark is there even if you can not see it.

3.3 Deleting, Copying and Pasting

Table 3.3 show commands for deleting, copying and pasting.

Group	Binding	Meaning
delete	**Del**	delete backward char
	C-d (d: delete)	delete current char
	M-Del	delete between start of word and one char before cursor
	M-d	delete between cursor and end of word
	C-k (k: kill)	delete between cursor and end of line
	M-k	delete between cursor and end of sentence
	M-- M-k	delete between start of sentence and one char before cursor
	C-w	delete marked region
copy	**M-w**	copy marked region
past	**C-y (y: yank)**	past the most recently deleted or copied

Tables 3.3: Deleting, Copying and Pasting

After the **C-y**, you can immediately use **M-y** several times to navigate the kill ring.

3.4 Transposition and Capitalization

Transposition and capitalization can be a edit trick and save you time. Here are some commands shown in Table 3.4 for achieving that.

Binding	Meaning
C-t	interchange characters around point, moving forward one character
M-t	Interchange words around point, leaving point at end of them
C-x C-t	exchange current line and previous line, leaving point after both
M-c (c: capital)	capitalize from point to the end of word, moving over
M-u (u: upper)	convert to upper case from point to end of word, moving over
M-l (l: lower)	convert to lower case from point to end of word, moving over

Tables 3.4: Transposition and Capitalization

3.5 Rectangle Editing

You can select a region and do region editing. Refer to Table 3.5 for this kind of editing.

Binding	Meaning
C-x r k (r: rectangle; k: kill)	delete a rectangle and store it
C-x r c (c: clear)	using spaces, blank out the rectangle and do not store it
C-x r o	insert a blank rectangle, shift text right
C-x r y	insert the last rectangle killed
C-x r r β	copy rectangle to register β (β is any character)
C-x r i β	insert rectangle from register β
C-x r t `string` **Enter**	change contents of rectangle to `string`

Tables 3.5: Rectangle Editing

Chapter 4

Search and Replace

4.1 Different Kinds of Searches

Name	Meaning
simple search	give a string and find the next occurrence
incremental search	start to search as soon as you type the first character and continues to search as you type more characters
word search	like a simple search, but search only for full words and phrases
regular expression search	search a pattern
incremental regular expression search	combination of incremental search and regular search
simple search and replace	replace all occurrences
query replace	conditional replace a string
query replace word	like query replace, but replace only for full words and phrase
regular expression replace	use pattern to find string and replace

Tables 4.1: Search and replace

By default, searches are case-insensitive. One exception: if you type any uppercase letters, Emacs makes the whole search string case-sensitive.

4.2 Bindings

Group	Binding	Meaning
search	**C-s**	incremental search
	C-M-s or C-u C-s	incremental regular expression search
replace	**M-%**	query replace
	C-u M-%	query replace word
	C-M-%	regular expression replace

Tables 4.2: Search and replace

> Remember:
> Type **C-h k C-s** to get the help information to learn how to use incremental search well.

Here are some useful bindings from the help page:

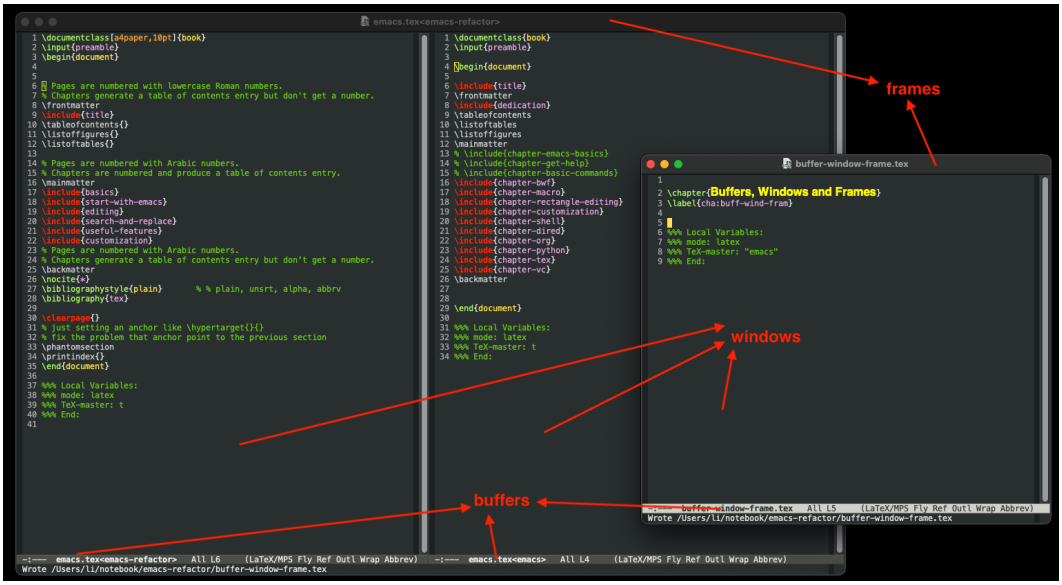
Binding	Meaning
C-j	match end of line, need directly follow **C-s**
C-w	yank next word or character in buffer
C-y	yank last killed text
M-y	yank previous killed text
C-s	next matched string
C-r	previous matched string
RET	exit, leaving point at location found
M-c (c: case)	toggle case-sensitivity
M-e (e: edit)	edit the search sting in the minibuffer
M-s M-<	go to the first match
M-s M->	got to the last match
M-s w	toggle word mode
M-n	search for the next item in the search ring
M-p	search for the previous item in the search ring
C-M-i	complete the search string using the search ring

Tables 4.3: Commands in incremental search

Chapter 5

Buffers, Windows and Frames

5.1 Concepts



Figures 5.1: Buffers, windows and frames

Buffers are independent of windows and frames. A Buffer may contains a file or be Emacs-generated. The name of the buffer that containing a file is the same name with the file. The name of the buffer that are Emacs-generated has the format `*buffer name*`. For example, `*Help*, *scratch*, *Messages*` as so on.

5.2 Buffer Commands

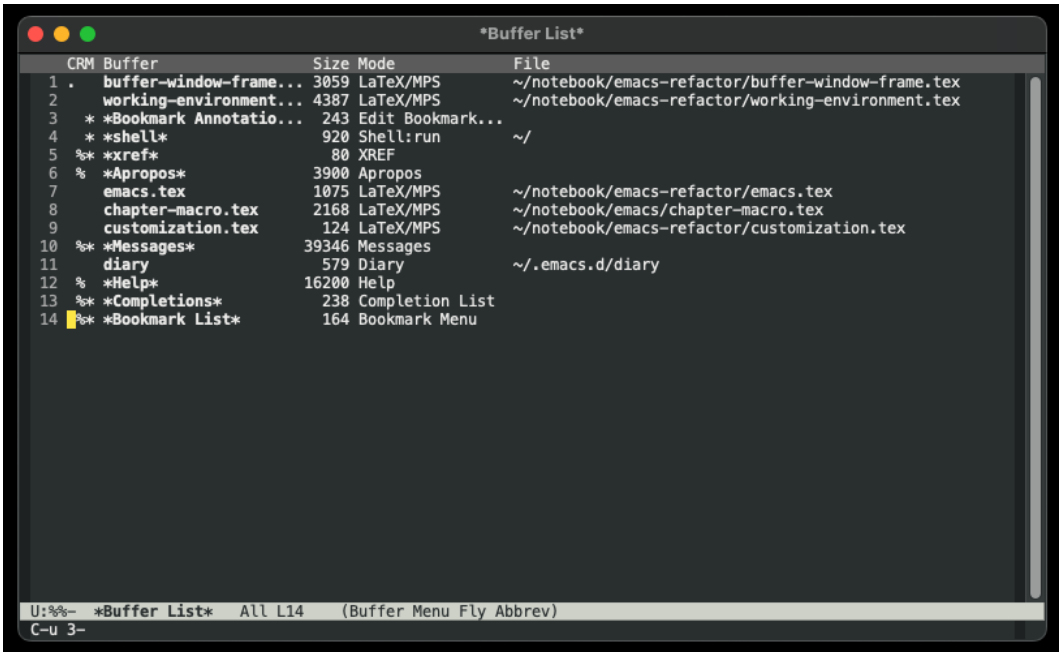
Binding	Meaning
C-x b	move to most recently buffer or the specified buffer
C-x k	delete buffer
C-x C-s	save buffer
C-x s	save all buffers
C-x C-q	toggle read only mode
C-x C-b	list all buffers

Tables 5.1: Buffer Commands

5.2.1 Buffer List

C-x C-b creates a new *Buffer List* window on the screen. It is shown in the following Figure 5.2. The CRM column has the following available values:

- . : displayed
- * : modified
- % : read only
- D : marked for deletion
- > : marked for display
- S : marked for saving



Figures 5.2: Buffer list

Keystroke	Meaning
C-n or n	down one line
C-p or p	up one line
m	mark buffer to be displayed
d	mark buffer for deletion
s	mark buffer for save
u	unmark buffer
U	remove all marks from all lines
Del	unmark the previous buffer, if there is no mark, move up one line
x	execute other one-letter commands on all marked buffers
v	display buffers marked with `m`
T	toggle whether the menu displays only file buffers
%	toggle read-only status of buffer
RET	select current line's buffer in place of the buffer menu
V	select current line's buffer, in View mode
1	display buffer in a full screen
2	display this buffer and the next one in horizontal windows
f	replace buffer list with this buffer
o	replace other window with this buffer
C-o	make another window display that buffer
t	visit the tags table in the buffer on this line.
M-s a C-s	incremental search in the marked buffers
M-s a C-M-s	isearch for regexp in the marked buffers
q	quit buffer list

Tables 5.2: Buffer list commands

5.3 Window Commands

Binding	Meaning
C-x o	switch to other window
C-x 0	delete current window
C-x 1	delete other windows
C-x 2	split window down
C-x 3	split window right
C-x ^	enlarge window vertically
C-x }	enlarge window horizontally
C-x {	shrink window horizontally
C-x -	shrink window if larger than buffer
C-x +	balance window

Tables 5.3: Window Commands

A number of the "other window" commands are just the ordinary command with a 4 inserted in it. For example, to find a file in another window, type **C-x 4 f**. To select a different buffer in another window, type **C-x 4 b**. These commands save you steps: you do not need move to the other window, give a command and move back.

5.3.1 Package "ace-window"

Using **C-x o** to switch between windows if there are only two windows. It quickly loses its value when there are more windows. "ace-window" is used to solve this problem.

```
;; Rebind M-o to ace-window
(global-set-key (kbd "M-o") 'ace-window)

;; This is the list of initial characters used in window labels.
;; The characters are used after ace-window command to select the window.
(setq aw-keys '(?a ?s ?d ?f ?g ?h ?j ?k ?l))
```

You can swap current window and a specified by calling **M-o** with a prefix argument **C-u**. You can delete the selected window by calling **ace-window** with a double prefix argument **C-u C-u**.

5.4 Frame Commands

Binding	Meaning
C-x 5 o	switch to other frame
C-x 5 0	delete current frame
C-x 5 1	delete other frame
C-x 5 2	create a new frame on the current buffer

Tables 5.4: Frame Commands

A number of the "other frame" commands are just the ordinary command with a 5 inserted in it. For example, to find a file in another frame, type **C-x 5 f**. To select a different buffer in another frame, type **C-x 5 b**. These commands save you steps: you do not need move to the frame, give a command and move back.

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 Macro Commands

Group	Binding	Meaning
define a macro	**C-x (**	start macro definition
	C-x)	end macro definition
execute macro	**C-x e**	call the last defined macro
	C-x C-k r	apply last macro to all lines in region
edit macro	**C-x C-k e**	edit macro
macro ring	**C-x C-k C-d**	delete current macro from macro ring
	C-x C-k C-t	swap first two elements in macro ring
	C-x C-k C-p	move to previous macro in macro ring
	C-x C-k C-n	move to next macro in macro ring
bind macro	**C-x C-k b**	bind macro
name macro	**C-x C-k n**	name macro

Tables 6.1: Macro Commands

6.2 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`.

6.3 Pausing a Macro for Keyboard Input

When you're defining a macro, type **C-u C-x q** at the point where you want the recursive edit to occur. Emacs enters a recursive edit. You can tell you're in a recursive edit because square brackets appear on the mode line. Nothing you type during the recursive edit becomes a part of the macro. You can type whatever you want to and then press **C-M-c** to exit the recursive edit.

6.4 Adding a Query to a Macro

When you're defining a macro, type **C-x q** at the point where you want to add a query. Nothing happens immediately; go on defining the macro as you normally would. When you execute the macro and it gets to the point in the macro where you typed **C-x q**, Emacs prints a query in the minibuffer:

```
Proceed with macro? (y, n, RET, C-l, C-r)
```

Here's the meaning of the options:

- **y** means to continue and go on to the next repetition, if any.
- **n** means to stop executing the macro but go on to the next repetition, if any.
- **Enter** means to stop executing the macro and cancel any repetitions.
- **C-r** C-r starts a recursive edit. To exit a recursive edit, press **C-M-c**.
- **C-l** puts the line the cursor in on in the middle of the screen.

Chapter 7

Emacs as Working Environment

7.1 Spell Checking

Emacs provide Ispell interface. We say "interfaces" because Emacs does not include the executable.

On MacOSX, There are two popular software manager **MacPorts** and **Homebrew**. You can use any of it to install Ispell. After installation, use command `which ispell` in terminal to locate the Ispell executable file. In Emacs, type **C-h a ispell** to get all functions containing the keyword ispell. You can find the function `ispell-check-version`. Type **M-x ispell-check-version** to check whether Ispell is working correctly. It will show you where should the Ispell executable should be located. Link the Ispell executable to where Emacs expects Ispell to be:

```
1 ln -s /opt/local/bin/ispell /usr/local/bin/ispell
```

Binding	Command	Meaning
	`ispell-buffer`	
	`ispell-region`	
M-$	`ispell-word`	
	`ispell-complete-word`	

Tables 7.1: Ispell Commands

You can use **C-h f ispell-help** for the options available when a misspelling is encountered. The options are the following:

Option	Meaning
DIGIT	replace the word with a digit offered in the *Choices* buffer
SPC	accept word this time
i	accept word and insert into private dictionary
a	accept word for this session
A	accept word and place in 'buffer-local dictionary'
r	replace word with typed-in value. Rechecked
R	replace word with typed-in value. Query-replaced in buffer. Rechecked
x	exit spelling buffer. Move cursor to original point
X	exit spelling buffer. Leaves cursor at the current point, and permits the aborted check to be completed later
q	quit spelling session
l	look up typed-in replacement in alternate dictionary
u	like 'i', but the word is lower-cased first
m	place typed-in value in personal dictionary, then recheck current word
C-r	recursive edit

Tables 7.2: Ispell options

7.2 Flyspell

Flyspell highlights misspelled words as you type. There are two mode related with Flyspell: **flyspell-mode** and **flyspell-prog-mode**. The latter mode is designed for programmers. In this mode Emacs highlights misspellings only in comments or strings. To check existing text, you run **M-x flyspell-buffer Enter**.

Flyspell highlights misspelled words in red. Words that are repeatedly misspelled are highlighted in yellow.

Here are some useful commands in **flyspell-mode**:

Binding	Meaning
M-$	correct words (using Ispell)
C-M-i	automatically correct word
C-;	automatically correct the last misspelled word

Tables 7.3: Flyspell commands

You can type **C-M-i** or **C-;** again to change to another correct word if the corrected word is not what you want.

7.3 Bookmarks

It saves any new bookmarks in this file automatically when you exit Emacs. Bookmark points to a position in a file and not to a piece of text.

7.3.1 Bookmark Commands

Binding	Meaning
C-x r m	set a bookmark
C-x r b	move to a bookmark
C-x r l	`*Bookmark List*` buffer appears

Tables 7.4: Bookmark Command

7.3.2 Bookmark List

Binding	Meaning
RET, or j	open bookmark
o	open the bookmark in another window and move the cursor to that window
C-o	open the bookmark in another window and keep the cursor in current window
r	rename bookmark
s	save all bookmark
m	mark bookmarks to be displayed in multiple windows
d	mark bookmark for deletion
u	remove mark
x	delete bookmarks marked for deletion
v	display marked bookmarks or the one the cursor is on if none are marked
e	edit (or create) annotation for the current bookmark
a	display annotation for current bookmark
A	display all annotations
q	exit bookmark list

Tables 7.5: Bookmark list commands

7.4 Shell

7.4.1 One Command at a Time

Binding	Meaning
M-!	execute command
M-|	execute command with region as input
M-&	execute command asynchronously in background

Tables 7.6: One command at a time

With **C-u** prefix, the command output is insert to current buffer.

7.4.2 Shell Mode

To start a shell buffer, type **M-x shell Enter**. This creates a buffer named `*shell*`.

How does Emacs know which shell to start? First, it looks at the variable `shell-file-name`. Then it looks for a Unix environment variable named `ESHELL`. Finally it looks for an environment variable named `SHELL`. If you want to run another particular shell (for example, the zed shell) when you're in Emacs, you can add the following command to your `.emacs` file:

```
1 (setq shell-file-name "/bin/zsh")
```

When Emacs starts an interactive shell, it runs an additional initialization file after your shell's normal startup files. The name of this file is `.emacs_shell-name`, where `shell-name` is the name of the shell you want to use in Emacs. It must be located in your home directory. For example, if you use zsh, you can add Emacs-only startup commands by placing them in the file `.emacs_zsh`

Group	Binding	Meaning
terminal	**C-c C-c**	like **C-c** in terminal
	C-c C-u	line **C-u** in terminal
	M-r	like **C-r** in terminal
	M-p	like **C-p** in terminal
	M-n	like **C-n** in terminal
	C-c SPC	like \ in terminal
delete	**C-c C-o**	delete output from last command
	C-c M-o	clear the shell buffer
window	**C-c C-r**	move first line of output to top of window
	C-c C-e	move last line of output to bottom of window
history	**C-c C-p**	move to previous command
	C-c C-n	move to next command
	C-c C-l	list the commands you have typed
argument	**C-c .**	insert previous command argument
move	**C-c C-f**	`shell-forward-command`
	C-c C-b	`shell-backward-command`
save	**C-c C-s**	write the privous output into a file (overwrite)

Tables 7.7: Shell mode commands

To create multiple shell buffer, you can use **C-u M-x shell**.

7.4.3 Eshell, Shell and Term

Here's the differences:

- **M-x shell** starts the standard emacs interface to Operating System's command line interface.
- **M-x term** starts the terminal emulator in emacs. It behaves like a dedicated terminal app, such as xterm, gnome-terminal, puTTY. It is compatible to more shell apps than emacs shell interface, but standard emacs keys such as moving cursor doesn't work here.
- **M-x eshell** starts eshell which is a shell written entirely in emacs lisp. Note: it is not a bash emulator. Eshell is a shell by itself, but similar to bash or other shells.

Which should you use? It depends on your preference and needs. The following is some general guide.

- shell is the most popular. It is good for general use of classic/standard unix shell commands.
- term are good if you want to run stuff like `ssh`, or other command line interactive interface (such as `python`), or text based GUI app such as `vim`.
- Eshell is super fast on startup. If you are a emacs lisp programer, you might prefer eshell because direct access to emacs lisp and better integration with emacs.

7.5 Directory Editor

There are several ways to start directory editing. If you're not in Emacs, invoke Emacs with a directory name as an argument. For example, if you want to edit the directory `notebook`, type the following

```
emacs notebook
```

If you are in Emacs, you can use **C-x C-f DIRECTORY** or **C-x d DIRECTORY** to edit the directory.

Group	Binding	Meaning
mark	**m**	mark current file
	*** ***	mark executable files
	*** @**	mark symlinks
	*** /**	mark directories
	% m	mark files matching `REGEXP`
	% g	mark files containing `REGEXP`
	% d	flag for deletion files that match regular expression
	d	mark file for deletion
	#	mark autosave files for deletion
	~	mark backup files for deletion, **C-u** to prefix remove the flag
	&	mark garbage files for deletion
	u	unmark current file
	U	unmark all files
	t	toggle mark
	*** c**	change marks on specified files
visit	**v**	view file read only, **q** to quit
	o	find file in another window; move there
	C-o	find file in another window; don't move there
	RET	visit file
operation	**O**	change ownership of file
	D	delete a file immediately
	R	rename
	C	copy marked files
	G	change group permissions
	L	load the marked Emacs Lisp files
	Z	compress or uncompress
	M	use **chmod** on current file
	S	create a symbolic link to this file
	s	sort the Dired display by date or by filename (toggle)
	w	copy filename into the kill ring
	y	display information on the type of the file using the **file** command
	!	run shell command

Continued on next page

Group	Binding	Meaning
	+	create a directory
	Q	replace matches in all marked files
	A	do a regular expression search on marked files
	x	delete the files flagged for deletion
navigation	**n**	next line
	p	previous line
	^	up directory
	>	next directory line
	<	previous directory line
	i	insert this subdirectory into the same dired buffer
	$	hide or show the current directory or subdirectory
	M-$	hide all subdirectories, leaving only their names; repeat command to show
	C-M-n	move to next subdirectory (if you've inserted subdirectories using **i**)
	C-M-p	move to previous subdirectory (if you've inserted subdirectories using **i**)
	M-}	next marked file
	M-{	previous marked file

Tables 7.8: Dired commands

Here's one trick. In Dired mode, you can use **C-x C-q** to change the readonly status. After is becomes editable, you can edit the dired buffer directly. When you finish the edit, type **C-c C-c** to make the change work.

7.6 Calendar

To display the calendar, type **M-x calendar**. Emacs displays a calendar window with three months: last month, this month, and next month. Here's my simple configuration in `.emacs` file:

```
1 ;; set the first day of a week to Monday
2 (setq calendar-week-start-day 1)
3 ;; start calendar at start emacs
4 (calendar)
```

Group	Binding	Meaning
relative move	**.**	today
	C-f	next day
	C-b	previous day
	C-n	forward by week
	C-p	backward by week
	M-}	forward by month
	M-{	backward by month
	C-x [	forward by year
	C-x]	backward by year
absolute move	**C-a**	beginning of the week
	C-e	end of the week
	M-a	beginning of the month
	M-e	end of the month
	M-<	beginning of the year
	M->	end of the year
	g d	go to the specified day
scroll	**C-x >**	scroll forward by 1 month
	C-x <	scroll backward by 1 month
	C-v	scroll forward by 3 months
	M-v	scroll backward by 3 months
holiday	**a**	show holidays for the current calendar window
	h	show whether today or the specified day is a holiday
	x	highlight holidays
	u	remove the highlights
add entry	**i d**	add day entry
	i w	add weekly entry
	i m	add month entry
	i y	add annual entry
	i a	add annual entry (the year is included for reference)
	i c	add cyclic entry
	i b	add block entry
display	**m**	highlight entries
	d	display entry for the current date
	s	display all entries

Tables 7.9: Calendar Commands

Chapter 8

Useful Features

8.1 Get Help

Emacs has extensive help. You should always check the help information to learn how to use the command, how to configure the Emacs, how to search for what you want and so on.

The whole help entry is **C-h C-h**. You should read the help buffer carefully to find what the next help information you want to get.

For example, if you want to replace some string but forget the key binding and the function, you can use **C-h a replace Enter**. You will get all the functions containing the keyword **replace**.

8.2 Auto Fill

8.2.1 Dynamic Expanding

M-/ runs the command `dabbrev-expand`. It expands previous word "dynamically". Expands to the most recent, preceding word for which this is a prefix. If no suitable preceding word is found, words following point are considered. It also search other buffers.

8.2.2 Auto Correct Word

If flyspell-mode is enabled, **C-M-i** runs the command `flyspell-auto-correct-word`. This command proposes various successive corrections for the current word. If invoked repeatedly on the same position, it cycles through the possible corrections of the current word.

Otherwise, **C-M-i** runs corresponding command to do completion.

8.2.3 Auto Fill Commands

If you are in minibuffer, you can use **Tab** to auto-fill commands and filenames.

For example, type **M-x open Tab**. If there are only one command matching the "open" prefix, it will be auto-filled. If there are more commands matching the prefix, type **Tab** again to get all

candidates. In AUCTeX mode, you use **C-c C-m** to start a macro. In the minibuffer, you can use **Tab** to auto-fill LaTeX command.

If you type **C-x C-f** to open a file. In the minibuffer, you can use **Tab** to auto-fill the filename or use **Tab Tab** to get all candidates.

8.3 Canceling Commands

When you want to cancel any command that's in progress, press **C-g**. The word **Quit** appears in the command area. This command is helpful when you are stuck in the minibuffer and didn't really mean to go there. Depending on what you were doing, you may have to press **C-g** a few times.

8.4 Undoing Changes

What happens if you make a mistake while you're editing? You can undo your changes by **undo** function. Type **C-h k undo Enter** and choose your favorite key binding.

What if you'd like to redo a command after you type undo? There is no formal redo command, but you can use undo in the following way. Just move the cursor in any direction, and use undo again. Emacs redoes the last command you undid. You can repeat it to redo previous undos.

8.5 History Commands

Emacs will record the commands you have used before, so you can reuse them to avoiding typing them again to save you time.

C-x Esc Esc will let you edit then re-evaluate the last complex command. A **command** is one that used the minibuffer. The command is placed in the minibuffer as a Lisp form for editing. The result is executed, repeating the command as changed. In the minibuffer, you can use **M-n** and **M-p** to navigate fore ward and backward the history.

C-x z repeats the most recently executed command.

In LaTeX mode, when you use **C-c C-e** to insert an environment, In the minibuffer, you can use **M-n** and **M-p** to navigate fore ward and backward the history.

8.6 C-u

C-u runs the command `universal-argument`. It has the following situations:

- ♠ following **digits**, it will repeat the following command **digits** times
- ♠ following **- or - digits**, some commands support negative digits and it will repeat the following command **1 or digits** times in the opposite direction
- ♠ without digits or minus sign, it provides 4 as argument
- ♠ repeating **C-u** without digits or minus sign many times, it multiplies the argument by 4 each time
- ♠ for some commands, just **C-u** by itself serves as a flag that change the command function.

8.7 Saving Positions in Registers

Typing **C-x r SPC (point-to-register)**, followed by a character **r**, saves both the position of point and the current buffer in register **r**. The register retains this information until you store something else in it.

The command **C-x r j r** switches to the buffer recorded in register **r**, pushes a mark, and moves point to the recorded position. (The mark is not pushed if point was already at the recorded position, or in successive calls to the command.) The contents of the register are not changed, so you can jump to the saved position any number of times.

8.8 rgrep

M-x rgrep can recursively search the content in the directory and files you specified.

Chapter 9

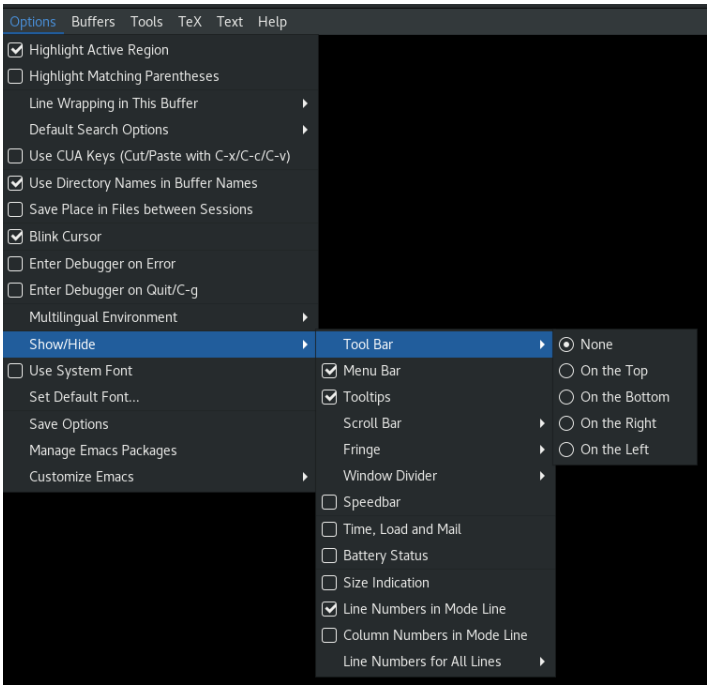
Customization

You you can customize Emacs in tree ways:

- using **Custom**, the interactive interface
- using **Options** menu
- adding lines of Lisp to you `.emacs` file

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`.

9.1 Using Options



Figures 9.1: Using Options to hide toolbar

Here the Figure 9.1 to show how to use Options to hide toolbar.

9.2 Using Custom

Emacs now ships with a quirky graphical-but-not interface that allows you to customize most aspects of Emacs without knowing the gory details. This feature, known as Custom, can be accessed by typing **M-x custom** or by clicking the tools icon on the toolbar.

9.3 Modifying the .emacs File Directly

Emacs actually looks for a variety of startup files. In order, they are:

- `.emacs.elc`: The byte-compiled Lisp version of your startup file. This is not editable, but can make startup quicker if you have a big, complex startup file.
- `.emacs.el`: The more formal name for your startup file. You can use Lisp commands to customize and initialize your entire Emacs environment.
- `.emacs`: The common name for the startup file.
- `.emacs.d/init.el`

As soon as Emacs finds one of these files, that's it; then it's on to the next step in startup.

Here are some useful functions:

```
;; This is the basic syntax.
(function-name arguments)
;; Example: Move forward one word.
(forward-word 1)

;; Set a value to a variable.
(setq variable-name variable-value)
;; Using setq-default has the advantage of setting the default value only.
;; Modes that choose to override this value may still do so.
(setq-default variable-name variable-value)
;; Example: set nil to indent-tabs-mode.
(setq-default indent-tabs-mode nil)

;; Add a hook to mode hook.
(add-hook 'hook-name 'hook-value)
;; Example: add font lock to emacs lisp mode
(add-hook 'emacs-lisp-mode-hook 'turn-on-font-lock)

;; Define key binding.
(define-key keymap "keystroke" 'command-name)
(global-set-key "keystroke" 'command-name)
(local-set-key "keystroke" 'command-name)
;; Example: bind goto-line to C-x l
(global-set-key "\C-xl" 'goto-line)
(define-key global-map "\C-xl" 'goto-line)
(define-key ctl-x-map "l" 'goto-line)

;; Unset key binding.
(global-unset-key "keystroke")
(define-key keymap "keystroke" nil)
```

```
31 ;; Example: unset C-x l
32 (global-unset-key "C-xl")
33 (define-key clt-x-map "l" nil)
```

When define key binding,the three commands have the same effect but aren't really any more efficient or better. You can just use `global-set-key` and use `define-key` when setting the global key is not appropriate, such as when adding a mode-specific keystroke. Here are some social characters for define key bindings shown in Table

\Special character	Meaning
C-	Control
e	Esc
M-	Meta
n	Newline
r	Enter
t	Tab
S-	shift
H-	hyper
s-	super
A-	alt

Tables 9.1: Special characters

Chapter 10

LaTeX

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 **AUCTEX** can help you write your papers efficiently.

10.1 Installation

Here is the link https://www.gnu.org/software/auctex/download.html to install AUCTEX.

10.2 Configuration

Here is my code for configuring Emacs to use AUCTEX:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; AUCTEX
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Automatically save style information when saving the buffer.
;; AUCTEX will create the auto directory automatically.
(setq TeX-auto-save t)

;; Parse file after loading it if no style hook is found for it.
(setq TeX-parse-self t)

;; If you often use \include or \input, you should make AUCTEX aware of the
    multifile document structure.
(setq-default TeX-master nil)           ; Query for master file.

;; Automatically insert '$...$' in plain TEX files, and '\(...\)' in LaTeX
    files by pressing $
(add-hook 'plain-TeX-mode-hook
          (lambda () (set (make-local-variable 'TeX-electric-math)
                          (cons "$" "$"))))
(add-hook 'LaTeX-mode-hook
          (lambda () (set (make-local-variable 'TeX-electric-math)
                          (cons "\\(" "\\)"))))
```

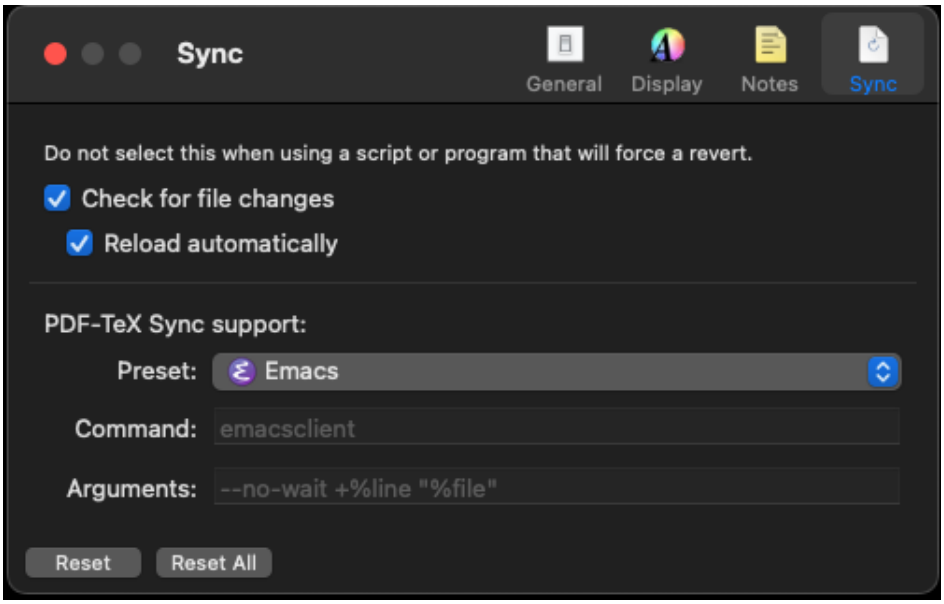
```
21
22 ;; If this option is on, just typing (, { or [ immediately
23 ;; adds the corresponding right brace ')', '}' or ']'.  The
24 ;; point is left after the opening brace. If there is an
25 ;; active region, braces are put around it.
26 (setq LaTeX-electric-left-right-brace t)
27
28 ;; Get a full featuered LaTeX-section command.
29 (setq LaTeX-section-hook
30       '(LaTeX-section-heading
31         LaTeX-section-title
32         LaTeX-section-toc
33         LaTeX-section-section
34         LaTeX-section-label))
35
36 ;; Enable LaTeX Math mode by default.
37 ;; Easy insertion of LaTeX math symbols.  If you give a
38 ;; prefix argument, the symbols will be surrounded by dollar signs.
39 (add-hook 'LaTeX-mode-hook #'LaTeX-math-mode)
40
41 ;; Define the function used when you type Enter.
42 (setq TeX-newline-function 'reindent-then-newline-and-indent)
43
44 ;; When point is on one of the characters, it'll be unprettified
      automatically,
45 ;; meaning you see the verbatim text again.
46 (setq prettify-symbols-unprettify-at-point 'right-edge)
47
48 ;; Enable prettification in AUCTEX.
49 (add-hook 'LaTeX-mode-hook 'prettify-symbols-mode)
50
51 ;; Add XeLaTeX program to run on the master or region file.
52 (eval-after-load "tex"
53   '(add-to-list 'TeX-command-list
54                 '("XeLaTeX" "xelatex %s" TeX-run-command t t :help "Run
                       xelatex")
55                 t))
56
57 ;; Set the default command to run in LaTeX mode.
58  (add-hook 'LaTeX-mode-hook (lambda ()
59                               (setq TeX-command-default "XeLaTeX")))
60
61 ;; ;; Add user defined viewer Skim.
62 (setq TeX-view-program-list '(("Skim" "/Applications/Skim.app/Contents/
      SharedSupport/displayline -g -b %n %o %b")))
63
64 ;; Select the viewer to start view pdf output.
65 (setq TeX-view-program-selection
66       '((output-dvi "open")
67         (output-pdf "Skim")
68         (output-html "open")))
```

```
69
70  ;; Add TeX-source-correlate-mode to LaTeX mode to support forward
71  ;; and inverse search.
72  (add-hook 'LaTeX-mode-hook 'TeX-source-correlate-mode)
73
74  ;; Whether Emacs retains the focus when viewing PDF files with Evince.
75  ;; If this option is set to non-nil, Emacs will retain the focus.
76  ;;(setq TeX-view-evince-keep-focus t)
77
78  ;; If TeX-source-correlate-mode is active and a viewer is invoked,
79  ;; the default behavior is to ask if a server process should be started.
80  ;; Set this variable to t if the question should be inhibited and the
81  ;; server should always be started.
82  (setq TeX-source-correlate-start-server t)
83
84  ;; Make special mouse event do forward search at the clicked position.
85  (eval-after-load "tex"
86    '(define-key TeX-source-correlate-map [C-down-mouse-1]
87       #'TeX-view-mouse))
88
89  ;; A function that will be called after performing an inverse search
90  ;; from pdf in order to raise the current Emacs frame.
91  (setq TeX-raise-frame-function #'x-focus-frame)
92
93  ;; When this boolean variable is non-nil, the error overview will be
94  ;; automatically opened after running TEX if there are errors or warnings to
       show.
95  (setq TeX-error-overview-open-after-TeX-run t)
96
97  ;; Get in-buffer notation for syntax error.
98  (add-hook 'LaTeX-mode-hook #'flymake-mode)
99
100 ;; The output files will be placed in `build` directory.
101 ;; This can improve the readability of the tex files.
102 ;;(setq-default TeX-output-dir "build")
103
104 ;; Add flyspell mode to latex mode.
105 (add-hook 'LaTeX-mode-hook 'flyspell-mode)
106
107 ;; Add visual-line-mode to latex mode.
108 (add-hook 'LaTeX-mode-hook 'visual-line-mode)
109
110 ;; Minor mode with distinct support for \label, \ref and \cite in LaTeX.
111 (add-hook 'LaTeX-mode-hook 'turn-on-reftex)
112 (setq reftex-plug-into-AUCTeX t)
113
114 ;; If non-nil, insert braces after typing '^' and '_' in math mode.
115 (setq TeX-electric-sub-and-superscript t)
116
117 ;; Support folding sections in LaTeX.
118 (add-hook 'LaTeX-mode-hook 'outline-minor-mode)
```

10.2.1 Skim

You should configure Skim to load automatically and let it know Emacs to locate the cursor.

The configuration is shown in Figure 10.1



Figures 10.1: Skim configuration

To display the TeX source line corresponding to a point in the PDF document, hold down the Shift and Apple key and click on a point in the PDF document.

10.3 AUCTEX Commands

Group	Binding	Meaning
insert	**C-c C-s**	insert section macros
	C-c C-e	insert environment, with **C-u** prefix will change the current environment
	C-c C-m	insert LaTeX macro
comment	**M-;**	insert comment, if there is a region, comment or comment the region
	C-c ;	comment or uncomment the current region
	C-c %	command or uncomment current paragraph
run	**C-c C-c**	run command on the current document
	C-c C-a	compile the document until it is ready and then run the viewer
	C-c C-r	run on region(also use preamble)
	C-c C-b	run only on current buffer(`\include` or `\input`), using the preamble from the master file
	C-c C-k	kill running process

Continued on next page

Group	Binding	Meaning
	C-c C-v	start a viewer without confirmation
debug	**C-c `**	find the next error in the TeX output buffer
	C-c C-l	display output so that most recent output can be seen
	M-g p	find the previous error in the TeX output buffer
move	**C-M-a**	move point to the beginning of current environment
	C-M-e	move point to the end of current environment
	**C-c **	go the 'master' file in the document associated with the current buffer
reftex	**C-c &**	view cross reference of macro at point
	C-c (	insert a unique label
	C-c)	make a LaTeX reference
	C-c -	display the TOC window and highlight line corresponding to current position
	C-c /	put selection or the word near point into the default index macro
	C-c <	query for an index macro and insert it along with its arguments
	C-c >	display a buffer with an index compiled from the current document
	C-c [	make a citation using BibTeX database files
	**C-c **	add current selection or word at point to the phrases buffer
	C-c |	switch to the phrases buffer, initialize if empty
outline	**C-c @ C-t**	hide all of buffer except headings
	C-c @ C-a	show all of the text in the buffer
	C-c @ C-q	hide everything but top levels headers
	C-c @ TAB	show all direct subheadings of this heading
	C-c @ C-k	show all subheadings, but not bodies
	C-c @ C-p	go to previous visible heading
	C-c @ C-n	go to next visible heading
narrow	**C-x n e**	make text outside environment invisible
	C-x n n	make text outside region invisible
	C-x n w	un-narrow
toggle	**C-c C-t C-p**	toggle between DVI and PDF output
	C-c C-t C-i	toggle interactive mode
	C-c C-t C-s	toggle SyncTEX(or source specials) support
	C-c C-t C-o	toggle useage of Omega/lambda
	C-c C-t C-w	toggle whther AUCTEX should stop at warnings as well as errors

Continued on next page

Group	Binding	Meaning
doc	**C-c ?**	get documentation about the packages installed on your system, using texdoc to find the manuals
mark	**C-c .** **C-c ***	mark current environment mark current section

Tables 10.1: Basic commands for editing TEX files

If you enable braces auto completion, using **C-q** before typing (, { or [will suppress it.

C-c c-s run `LaTeX-section arg` to insert a section. The argument `arg` determines the type of section to be inserted:

- If `arg` is nil or missing, use the current level.
- If `arg` is a list (selected by **C-u**), go downward one level.
- If `arg` is negative, go up that many levels.
- If `arg` is positive or zero, use absolute level:
  - ✠ + 0 : part
  - ✠ + 1 : chapter
  - ✠ + 2 : section
  - ✠ + 3 : subsection
  - ✠ + 4 : subsubsection
  - ✠ + 5 : paragraph
  - ✠ + 6 : subparagraph

TeX-error-overview Show an overview of the errors and warnings occurred in the last TEX run.

Running TEX or LaTEX will only find regular errors in the document, not examples of bad style. Furthermore, description of the errors may often be confusing. The utilities `lacheck` and `chktex` can be used to find style errors, such as forgetting to escape the space after an abbreviation or using '..' instead of '. . . ' and other similar problems. You start `lacheck` with **C-c C-c Check RET** and `chktex` with **C-c C-c ChkTeX RET**. The result will be a list of errors in the `'*compilation*'` buffer. You can go through the errors with **C-x `**, which will move point to the location of the next error.

Chapter 11

Version Control

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

Here we use Git as the version control system. In Emacs, we use Magit. Magit is an interface to the version control system Git, implemented as an Emacs package.

11.1 Installation

In you `.emacs` or `.emacs.el` file, add the following code.

```
;; Use package manager interface.
(require 'package)

;; Add melpa site to package archives.
;; This is used define where to fetch package.
(add-to-list 'package-archives '("melpa-stable" . "https://stable.melpa.org/
    packages/") t)

;; Load Emacs Lisp packages, and activate them.
(package-initialize)


;; Install Magit
(package-install 'magit)
```

11.2 Basic

11.2.1 Show Status Buffer

Type C-x g to display information about the current Git repository in a dedicated buffer, called the status buffer. If the current directory isn't located within a Git repository, then prompt for an existing repository or an arbitrary directory, depending on option "magit-repository-directories", and show the status of the selected repository instead.

- If that option specifies any existing repositories, then offer those for completion and show the status buffer for the selected one.
- Otherwise read an arbitrary directory using regular file-name completion. If the selected directory is the top-level of an existing working tree, then show the status buffer for that.
- Otherwise offer to initialize the selected directory as a new repository. After creating the repository show its status buffer.

Depending on what state your repository is in, this buffer may contain sections titled "Staged changes", "Unstaged changes", "Unmerged into origin/master", "Unpushed to origin/master", and many others.

Move between sections using **p** and **n**. Note that the bodies of some sections are hidden. Type **TAB** to expand or collapse the section at point. You can also use **C-tab** to cycle the visibility of the current section and its children.

> In status buffer, remember use **C-h m** to get help information about this mode.

11.2.2 Stage and Unstage

Move to a file section inside the section named "Unstaged changes" and type **s** to stage the changes you have made to that file. That file now appears under "Staged changes".

Magit can stage and unstage individual hunks, not just complete files. Move to the file you have just staged, expand it using **TAB**, move to one of the hunks using **n**, and unstage just that by typing **u**. Note how the staging (**s**) and unstaging (**u**) commands operate on the change at point. Many other commands behave the same way.

You can also un-/stage just part of a hunk. Inside the body of a hunk section (move there using **C-n**), set the mark using **C-SPC** and move down until some added and/or removed lines fall inside the region but not all of them. Again type **s** to stage.

It is also possible to un-/stage multiple files at once. Move to a file section, type **C-SPC**, move to the next file using **n**, and then **s** to stage both files.

11.2.3 Commit

If you want to commit your changes. Type **c**. This shows the available commit commands and arguments in a buffer at the bottom of the frame. Each command and argument is prefixed with the key that invokes/sets it. If you want to create a "normal" commit, just type **c** again.

Now two new buffers appear. One is for writing the commit message, the other shows a diff with the changes that you are about to commit. Write a message and then type **C-c C-c** to actually create the commit.

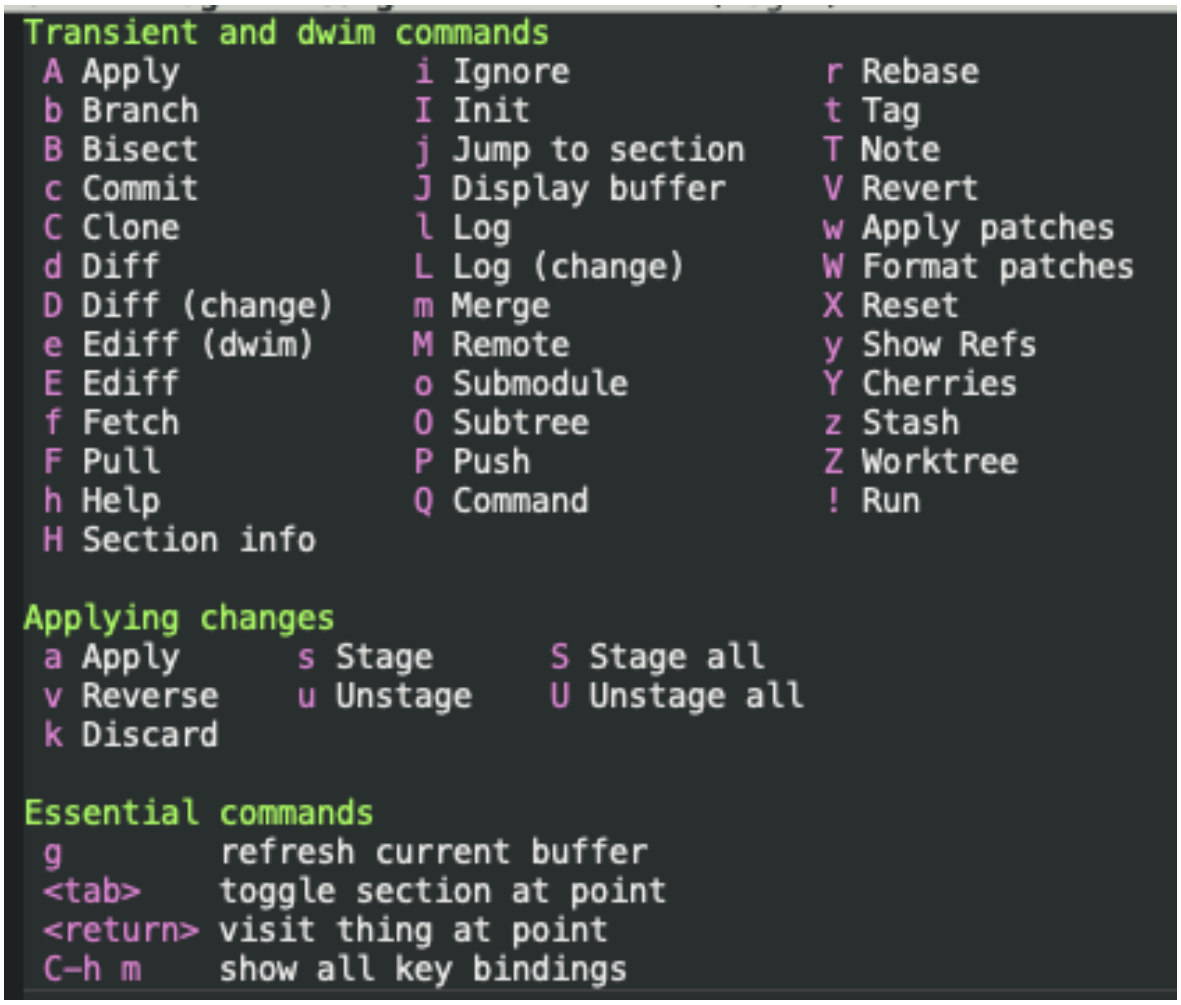
11.2.4 Push

You can push it by typing **p** to show all the available push commands and arguments and then **p** to push to a branch with the same name as the local branch onto the remote configured as the push-remote. If the push-remote is not configured yet, then you would first be prompted for the remote to push to.

11.2.5 Transient prefix commands

To show a menu that lists all menus, type **h**. (Such menus are also called "transient prefix commands" or just "transients".)

C-x M-g is the global binding of this menu. You can invoke this menu even not in status buffer. In file visiting buffers **C-c M-g** brings up a similar menu featuring commands that act on just the visited file.



Figures 11.1: Transient prefix commands

11.2.6 Summary

Binding	Meaning
C-x g	Show the status of the current Git repository in a buffer.
TAB	expand or hide the section at point.
C-TAB	Cycle the visibility of the current section and its children.
RET	visit the change or commit at point
n	Move to the beginning of the next visible section.
p	Move to the beginning of the current or the previous visible section.
s	stage
u	unstage
c	commit
p	push
h	invoke major commands
C-x M-g	invoke major commands
C-c M-g	invoke major commands that act on just the visited file

Tables 11.1: Basic magit commands

11.3 Interface Concepts

11.3.1 Modes and Buffers

Magit provides several major-modes. For each of these modes there usually exists only one buffer per repository. Separate modes and thus buffers exist for commits, diffs, logs, and some other things.

Besides these special purpose buffers, there also exists an overview buffer, called the status buffer. It's usually from this buffer that the user invokes Git commands, or creates or visits other buffers.

Binding	Meaning
q	Bury the current buffer. With a prefix argument, kill the buffer instead. With two prefix arguments, also kill all Magit buffers associated with this repository.
g	Refresh the current buffer if its major mode derives from "magit-mode", and refresh the corresponding status buffer.
G	Refreshes all Magit buffers belonging to the current repository and also reverts all unmodified buffers that visit files being tracked in the current repository.

Tables 11.2: Modes and buffers commands

11.3.2 Sections

Magit buffers are organized into nested sections, which can be collapsed and expanded, similar to how sections are handled in Org mode. Each section also has a type, and some sections also have a

value. For each section type there can also be a local keymap, shared by all sections of that type.

Taking advantage of the section value and type, many commands operate on the current section, or when the region is active and selects sections of the same type, all of the selected sections. Commands that only make sense for a particular section type are usually bound in section type keymaps.

Binding	Meaning
n	Move to the beginning of the next visible section.
p	Move to the beginning of the current or the previous visible section.
M-p	Move to the beginning of the previous sibling section. If there is no previous sibling section, then move to the parent section instead.
M-n	Move to the beginning of the next sibling section. If there is no next sibling section, then move to the parent section instead.
^	Move to the beginning of the parent of the current section.
TAB	Expand or hide the section at point.
C-TAB	Cycle the visibility of current section and its children.
magit-section-cycle-diffs	Cycle the visibility of diff-related sections in the current buffer.
magit-section-cycle-global	Cycle the visibility of all sections in the current buffer.
1 **2** **3** **4**	Show sections surrounding the current section up to level N.
M-1 **M-2** **M-3** **M-4**	Show all sections up to level N.
H	Shows information about the section at point in a separate buffer.

Tables 11.3: Sections commands

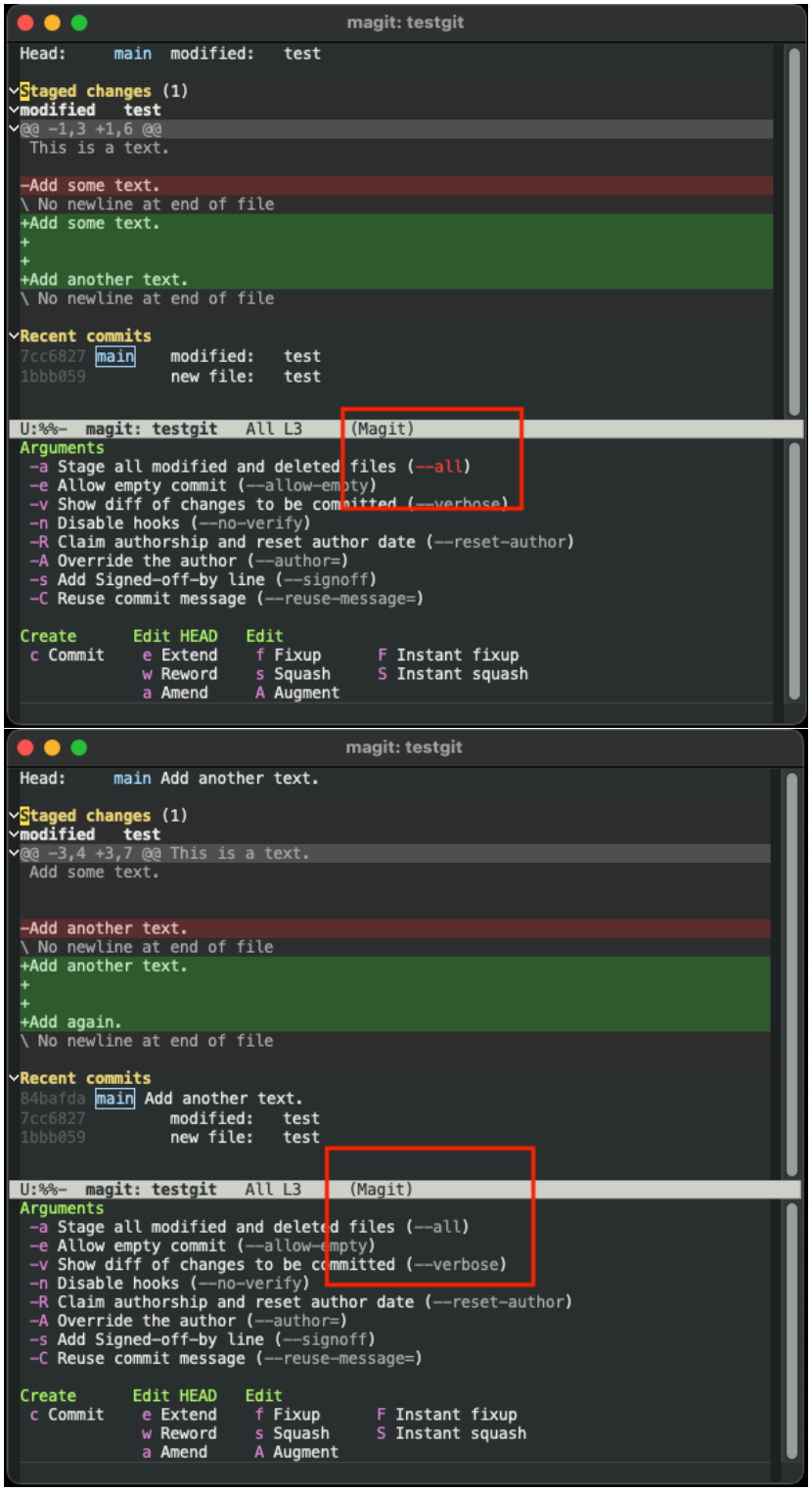
11.3.3 Transient Commands

Many Magit commands are implemented as transient commands. First the user invokes a prefix command, which causes its infix arguments and suffix commands to be displayed in the echo area. The user then optionally sets some infix arguments and finally invokes one of the suffix commands.

11.3.4 Transient Arguments and Buffer Variables

The infix arguments of many of Magit's transient prefix commands cease to have an effect once the git command that is called with those arguments has returned. Commands that create a commit are a good example for this. If the user changes the arguments, then that only affects the next invocation of a suffix command. If the same transient prefix command is later invoked again, then the arguments

are initially reset to the default value. It is possible to cycle through previously used sets of arguments using C-M-p and C-M-n.



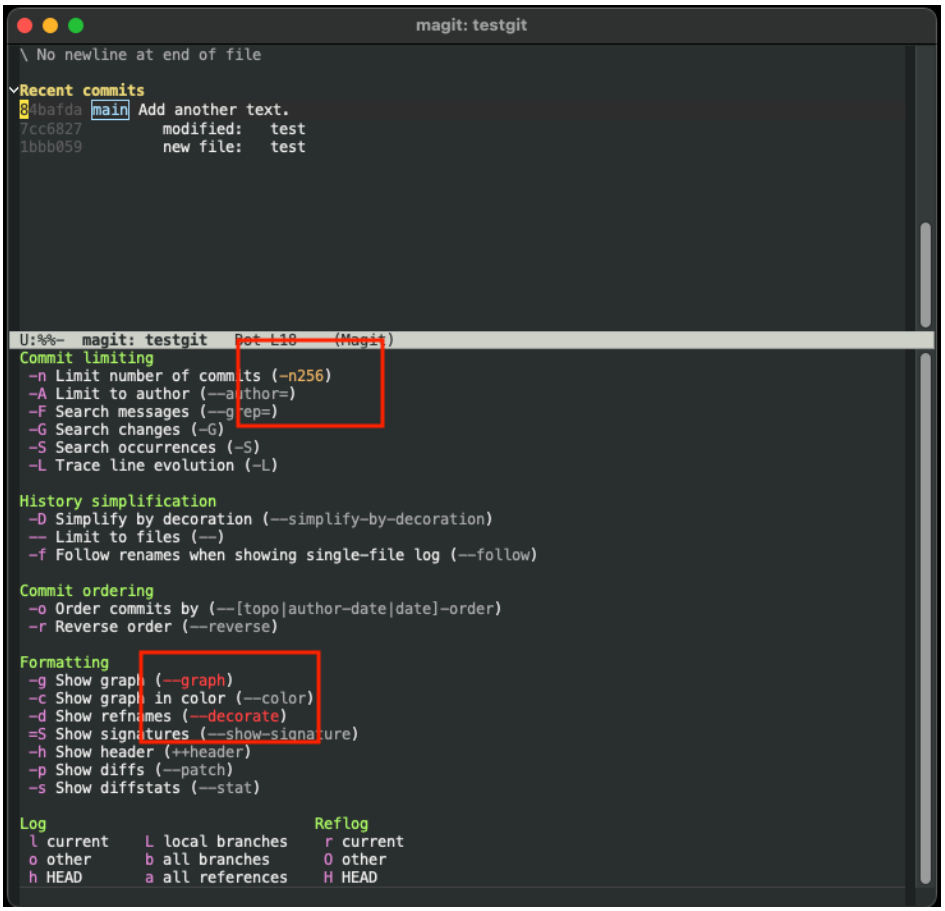
Figures 11.2: Transient commit arguments

Figure 11.2 show this effect. After the commit, the next time you commit, the argument is reset to the default value.

However the infix arguments of many other transient commands continue to have an effect even

after the git command that was called with those arguments has returned. The most important commands like this are those that display a diff or log in a dedicated buffer. Their arguments obviously continue to have an effect for as long as the respective diff or log is being displayed. Furthermore the used arguments are stored in buffer-local variables for future reference. We can also use **C-M-p** and **C-M-n** to previous and next arguments stored in buffer-local variables.

It is also possible to change the diff and log arguments used in the current buffer (including the status buffer, which contains both diff and log sections) using the respective "refresh" transient prefix commands on **D** and **L**. (**d** and **l** on the other hand are intended to change what diff or log is being displayed. It is possible to also change how the diff or log is being displayed at the same time, but if you only want to do the latter, then you should use the refresh variants.) Because these secondary diff and log transient prefixes are about changing the arguments used in the current buffer, they always start out with the set of arguments that are currently in effect in that buffer.



Figures 11.3: Transient log arguments

Figure 11.3 show this effect. The arguments are still there.

11.3.5 Running Git

Magit runs Git either for side-effects (e.g. when pushing) or to get some value (e.g. the name of the current branch). When Git is run for side-effects, the process output is logged in a per-repository log buffer, which can be consulted using the **magit-process** command when things don't go as expected.

Binding	Meaning
$	Displays the process buffer for the current repository.
k	Kills the process represented by the section at point.
!	Run git command

Tables 11.4: Git commands

11.4 Inspecting

The functionality provided by Magit can be roughly divided into three groups:

- ♠ inspecting existing data
- ♠ manipulating existing data or adding new data
- ♠ transferring data

Group	Binding	Meaning
status	**C-x g**	Show status buffer.
logging	**l**	show a commit or reference log.
	L	Change the arguments used for the log(s) in the current buffer.
	Y	Show commits that are in a certain branch but that have not been merged in the upstream branch.
diffing	**d**	Show changes between different versions.
	D	Change the arguments used for the diff(s) in the current buffer.
ediffing	**e**	Compare, stage, or resolve using Ediff. This command tries to guess what file, and what commit or range the user wants to compare, stage, or resolve using Ediff.
	E	Show differences using the Ediff package.
references	**y**	Lists branches and tags in a dedicated buffer.
bisecting	**B**	Narrow in on the commit that introduced a bug.

Tables 11.5: Inspecting Commands

11.5 Manipulating

Group	Binding	Meaning
init	**I**	Initialize a repository and then shows the status buffer for the new repository.
clone	**C**	Clone a repository.
staging	**s**	Add the change at point to the staging area.
	S	Stage all changes to files modified in the worktree.
unstaging	**u**	Remove the change at point from the staging area.
	U	Remove all changes from the staging area.
applying	**a**	Apply the change at point to the working tree.
	k	Remove the change at point from the working tree.
	v	Reverse the change at point in the working tree.
commit	**c**	Reverse the change at point in the working tree.
branching	**b**	Add, configure or remove a branch.
merging	**m**	Merge branches.
rebasing	**r**	Transplant commits and/or modify existing commits.
cherry picking	**A**	Apply or transplant commits.
	V	Revert existing commits, with or without creating new commits.
resetting	**x**	Reset the HEAD and index to some commit read from the user and defaulting to the commit at point, and possibly also reset the working tree. With a prefix argument reset the working tree otherwise don't.
	X	Reset the HEAD and index to some commit read from the user and defaulting to the commit at point. The working tree is kept as-is.
stashing	**z**	Stash uncommitted changes.
tag	**t**	Create or delete a tag.
note	**T**	Edit notes attached to commits.
submodules	**o**	Act on a submodule.
subtree	**O**	Import or export subtrees.
worktree	**Z**	Act on a worktree.

Tables 11.6: Manipulating commands

11.6 Transferring

Group	Binding	Meaning
remote	**M**	Add, configure or remove a remote.
fetching	**f**	Fetch from repository.
pulling	**F**	Pull from repository.
pushing	**P**	Push to repository.
patches	**W**	Create or apply patches.
	w	Apply patches received by email.

Tables 11.7: Manipulating commands

Chapter 12

Org

Org is a mode for keeping notes, maintaining TODO lists, and project planning with a fast and effective plain-text markup language. It also is an authoring system with unique support for literate programming and reproducible research. Org is implemented on top of Outline mode.

Files with the '.org' extension use Org mode by default.

12.1 Document Structure

12.1.1 Headlines

Headlines define the structure of an outline tree. Org headlines start on the left margin with one or more stars followed by a space.

```
* Top level headline
** Second level
*** Third level
    some text
*** Third level
    more text
* Another top level headline
```

12.1.2 Motion

Binding	Meaning
C-c C-n	Next heading
C-c C-p	Previous heading
C-c C-f	Next heading same level
C-c C-b	Previous heading same level
C-c C-u	Backward to higher level heading.

Tables 12.1: Motion commands

12.1.3 Visibility Cycling

Outlines make it possible to hide parts of the text in the buffer. Org uses just two commands, bound to **TAB** and **S-TAB** to change the visibility in the buffer.

Binding	Meaning
TAB	Rotate current subtree among "fold - children - subtree". Point must be on a headline for this to work.
S-TAB	Rotate the entire buffer among "overview - content - show all"
C-c C-r	Reveal context around point, showing the current entry, the following heading and the hierarchy above.
C-c C-k	Expose all the headings of the subtree, but not their bodies.
C-c TAB	Expose all direct children of the subtree. With a numeric prefix argument N, expose all children down to level N.
C-c C-x b	Show the current subtree in an indirect buffer

Tables 12.2: Visibility cycling commands

12.1.4 Structure Editing

Binding	Meaning
M-RET	Insert a new heading, item or row.
C-RET	Insert a new heading at the end of the current subtree.
M-S-RET	Insert new TODO entry with same level as current heading.
TAB	In a new entry with no text yet, the first TAB demotes the entry to become a child of the previous one. The next TAB makes it a parent, and so on, all the way to top level. Yet another TAB, and you are back to the initial level.
M-←	Promote current heading by one level.
M-→	Demote current heading by one level
M-S-←	Promote the current subtree by one level.
M-S-→	Demote the current subtree by one level.
M-↑	Move subtree up, i.e., swap with previous subtree of same level.
M-↓	Move subtree down, i.e., swap with next subtree of same level.
C-c @	Mark the subtree at point. Hitting repeatedly marks subsequent subtrees of the same level as the marked subtree.
C-c C-x C-w	Kill subtree, i.e., remove it from buffer but save in kill ring. With a numeric prefix argument N, kill N sequential subtrees.
C-c C-x M-w	Copy subtree to kill ring. With a numeric prefix argument N, copy the N sequential subtrees.
C-c C-x C-y	Yank subtree from kill ring. This does modify the level of the subtree to make sure the tree fits in nicely at the yank position.
C-c C-w	Move the entry or entries at point to another heading.
C-c ^	Sort same-level entries. When there is an active region, all entries in the region are sorted. Otherwise the children of the current headline are sorted.
C-x n s	Narrow buffer to current subtree.
C-x n b	Narrow buffer to current block.
C-x n w	Widen buffer to remove narrowing.
C-c *	Turn a normal line or plain list item into a headline—so that it becomes a subheading at its location.

Tables 12.3: Structure editing commands

12.1.5 Sparse Tree

An important feature of Org mode is the ability to construct **sparse trees** for selected information in an outline tree, so that the entire document is folded as much as possible, but the selected information is made visible along with the headline structure above it.

Binding	Meaning
C-c /	This prompts for an extra key to select a sparse-tree creating command.
M-g n or M-g M-n	Jump to the next sparse tree match in this buffer.
M-g p or M-g M-p	Jump to the previous sparse tree match in this buffer.

Tables 12.4: Sparse tree commands

12.1.6 Drawers

Sometimes you want to keep information associated with an entry, but you normally do not want to see it. For this, Org mode has **drawers**. They can contain anything but a headline and another drawer. Drawers look like this:

```
** This is a headline
Still outside the drawer
:DRAWERNAME:
This is inside the drawer.
:END:
After the drawer.
```

You can interactively insert a drawer at point by typing **C-c C-x d**. With an active region, this command puts the region inside the drawer. With a prefix argument, this command creates a 'PROPERTIES' drawer right below the current headline. Org mode uses this special drawer for storing properties. You cannot use it for anything else.

Visibility cycling on the headline hides and shows the entry, but keep the drawer collapsed to a single line. In order to look inside the drawer, you need to move point to the drawer line and press **TAB** there.

12.1.7 Block

Org mode uses **#+BEGIN ... #+END** blocks for various purposes from including source code examples to capturing time logging information. These blocks can be folded and unfolded by pressing TAB in the **#+BEGIN** line.

12.2 Plain Lists

Org knows ordered lists, unordered lists, and description lists.

- Unordered list items start with **-, +** or ***** as bullets.
- Ordered list items start with a numeral followed by either a period or a right parenthesis, such as **1.** or **1)**. If you want a list to start with a different value — e.g., 10 — start the text of the item with **[@10]**
- Description list items are unordered list items, and contain the separator **::** to distinguish the description term from the description.

The following commands act on items when point is in the first line of an item—the line with

the bullet or number. Some of them imply the application of automatic rules to keep list structure intact. If some of these actions get in your way, configure `org-list-automatic-rules` to disable them individually.

Binding	Meaning
TAB	Items can be folded or unfolded
M-RET	Insert new item at current level.
M-S-RET	Insert a new item with a checkbox
M-↑ or M-↓	Move the item including subitems up/down.
M-← or M-→	Decrease/increase the indentation of an item, leaving children alone.
M-S-← or M-S-→	Decrease/increase the indentation of the item, including subitems.
C-c C-c	If there is a checkbox in the item line, toggle the state of the checkbox.
C-c #	Update the statistic cookie in the current outline entry.
C-c -	Cycle the entire list level through the different itemize/enumerate bullets .
C-c *	Turn a plain list item into a headline.
C-c C-*	Turn the whole plain list into a subtree of the current heading.
C-c ^	Sort the plain list.

Tables 12.5: Plain list commands

12.2.1 Checkboxes

Every item in a plain list can be made into a checkbox by starting it with the string [ ]. This feature is similar to TODO items, but is more lightweight. Checkboxes are not included into the global TODO list, so they are often great to split a task into a number of simple steps.

```
* light task [25%]
  - [-] task 1 [33%]
    - [X] task 1-1
    - [ ] task 1-2
    - [ ] task 1-3
  - [X] task 2
  - [ ] task 3
  - [ ] task 4
```

12.3 Hyperlinks

12.3.1 Link Format

The general link format looks like this:

```
[[LINK][DESCRIPTION]]
or alternatively
[[LINK]]
```

Once a link in the buffer is complete, with all brackets present, Org changes the display so that **DESCRIPTION** is displayed instead of **[[LINK][DESCRIPTION]]** and **LINK** is displayed instead of **[[LINK]]**. You can directly edit the visible part of a link. This can be either the LINK part, if there is no description, or the DESCRIPTION part otherwise. To also edit the invisible LINK part, use **C-c C-l** with point on the link.

12.3.2 Internal Links

A link that does not look like a URL—i.e., does not start with a known scheme or a file name—refers to the current document. You can follow it with **C-c C-o** when point is on the link.

Org provides several refinements to internal navigation within a document. Most notably, a construct like `[[#my-custom-id]]` specifically targets the entry with the **CUSTOM_ID** property set to **my-custom-id**. Also, an internal link looking like `[[*Some section]]` points to a headline with the name **Some section**.

When the link does not belong to any of the cases above, Org looks for a dedicated target: the same string in double angular brackets, like **<<My Target>>**.

If no dedicated target exists, the link tries to match the exact name of an element within the buffer.

Following a link pushes a mark onto Org's own mark ring. You can return to the previous position with >C-c &. Using this command several times in direct succession goes back to positions recorded earlier.

12.3.3 External Links

External links are URL-like locators. They start with a short identifying string followed by a colon. There can be no space after the colon.

Here is the part set of built-in link types:

- ♠ file
  File links. File name may be remote, absolute, or relative. Additionally, you can specify a line number, or a text search. In Org files, you may link to a headline name, a custom ID, or a code reference instead. `file:/home/li/notebook/emacs/emacs.tex`.

- ♠ attachment
  Same as file links but for files and folders attached to the current node. Attachment links are intended to behave exactly as file links but for files relative to the attachment directory. `attachment:projects.org`.

- ♠ docview
  Link to a document opened with DocView mode. You may specify a page number.
  `docview:papers/last.pdf`.

- doi
  Link to an electronic resource, through its handle. `doi:10.1000/182`.
- elisp
  Execute an Elisp command upon activation. `elisp:(find-file "~/notebook")`.
- http
  `http://www.google.com`.
- https
  `https://www.google.com`.
- mailto
  Link to message composition. `mailto:mingmingli916@gmail.com`.
- shell
  Execute a shell command upon activation. `shell:date`.

12.3.4 Handling Links

Org provides methods to create a link in the correct syntax, to insert it into an Org file, and to follow the link. The main function is `org-store-link` (**C-c l**)[1]. It stores a link to the current location. The link is stored for later insertion into an Org buffer. The kind of link that is created depends on the current buffer:

- Org mode buffers
  For Org files, if there is a **<<target>>** at point, the link points to the target. Otherwise it points to the current headline. If the headline has a **CUSTOM_ID** property, store a link to this custom ID.
- Other files
  For any other file, the link points to the file, with a search string pointing to the contents of the current line. If there is an active region, the selected words form the basis of the search string.
- Agenda view
  The created link points to the entry referenced by the current line.

12.3.5 Link Abbreviations

Long URL can be cumbersome to type, and often many similar links are needed in a document. For this you can use link abbreviations. An abbreviated link looks like this:

```
[[linkword:tag][description]]
```

where the tag is optional. Abbreviations are resolved according to the information in the variable `org-link-abbrev-alist` that relates the linkwords to replacement text.

```
(setq org-link-abbrev-alist
      '(("google" . "https://www.google.com")))
```

If the replacement text contains the string **%s**, it is replaced with the tag. Using **%(my-function)** passes the tag to a custom Lisp function, and replace it by the resulting string.

[1] `(global-set-key (kbd "C-c l") #'org-store-link)`

12.3.6 Search Options in File Links

File links can contain additional information to make Emacs jump to a particular location in the file when following a link. This can be a line number or a search option after a double colon.

```
[[file:~/code/main.c::255]]
[[file:~/xx.org::My Target]]
[[file:~/xx.org::*My Target]]
[[file:~/xx.org::#my-custom-id]]
[[file:~/xx.org::/regexp/]]
[[attachment:main.c::255]]
```

12.3.7 Summary

Binding	Meaning
C-c C-l	Insert a link. With a **C-u** prefix, prompts for a file to link to. When point is on an existing link, edit the link and description parts of the link.
C-c C-o	Open the link when point is on the link.
C-c %	Push the current position onto the Org mark ring, to be able to return easily.
C-c &	Following a link pushes a mark onto Org's own mark ring. You can return to the previous position with this command.
M-n	Move forward to the next link in the buffer.[2]
M-p	Move backward to the previous link in the buffer.

Tables 12.6: Hyperlinks command summary

12.4 TODO items

Org mode does not maintain **TODO** lists as separate documents1. Instead, TODO items are an integral part of the notes file, because TODO items usually come up while taking notes! With Org mode, simply mark any entry in a tree as being a TODO item.

12.4.1 Basic TODO Functionality

Any headline becomes a TODO item when it starts with the word **TODO**.

[2]This is achieved by:

```
(with-eval-after-load 'org
  (define-key org-mode-map (kbd "M-n") #'org-next-link)
  (define-key org-mode-map (kbd "M-p") #'org-previous-link))
```

Binding	Meaning
C-c C-t	Roate the TODO state of the current item.
S-→ S-←	Select the following/preceding TODO state.
S-M-RET	Insert a new TODO entry below the current one.

Tables 12.7: Basic TODO commands

12.4.2 Extended Use of TODO Keywords

By default, marked TODO entries have one of only two states: TODO and DONE. Org mode allows you to classify TODO items in more complex ways with TODO keywords (stored in **org-todo-keywords**).

The structure of Org files makes it easy to define TODO dependencies. Usually, a parent TODO task should not be marked as done until all TODO subtasks, or children tasks, are marked as done. Sometimes there is a logical sequence to (sub)tasks, so that one subtask cannot be acted upon before all siblings above it have been marked as done. If you customize the variable `org-enforce-todo-dependencies`, Org blocks entries from changing state to DONE while they have TODO children that are not DONE. Furthermore, if an entry has a property **ORDERED**, each of its TODO children is blocked until all earlier siblings are marked as done.

Binding	Meaning
C-c C-x o	Toggle the 'ORDERED' property of the current entry.

Tables 12.8: Extended TODO commands

12.4.3 Progress Logging

Binding	Meaning
C-u C-c C-t	Prompt for a note and record the time of the TODO state change

Tables 12.9: Progress logging commands

12.4.4 Priorities

Prioritizing can be done by placing a **priority cookie** into the headline of a TODO item right after the TODO keyword, like this:

```
* TODO [#A] Summarize org mode
```

By default, Org mode supports three priorities: 'A', 'B', and 'C'. 'A' is the highest priority. An entry without a cookie is treated as equivalent if it had priority 'B'. Priorities make a difference only for sorting in the agenda. Outside the agenda, they have no inherent meaning to Org mode.

Priorities can be attached to any outline node; they do not need to be TODO items.

Binding	Meaning
C-c ,	Set the priority of the current headline. The command prompts for a priority character 'A', 'B' or 'C'. When you press SPC instead, the priority cookie, if one is set, is removed from the headline.
S-↑	Increase the priority of the current headline.
S-↓	Decrease the priority of the current headline.

Tables 12.10

12.4.5 Breaking Down Tasks 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 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.

12.5 Dates and Times

To assist project planning, TODO items can be labeled with a date and/or a time. The specially formatted string carrying the date and time information is called a **timestamp** in Org mode.

12.5.1 Timestamps

A timestamp is a specification of a date in a special format. A timestamp can appear anywhere in the headline or body of an Org tree entry. Its presence causes entries to be shown on specific dates in the agenda.

There are the following timestamps:

- Plain timestamp
  A simple timestamp just assigns a date/time to an item.
  `<2022-11-07 Mon 14:00>`
- Timestamp with repeater interval
  A timestamp may contain a **repeater interval**, indicating that it applies not only on the given date, but again and again after a certain interval of N days (d), weeks (w), months (m), or years (y).
  `<2022-11-07 Mon 7:00 +1d>`
- Diary-style expression entries
  For more complex date specifications, Org mode supports using the special expression diary entries implemented in the Emacs Calendar package.
  `<%%(diary-float t 4 2)>`
- Time/Data range
  Two timestamps connected by **--** denote a range.

`<2022-11-07 Mon>--<2022-11-08 Tue>`

♠ Inactive timestamp
Just like a plain timestamp, but with square brackets instead of angular ones. These timestamps are inactive in the sense that they do not trigger an entry to show up in the agenda.
`[2022-11-07 Mon]`

12.5.2 Creating Timestamps

For Org mode to recognize timestamps, they need to be in the specific format. All commands listed in Table 12.11 produce timestamps in the correct format.

Binding	Meaning
C-c .	Prompt for a date and insert a corresponding timestamp. When point is at an existing timestamp in the buffer, the command is used to modify this timestamp. When this command is used twice in succession, a time range is inserted.
C-u C-c .	Use the alternative format which contains date and time.
C-u C-u C-c .	Insert an active timestamp with the current time without prompting.
C-c !	Like C-c ., but insert an inactive timestamp that does not cause an agenda entry.
C-c C-c	Normalize timestamp, insert or fix day name if missing or wrong.
C-c <	Insert a timestamp corresponding to point date in the calendar.
C-c >	Access the Emacs calendar for the current date. If there is a timestamp in the current line, go to the corresponding date instead.
C-c C-o	Access the agenda for the date given by the timestamp or -range at point.
S-← or S-→	Change date at point by one day
S-↑ or S-↓	On the beginning or enclosing bracket of a timestamp, change its type. Within a timestamp, change the item under point. Point can be on a year, month, day, hour or minute. When the timestamp contains a time range like '15:30-16:30', modifying the first time also shifts the second, shifting the time block with constant length. To change the length, modify the second time.
C-c C-y	Evaluate a time range by computing the difference between start and end. With a prefix argument, insert result after the time range.

Tables 12.11: Creating timestamp commands

When Org mode prompts for a date/time, the default is shown in default date/time format, and the prompt therefore seems to ask for a specific format. But it in fact accepts date/time information

in a variety of formats. Generally, the information should start at the beginning of the string. Org mode finds whatever information is in there and derives anything you have not specified from the **default date and time**. The default is usually the current date and time, but when modifying an existing timestamp, or when entering the second stamp of a range, it is taken from the stamp in the buffer. When filling in information, Org mode assumes that most of the time you want to enter a date in the future: if you omit the month/year and the given day/month is before today, it assumes that you mean a future date.

For example, let's assume that today is June 13, 2006.

```
'3-2-5'                 =>  2003-02-05
'2/5/3'                 =>  2003-02-05
'14'                    =>  2006-06-14
'12'                    =>  2006-07-12
'2/5'                   =>  2007-02-05
'Fri'                   =>  nearest Friday (default date or later)
'sep 15'                =>  2006-09-15
'feb 15'                =>  2007-02-15
'sep 12 9'              =>  2009-09-12
'12:45'                 =>  2006-06-13 12:45
'22 sept 0:34'          =>  2006-09-22 0:34
'w4'                    =>  ISO week for of the current year 2006
'2012 w4 fri'           =>  Friday of ISO week 4 in 2012
'2012-w04-5'            =>  Same as above
```

Furthermore you can specify a relative date by giving, as the first thing in the input: a plus/minus sign, a number and a letter—'h', 'd', 'w', 'm' or 'y'—to indicate a change in hours, days, weeks, months, or years. With 'h' the date is relative to the current time, with the other letters and a single plus or minus, the date is relative to today at 00:00. With a double plus or minus, it is relative to the default date. If instead of a single letter, you use the abbreviation of day name, the date is the Nth such day.

```
'+0'            =>  today
'.'             =>  today
'+2h'           =>  two hours from now
'+4d'           =>  four days from today
'+4'            =>  same as +4d
'+2w'           =>  two weeks from today
'++5'           =>  five days from default date
'+2tue'         =>  second Tuesday from now
```

Parallel to the minibuffer prompt, a calendar is popped up. You can control the calendar fully from the minibuffer:

Binding	Meaning
RET	Choose date at point in calendar.
S-→	One day forward.
S-←	One day backward.
S-↓	One week forward.
S-↑	One week backward.
M-S-→	One month forward.
M-S-←	One month backward.
>	Scroll calendar forward by one month.
<	Scroll calendar backward by one month.
C-v	Scroll calendar forward by 3 months.
M-v	Scroll calendar backward by 3 months.
C-.	Select today's date.

Tables 12.12: Minibuffer commands

12.5.3 Deadlines and Scheduling

A timestamp may be preceded by special keywords to facilitate planning. Both the timestamp and the keyword have to be positioned immediately after the task they refer to.

♠ **DEADLINE**

```
* TODO Task
  DEADLINE: <2022-11-07 Mon>
```

♠ **SCHEDULED**

```
* TODO Task
  SCHEDULED: <2022-11-07 Mon>
```

Binding	Meaning
C-c C-d	Insert **DEADLINE** keyword along with a stamp.
C-c C-s	Insert **SCHEDULED** keyword along with a stamp.

Tables 12.13

Repeated tasks

Some tasks need to be repeated again and again. Org mode helps to organize such tasks using a so-called **repeater** in a 'DEADLINE', 'SCHEDULED', or plain timestamps.

```
** TODO Pay the rent
   DEADLINE: <2022-11-07 Mon +1m>
```

the **+1m** is a repeater; the intended interpretation is that the task has a deadline on '<2022-11-07>' and repeats itself every (one) month starting from that time. You can use yearly, monthly,

weekly, daily and hourly repeat cookies by using the ‘y’, ‘m’, ‘w’, ‘d’ and ‘h’ letters. If you need both a repeater and a special warning period in a deadline entry, the repeater should come first and the warning period last.

```
DEADLINE: <2022-11-07 Mon +1m -3d>
```

Deadlines and scheduled items produce entries in the agenda when they are over-due, so it is important to be able to mark such an entry as done once you have done so. When you mark a ‘DEADLINE’ or a ‘SCHEDULED’ with the TODO keyword ‘DONE’, it no longer produces entries in the agenda. The problem with this is that then also the next instance of the repeated entry will not be active. Org mode deals with this in the following way: when you try to mark such an entry as done, using C-c C-t, it shifts the base date of the repeating timestamp by the repeater interval, and immediately sets the entry state back to TODO. In the example above, setting the state to ‘DONE’ would actually switch the date like this:

```
** TODO Pay the rent
   DEADLINE: <2022-12-07 Wed +1m>
```

To mark a task with a repeater as DONE, use **C-- 1 C-c C-t**, i.e., with a numeric prefix argument of **-1**.

With the **+1m** cookie, the date shift is always exactly one month. Org mode has special repeaters **++** and **.+**.

```
** TODO Call Father
   DEADLINE: <2008-02-10 Sun ++1w>
   Marking this DONE shifts the date by at least one week, but also
   by as many weeks as it takes to get this date into the future.
   However, it stays on a Sunday, even if you called and marked it
   done on Saturday.

** TODO Check the batteries in the car
   DEADLINE: <2005-11-01 Tue .+1m>
   Marking this DONE shifts the date to one month after today.
```

12.5.4 Clocking Work Time

Binding	Meaning
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. With two C-u C-u prefixes, clock into the task at point and mark it as the default task. With three C-u C-u C-u prefixes, force continuous clocking by starting the clock when the last clock stopped.
C-c C-x C-o	Stop the clock (clock-out).
C-c C-x C-x	Re-clock the last clocked task. With one C-u prefix argument, select the task from the clock history. With two C-u prefixes, force continuous clocking by starting the clock when the last clock stopped.
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.
C-c C-x C-d	Display time summaries for each subtree in the current buffer.
C-c C-x x	Insert or update a clock table.

Tables 12.14: Clocking commands

Effort Estimates

Binding	Meaning
C-c C-x e	Set the effort estimate for the current entry.
C-c C-x C-e	Modify the effort estimate of the item currently being clocked.

Tables 12.15: Effort commands

Relative Timer

Org provides two types of timers. There is a relative timer that counts up, which can be useful when taking notes during, for example, a meeting or a video viewing. There is also a countdown timer. The relative and countdown are started with separate commands.

Binding	Meaning
C-c C-x 0	Start or reset the relative timer. By default, the timer is set to 0. When called with a C-u prefix, prompt the user for a starting offset.
C-c C-x ;	Start a countdown timer.
Once started, relative and countdown timers are controlled with the same commands.	
C-c C-x .	Insert a relative time into the buffer.
C-c C-x -	Insert a description list item with the current relative time. With a prefix argument, first reset the timer to 0.
C-c C-x ,	Pause the timer, or continue it if it is already paused.
C-c C-x _	Stop the timer.

Tables 12.16: Timer

12.6 Tags

An excellent way to implement labels and contexts for cross-correlating information is to assign **tags** to headlines. Every headline can contain a list of tags; they occur at the end of the headline. Tags are normal words containing letters, numbers, '_', and '@'. Tags must be preceded and followed by a single colon, e.g., **:work:**. Several tags can be specified, as in **:work:urgent:**.

Tags make use of the hierarchical structure of outline trees. If a heading has a certain tag, all subheadings inherit the tag as well. To limit tag inheritance to specific tags, or to turn it off entirely, use the variables `org-use-tag-inheritance` and `org-tags-exclude-from-inheritance`.

Binding	Meaning
C-c C-q	Enter new tags for the current headline.
C-u C-c C-q	Align all tags to make things look nice.

Tables 12.17: Tag commands

Org supports tag insertion based on a list of tags. By default this list is constructed dynamically, containing all tags currently used in the buffer. You may also globally specify a hard list of tags with the variable `org-tag-alist`.

12.7 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.

Org can select items based on various criteria and display them in a separate buffer. Six different view types are provided:

- **agenda** that is like a calendar and shows information for specific dates,
- **TODO list** that covers all unfinished action items,
- **match view**, showings headlines based on the tags, properties, and TODO state associated with them,
- **text search view** that shows all entries from multiple files that contain specified key- words,
- **stuck projects view** showing projects that currently do not move along, and
- **custom views** that are special searches and combinations of different views.

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.

12.7.1 Agenda Files

The information to be shown is normally collected from all agenda files, the files listed in the variable `org-agenda-files`. If a directory is part of this list, all files with the extension '.org' in this directory are part of the list.

Binding	Meaning
C-c [	Add current file to the list of agenda files.
C-c]	Remove current file from the list of agenda files.
C-,	Cycle through agenda file list, visiting one file after the other.
C-c C-x <	Restrict the agenda to the current subtree.
C-c C-x >	Remove the restriction created by C-c C-x <.

Tables 12.18: Agenda files commands

12.7.2 The Agenda Dispatcher

The views are created through a dispatcher, accessible with **C-c a**. It displays a menu from which an additional letter is required to execute a command.

12.8 org-roam

Org-roam is a tool for networked thought. It reproduces some of Roam Research's[3] key features within Org-mode.

Org-roam implement the "Zettelkasten" method or "slip-box" digitally. The Zettelkasten is a personal tool for thinking and writing. It places heavy emphasis on connecting ideas, building up a web of thought.

This method is attributed to German sociologist Niklas Luhmann, who using the method had produced volumes of written works. Luhmann's slip-box was simply a box of cards. These cards are small – often only large enough to fit a single concept. The size limitation encourages ideas to be broken down into individual concepts. These ideas are explicitly linked together. The breakdown of ideas encourages tangential exploration of ideas, increasing the surface for thought. Making linking explicit between notes also encourages one to think about the connections between concepts.

[3] https://roamresearch.com/

At the corner of each note, Luhmann ascribed each note with an ordered ID, allowing him to link and jump between notes. In Org-roam, we simply use hyperlinks.

Org-roam is the slip-box, digitalized in Org-mode. Every zettel (card) is a plain-text, Org-mode file. In the same way one would maintain a paper slip-box.

12.8.1 Installation

Add the following into your emacs configuration file.

```
(require 'package)
(add-to-list 'package-archives
             '("melpa" . "http://melpa.org/packages/") t)

(add-to-list 'package-archives '("org" . "https://orgmode.org/elpa/") t)
```

Then execute the following command in emacs:

```
M-x package-refresh-contents RET
M-x package-install RET org-roam RET
```

12.8.2 The Basic

The Org-roam Node

A node is any headline or top level file with an ID. For example, with this example file content:

```
:PROPERTIES:
:ID:       foo
:END:
#+title: Foo

* Bar
:PROPERTIES:
:ID:       bar
:END:
```

We create two nodes:

- A file node “Foo” with id foo.
- A headline node “Bar” with id bar.

Links between Nodes

Headlines without IDs will not be considered Org-roam nodes. Org IDs can be added to files or headlines via the interactive command `M-x org-id-get-create`.

We link between nodes using Org’s standard ID link (e.g. id:foo). While only ID links will be considered during the computation of links between nodes, Org-roam caches all other links in the documents for external use.

Setting up Org-roam

Org-roam's capabilities stem from its aggressive caching: it crawls all files within `org-roam-directory`, and maintains a cache of all links and nodes.

To start using Org-roam, pick a location to store the Org-roam files. The directory that will contain your notes is specified by the variable `org-roam-directory`. Org-roam searches recursively within `org-roam-directory` for notes. This variable needs to be set before any calls to Org-roam functions.

```
(use-package org-roam
  :ensure t
  :config
  ;; set org-roam-directory.
  ;; The file-truename function is only necessary when you use symbolic links inside org-roam-directory: Org-roam does not resolve symbolic links.
  ;; One can however instruct Emacs to always resolve symlinks, at a performance cost:
  ;; (setq find-file-visit-truename t)
  (setq org-roam-directory (file-truename "~/note/org-roam")
        org-roam-capture-templates
        '(("d" "default" plain "%?" :target
           (file+head "${slug}-%<%Y%m%d%H%M%S>.org" "#+title: ${title}
")
           :unnarrowed t)
          ("m" "machine learning" plain "%?" :target
           (file+head "machine-learning/${slug}-%<%Y%m%d%H%M%S>.org" "#+title: ${title}
")
           :unnarrowed t)))
  ;; setup Org-roam to run functions on file changes to maintain cache consistency.
  (org-roam-db-autosync-mode)
  :bind (("\C-ci" . org-roam-node-insert)
         ("\C-cf" . org-roam-node-find)))

(use-package org-roam-ui
  :after org-roam
  ;;         normally we'd recommend hooking orui after org-roam, but since org-roam does not have
  ;;         a hookable mode anymore, you're advised to pick something yourself
  ;;         if you don't care about startup time, use
  ;;  :hook (after-init . org-roam-ui-mode)
  :config
  (setq org-roam-ui-sync-theme t
        org-roam-ui-follow t
        org-roam-ui-update-on-save t
        org-roam-ui-open-on-start t))

;; Use M-x org-roam-ui-mode RET to enable the global mode.
;; It will start a web server on http://127.0.0.1:35901/ and connect to it
```

```
    via a WebSocket for real-time updates.
```

Creating and Linking Nodes

- **org-roam-node-insert**: creates a node if it does not exist, and inserts a link to the node at point.
- **org-roam-node-find**: creates a node if it does not exist, and visits the node.
- **org-roam-capture**: creates a node if it does not exist, and restores the current window configuration upon completion.

If you want to insert whitespace in node, use `C-q` prefix.

Chapter 13

Python

Here we describe how to configure Emacs as a Python IDE for better Python development.

lsp-mode (Lanuage Server Protocol) aims to provide IDE-like experience by providing optional integration with the most popular Emacs packages like `company`, `flycheck` and `projectile`.

13.1 Installtion

13.1.1 Client

You need first `lsp-mode`, that is a Emacs client for an LSP server. Then you need to install the specific LSP server for your language. Finally, call **M-x lsp** or use the corresponding major mode hook to autostart the server.

```
;; Use package manager interface.
(require 'package)

;; Add melpa site to package archives.
;; This is used define where to fetch 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)

;; Load Emacs Lisp packages, and activate them.
(package-initialize)


;; The use-package macro allows you to isolate package configuration in your
    .emacs file
;; in a way that is both performance-oriented and, well, tidy.
(require 'use-package)
```

```
(use-package lsp-mode
  :init
  ;; set prefix for lsp-command-keymap (few alternatives - "C-l", "C-c l")
  (setq lsp-keymap-prefix "C-c l")
  :hook (;; replace XXX-mode with concrete major-mode(e. g. python-mode)
         ;; (XXX-mode . lsp)
         (python-mode . lsp)
         ;; if you want which-key integration
         (lsp-mode . lsp-enable-which-key-integration))
  :commands lsp)

;; optionally
(use-package lsp-ui :commands lsp-ui-mode)


;; optionally if you want to use debugger
(use-package dap-mode)
;; (use-package dap-LANGUAGE) to load the dap adapter for your language
(use-package dap-python)

;; optional if you want which-key integration
(use-package which-key
  :config
  (which-key-mode))
```

13.1.2 Install a language server

Here we use Python LSP Server.

```
pip install 'python-lsp-server[all]'
```

Bibliography

[1] Debra Cameron. *Learning GNU Emacs*. O'Reilly Media, Inc., 2004.

[2] D.E. Knuth. *The TEXbook*. Addison Wesley, 1986.

[3] D.E. Knuth. Typesetting concrete mathematics. *TUGboat*, 10(1):31–36, April 1989.