1

Why Use Emacs?

Most people believe that the first step in becoming more productive is finding the perfect software tool to help you manage the vast amounts of information we consume daily. Most people manage a complex web of tools to satisfy their needs: email clients, note-taking applications, task managers, word processors and so on are needed to complete a project. This chapter explains why you should use Emacs for writing, as you can use it to undertake almost every task in your writing project, from ideation to publication.

I used to hop from application to application—jumping from the action list to my schedule, onward to the word processor, spreadsheet, PDF reader, and so on. Wouldn't it be nice if there was one program that could help you with almost all your tasks?

This book is written for authors who have been frustrated by commercial software and are looking for a more efficient workflow and flexibility in their computing needs. This books covers everything you need to know to do read books in Emacs, write notes, write books, articles or websites and publish them electronically or for print. You don't need to be a computer wiz to use Emacs as a writer. This books provides a gentle introduction to flatten the learning curve as much as possible. Most existing Emacs documentation assumes prior knowledge and is written for knowledgeable users. This book is written for beginners with not experience with writing code. These chapters peel Emacs like an onion, revealing one aspect of using this software at a time. Each chapter adding more detail and complexity.

Welcome to the Emacs computing system, the Swiss army chainsaw of productivity.

1.1 What is Emacs?

The official tagline of Emacs is that it is an "extensible self-documenting text editor". These words barely do justice to Emacs because they focus on its original purpose as a software development tool. Emacs is a multi-purpose computing environment that can help you manage your information, track projects, write and publish articles, books, websites and any other text-based activity. Emacs is not a productivity hack; it is a productivity hacking system.

The first version of Emacs was released almost fifty years ago (Stallman, 1981b), which might seem like obsolete software. However, a vibrant community of developers continually improves the system. Emacs users are happy to share their configurations and have developed thousands of packages that extend the system's functionality.

Many versions of Emacs have existed over the decades. The most widely used version is GNU Emacs, first released by Richard Stallmann in 1984 (Johnson, 2022). GNU Emacs (further referred to as Emacs) is free software released by the Free Software Foundation. The foundation loosely defines free software as:[1]

"Free software" means software that respects users' freedom and community. Roughly, it

[1] Free Software Foundation. What is Free Software? `https://www.gnu.org/philosophy/free-sw.en.html`

> means that the users have the freedom to run, copy, distribute, study, change and improve the software. Thus, "free software" is a matter of liberty, not price. To understand the concept, you should think of "free" as in "free speech," not as in "free beer."

Emacs is a text editor, but this needs to be clarified for authors. A text editor is more akin to developing software. From an author's perspective, Emacs is a text processor, and editing is only the last step in writing an article or book. A text editor is a tool for programmers to write code, and a text processor is a tool for authors to write prose. *Emacs Writing Studio* (EWS) is a bespoke configuration of Emacs that turns it into a tool for authors.

When you use a computer, you more than likely write a memo in a word processor, then switch to a spreadsheet to create a graph, which you copy and paste into your memo. Next, you open your email client to send the memo to your colleague. The last step is to close the action in your productivity tool. After this hard work, you might play a game and open Tetris. You have to learn different skills for each application when using these other programs. Each program has a different internal logic and workflow to complete your tasks. Most software is non-malleable, meaning you have to use it the way the developers intended you to use it.

Working with Emacs is a different experience. You write the memo in Emacs, create the graph with Emacs, email the result and close the action with Emacs. Finally, you play the game, you guessed it, with Emacs. If you use Emacs, you only need to learn one program, and you can configure and extend it to how you prefer to use it, not how a programmer wants you to use the software.

Emacs looks different from the modern graphical interfaces most computer users are accustomed to. Most users remove any graphical elements from the screen, leaving a display with only text in all colours of the rainbow. The default interface is perhaps not as appealing as a modern graphical interface with pretty pictures to click on but don't let this austere aesthetic fool you, as, under the hood, Emacs is a modern and robust computing environment that is fully configurable to your liking.

1.2 Why use Emacs?

Emacs is a 'malleable software' platform, meaning you are free to change and enhance how it works. This malleability ensures that Emacs can perform any task that you can undertake with a keyboard. As such, it is the Swiss Army Chainsaw of productivity.

The first principle of malleable software is that it is easy to change.[2] With Emacs you can build applications using the LISP language, abbreviated as Elisp (Monnier & Sperber, 2020). This task might sound daunting, but it is about the possibility. Writing code is optional because most Emacs users share what they have developed, so you can freely copy their work. You can extend and configure Emacs with any of the thousands of freely available packages.

Users can configure almost everything in the system with little knowledge of Elisp. This knowledge requirement might seem like a hurdle, but learning how to use it will give you nearly unlimited power over how you use your computer. Software should adjust to the user, not vice versa.

The advantage of this approach is that you have complete freedom when using this software. You can instruct it to do almost anything you like and configure it to your specific needs. The disadvantage is that it requires a different computing approach than contemporary software. Using Emacs throws you back to the original intent of using a computer and genuine user-friendliness. Are you ready to change the way you use your computer? To paraphrase a famous scene from The Matrix:

[2] Malleable Systems Collective, `https://malleable.systems/`

nice way to say it

If you take the blue Microsoft pill, the story ends, and everything stays the same. If you take the purple Emacs pill, you stay in Wonderland, and I show you how deep the rabbit hole goes.

1.3 Redefining User-Friendliness

Emacs' lack of a slick graphical interface might discourage new users. Unfortunately, most people confuse user-friendliness with a smooth design and using a mouse. However, this approach is not user-friendly because the user loses freedom. Graphically driven software is a gilded cage. It might be pleasant to work in, but it is still a cage. The *What You See is What You Get* approach is outdated. This approach is only relevant for printed documents, while in reality, most writing remains in electronic format.

Emacs is a plain text editor that focuses on the meaning of words instead of how they will eventually look on a page or screen. Plain text is not the same as plain English; it relates to how the information is stored. Plain text is the opposite of rich text, which includes hidden definitions for font sizes, colours and other text and formatting attributes.

Plain text is not formatted and most commonly has a `.txt` extension. However, there are many other plain text formats, such as HTML, Markdown, LaTeX, and Org Mode. Windows users might be familiar with the venerable Notepad software (which is even older than GNU Emacs, but unlike Emacs, it has not grown beyond its original capabilities).

Plain text can be read across computer systems, so you never have to worry about locking your writing into a proprietary format and being stuck using a particular software package. The internet runs on plain text files, which will likely stay the same in the future.

Text modes can display 'graphics'. When I went to primary school in the 1970s, our teacher showed us some art printed with a computer. The art consisted of a series of keyboard characters that resemble a picture, such as this cute Linux penguin (Source: `asciiart.website`). However, there is no need to resort to these ancient techniques as Emacs can display images in the most common file forts such as JPG and PNG.

```
    .--.
   |o_o |
   |:_/ |
  //   \ \
 (|     | )
/'\_   _/`\
\___)=(___/
```

Graphical interfaces simulate the physical world by making objects on the screen look like pieces of paper and folders on a desk. You point, click and drag documents into folders; documents appear as they would on paper and when done, they go into the rubbish bin. Graphical interfaces are a magic trick that makes you believe you are doing something physical (Tognazzini, 1993). This approach might be convenient, but it prevents people from understanding how a computer works. In word processors, the screen looks like a printed page. While this might be aesthetically pleasing, it distracts the writer from creating content and instead motivates them to fiddle with formatting.

... these days

Only a tiny part of written text is printed on paper, so the *What You See is What You Get* (WYSIWYG) approach does not make much sense in the digital age. The graphical approach distracts the mind from the content and lures the user into working on style instead of writing text. Word processors hide the instructions for layout and typography behind the graphical interface. Office workers

around the globe waste a lot of time trying to format or typeset documents in graphical environments.

Following the plain text Emacs way helps you become more productive by not worrying about the document's design until you complete the text. As I write this book, it only takes a few keystrokes to convert the text into a fully formatted ebook in either PDF or ePub, ready for distribution. The main benefit of using plain text over rich text is that it provides a distraction-free writing environment. Plain text uses the *What You See is What You Mean* (WYSIWYM) approach. Instead of focusing on the format or presentation of the document, a WYSIWYM editor preserves the intended meaning of each element. For example, sections, paragraphs, illustrations and other document elements are labelled as such (Khalili & Auer, 2015).

As I write this book, I don't see what it will look like in printed form as you would using modern word processors. In Emacs, I only see text, images and some instructions for the computer on what the final product should look like. When exporting this document to a web page or any other format, a template defines the final product's design, such as layout and typography. This approach ensures that your text can be easily exported to multiple formats without loss of information. The image in figure 1.1 shows writing in Emacs in action. The left side shows the Emacs screen of a page on one of my websites. The right side shows the result after compiling the content. The top of the Emacs screen contains the metadata for this webpage, followed by the text. Fonts and colours have semantic meaning and do not necessarily relate to how they are displayed in the final product. The template, in this case a CSS file, determines the final product.

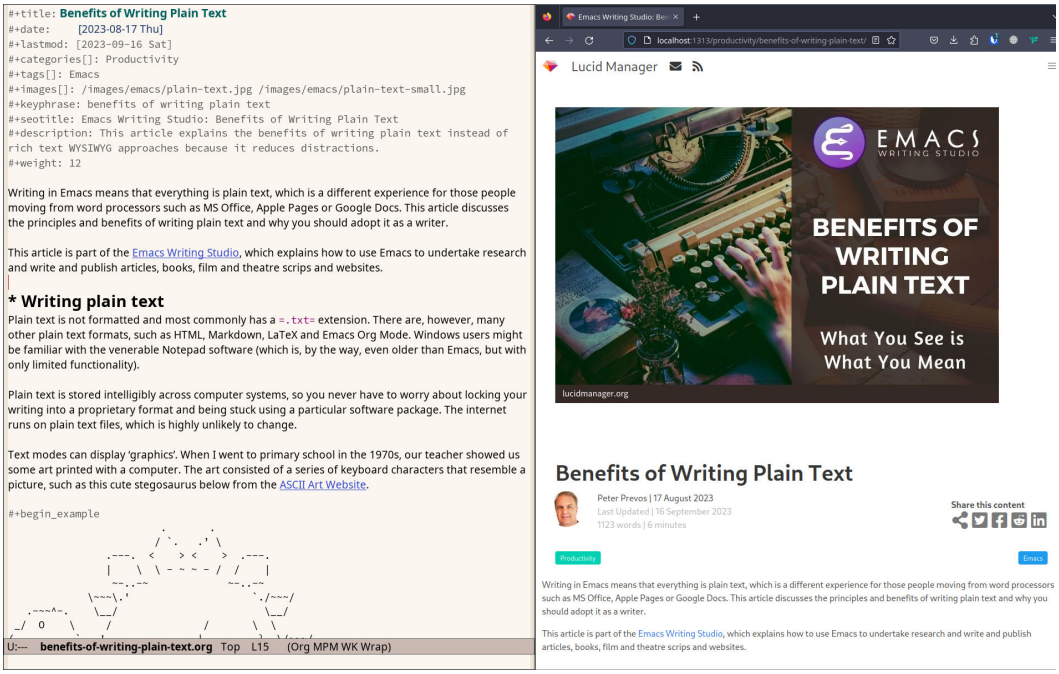

FIGURE 1.1 What You See is What You Mean approach to writing.

In a WYSIWYG word processor, formatting instructions are invisible to the user, which can cause repeated issues in getting the final result to look how you want it to. In plain text, the content and most semantics are directly visible and editable by the user. Regular plain text files are the simplest form of plain text and don't contain any semantics. Other plain text formats like HTML, LaTeX, Markdown and Org mode include instruction sets to define the final result. The table below shows how *italic text* is signified in five popular plain text formats.

TABLE 1.1 Italic text in common plain text formats.

Format	Italic semantics
TXT	No formatting
HTML	`<i>Italic Text</i>`
LaTeX	`\emph{Italic Text}`
Markdown	`**Italic Text**`
Org mode	`/Italic Text/`

The style sheet in plain text writing is separated from the text. This means that you can easily export your document to various formats. Org mode in Emacs has a potent export engine (Chapter 7) that allows you to convert your writing to a website, ebook or physical book with just a few keystrokes and some configuration.

In summary, the benefits of writing in plain text over using graphical software are:

1. The plain text is independent of the software you use.
2. Plain text removes distractions from the screen.
3. Plain text is versatile and can be exported to any other format.

1.4 The Learning Curve

The second hurdle is the steep learning curve. To make Emacs work for you, you must learn the basic principles of using this editor and some of the associated add-on packages. Within the computing community, the Emacs learning curve is well known and mocked in this internet meme (Figure 1.2). Perhaps Emacs is more complex than other plain text editors, but it also is much more powerful then any other tool. But with this great power comes great responsibility, so you have to learn some new skills to use it as your main writing tool.

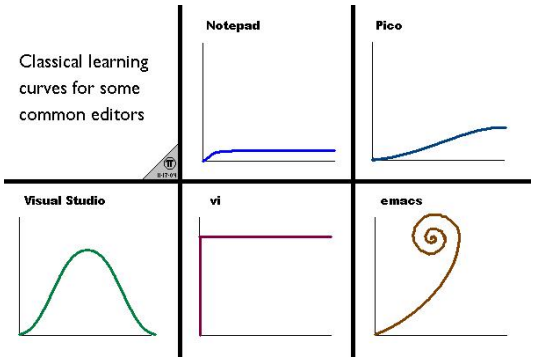

FIGURE 1.2 Learning curve for some common editors.

As Emacs is old software, some default settings and terminology differ from contemporary graphical software. For example, opening a file is 'visiting a file'. Pasting a text is' yanking', and cutting it is the same as killing. Perhaps the Emacs terminology is more poetic than the current terms, based on handicraft activities, such as cutting, pasting, and moving files between folders as if they were pieces of paper.

The key to learning Emacs is not to get overwhelmed by the virtually infinite configuration options but to master only those bits of functionality that you need to do what you need to do. Even without any configuration, you can do a lot with Emacs.

You should eventually learn some Elisp to configure the software to your wishes. While that might sound daunting, you can simply copy and paste (kill and yank) examples from the internet. *Emacs Writing Studio* is optimised for authors to shorten this learning curve. Just remember that the steeper the learning curve, the bigger the reward.

EWS provides authors with a fine-tuned configuration to convert vanilla Emacs into a research, writing, and publication engine. This book focuses on using this configuration instead of delving deeply into the technical details. The Appendix describes the full configuration for readers interested in venturing into the depths of Emacs Lisp.

1.5 Advantages and Limitations of Emacs

In summary, these are some of the significant advantages of using Emacs to create written content:

1. One piece of software to undertake most of your computing activities makes you more productive because you only need to master one system.
2. You store all your information in plain text files. You will never have any problems with esoteric file formats.
3. You can modify almost everything in the software to suit your workflow.
4. Emacs runs on all major operating systems: GNU/Linux, Windows, Chrome, and MacOS.
5. Emacs is free (libre) software supported by a large community willing to help.

After singing the praises of this multi-functional editor, you would almost think that Emacs is the omnipotent god of software. Some people even have established the *Church of Emacs* as a mock religion to express their admiration for this supremely malleable software environment.

Emacs can display images and integrate them with text, but it has limited functionality in creating or modifying graphical files. If you need to create or edit pictures, consider using GIMP (GNU Image Manipulation Program). Video content is unsupported other than hyperlinks to a file or website.

The second disadvantage is that Emacs must still include a fully operational web browser. You can surf the web within Emacs, but only within the limitations of a plain text interface.

Lastly, Emacs risks becoming a productivity sink. Just because you can configure everything does not mean you should. Don't spend too much time *on* your workflow. Spend this time *in* your workflow being creative. Most productivity hacks do not significantly impact your output because you write with your mind, not the keyboard.

1.6 How to Read this Book

This book is not a manual on using Emacs but a guided tour for authors. It describes the typical use cases for an author and how to implement these using Emacs. Each chapter contains references to the comprehensive built-in help system for the reader to find out more details. The knowledge

in this book is enough to get you started on your writing project, and Emacs itself contains all the documentation you need to become a ninja at the keyboard.

The next chapter explains the principles of using an unconfigured vanilla GNU Emacs system to get you started on the learning curve. This chapter also explains the principles of modifying and enhancing Emacs and how to install the EWS configuration.

Chapter three takes you through the principles of using *Emacs Writing Studio* and how it is different from an unconfigured Emacs experience. The EWS configuration changes how Emacs looks and feels and adds enhancements to help you find the information you need. EWS also uses a series of external packages to help authors, such as the Denote note-taking tool. The guiding principle of EWS is to stay as close to the vanilla Emacs experience as is humanly bearable.

The remainder of the book follows the EWS workflow, which is based on the typical workflow for an author. Chapters four to eight describe how to use Emacs to implement this workflow.

4. Inspiration: Reading, Listening and Watching
5. Ideation: Recording and Managing Ideas
6. Production: Writing and editing
7. Publication: Sharing Your Writing with the World
8. Administration: Manage your Tasks and Files

The final chapter contains some advice on how to become an Emacs Ninja.

2

Getting Started with Vanilla Emacs

Start your engines; it is time to use Emacs. The installation process for Emacs depends on your operating system. Emacs is not standalone software and interfaces with other free programs; you must install additional software. The GNU Emacs website (`gnu.org/software/emacs`) contains instructions on installing Emacs on the most common operating systems. Please note that you will need the latest version, which at the time of writing is 29.3. The relevant chapters in this book provide instructions on which additional software is required.

This can be hard for newcomers. Even I am only on 28.1 using an Ubuntu 22.04 LTS.

This chapter explains the basic principles of using Emacs without configuration, also called vanilla Emacs. Emacs' methods and vocabulary seem foreign compared to other contemporary software. The main reason for these differences is that the development of Emacs started in 1974, a time when computing was notably different to our current experience. The Emacs vocabulary is vestigial, a remnant of an earlier epoch in the evolution of computing. These differences are part of Emacs' charm and its power. How you use Emacs now will also be the way you use Emacs in decades to come. Reading the 1981 Emacs manual is almost like reading the most recent version, as the underlying basic functionality has changed only slightly (Stallman, 1981a).

Now that you have cleared the first hurdle, it is time to open Emacs and look around. When you first start Emacs, you see a splash screen with links to help files (Figure 2.1). Click on any of the links to read the tutorial, or press the `q` button to close (kill in Emacs-speak) the screen. Pressing `q` is the standard method to kill read-only screens in Emacs. When the splash screen closes, you enter the 'scratch buffer', which you can use for temporary notes. Emacs does not save the content of the scratch buffer, so don't start writing your dissertation just yet.

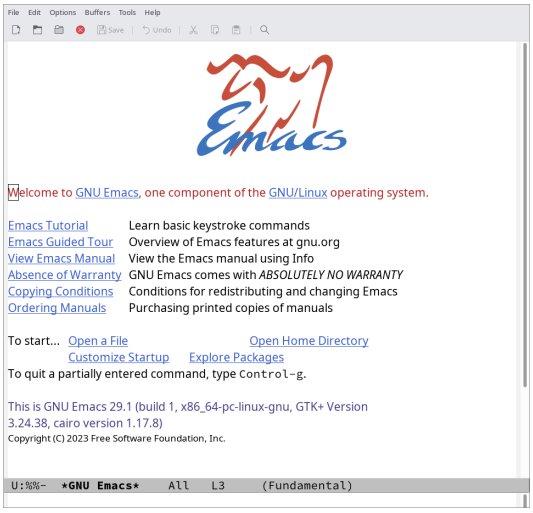

FIGURE 2.1 Emacs 29.1 splash screen.

2.1 Working with the Keyboard

Emacs has a mouse-driven menu system with drop-down menus on the top of the screen. Most users ignore this convenience and rely on keyboard-driven instructions. There is no agreement on whether using a keyboard or a mouse is most efficient (Tognazzini, 1992; Omanson et al., 2010). As visual creatures, clicking on an icon in a menu bar requires less brain capacity than remembering sequences of keystrokes. However, the problem with icon bars is that there is simply insufficient space to display icons for all functionality. Keyboard shortcuts are easy to remember as they become part of your muscle memory. Also, regularly moving your hands between the keyboard and the mouse can be annoying and impede your workflow. When misspelling a word in a word processor, you move your hand from the keyboard to the mouse, click on the offending word and select the desired spelling. In Emacs, you use one keystroke to change the typo word to the most likely correct version and keep writing. The most important thing to remember in the keyboard versus mouse debate is that writing is more about thinking than words per minute. Of course, a mouse has advantages. You can use it in Emacs for some tasks, like selecting text or moving the cursor. Vanilla Emacs also contains dropdown menus and a toolbar for mouse usage. The main advantage of the menu system is that it helps discover functionality in Emacs, but you don't need a mouse for this. Press `F10` and use the arrow keys to navigate the dropdown menu.

A standard computer keyboard has five types of keys:

1. Alphanumeric / punctuation
2. Editing keys
3. Function and multimedia keys
4. Escape key
5. Modifier keys

You describe these below but to reduce reader's head scratching, one or two examples per point (except ESC) could be nice.

Alphanumeric and punctuation keys activate the `self-insert` function, which adds text to the computer's memory (the buffer) and displays it on the screen. Editing keys, such as arrow keys, page up and down, delete, backspace, enter and Tab, do what it says on their label. Function and multimedia keys perform complex tasks. For example, pressing `F10` shows the menu bar. Multimedia keys are usually bound to the operating system and undertake tasks such as increasing the screen brightness or playing music. The escape key is the most potent key on the board. Like Dorothy's Ruby Slippers in the *Wizard of Oz*, it gets you out of trouble when you are stuck by clicking it three times.

Wow, I thought those keys enter the character on them into the text :-) You're deep emacs speak here. The word "function" appears for the first time here. Maybe translate this for normal people.

typo?

In principle, these are the only keys you ever need to write prose, but we like to do more than insert text. Computer keyboards also have modifier keys, which are special keys that temporarily modify the standard action of another key when pressed together.

The modifier keys on modern PC or Apple keyboards are Shift (and Caps Lock), Control, Alt / Option, and Windows / Command. Chromebook computers don't have an equivalent to the Windows/Command key. Some smaller keyboards also have additional modifier keys, such as `Fn`, to expand the available options. Modifier keys have no effect when pressed by themselves. As the name suggests, these keys modify other keys when pressed simultaneously.

Emacs documentation uses a special notation for modifier keys. Some of the Emacs terminology for these keys stems from a time when the current standard keyboard layout did not yet exist. What we now call the Alt key used to be the *Meta* key. The Windows key on PC keyboards or Command on Apple systems maps to the Super key, which was available on keyboards in the 1980s. Your operating system uses this key, so vanilla Emacs does not use it. There is also the *Hyper* modifier key, which no longer exists on modern keyboards, so it is no longer used but is still available as an Emacs modifier key.

...at the same time

Emacs documentation and this book abbreviate key sequences. For example, `C-a` stands for pressing the Control and a keys. The dash indicates that the first key modifies the second key. Each modifier key has its own letter, as shown in table 2.1. You can combine modifier keys, occasionally leading to awkward combinations, such as `C-M-S a` (Control, Alt and Shift `a`), which requires the nimble fingers of a sleight-of-hand artist to execute smoothly. The cover of the 1981 version of the Emacs manual even suggested that Emacs is best used by aliens with super flexible fingers (Figure 2.2). The shift modifier is often not used in Emacs notation because `C-M A` is the same as `C-M-S a`.

TABLE 2.1 Emacs modifier keys.

Modifier	Example	Function
Shift	`S-8`	`*` sign on US keyboard
Control	`C-e`	End of line
Alt / Option	`M-d`	Delete (kill) word
Windows / Command	`s`	Used by the operating system
Hyper	`H`	Not mapped to regular keys

The most critical shortcut is `C-g` (`keyboard-quit`), which cancels a partially typed command. Unlike the triple escape key, this command can also quit running functions.

But wait, there is more. Emacs also uses prefix keys. When you press these, the system will wait for further input. For example, `C-x C-f` means that you first press Control and `x` and then Control and `f`, the default sequence for finding (opening or creating) a file with the `find-file` function. A second after pressing a prefix key, Emacs displays it in the echo area, awaiting further input. The length of key sequences is unlimited, but they are usually at most three or four input events in practice. The standard prefix keys are:

- `C-x`: Mostly used for built-in Emacs commands
- `C-c`: Mostly used by packages
- `C-h`: Help functions
- `M-x`: Execute commands

Some packages also use prefix keys. *Emacs Writing Studio* uses `C-c w` as a prefix to store all its keybindings. This means that you can group key bindings for easy memorisation.

The Emacs keyboard shortcuts can be complex to enter. To repeat a command and spare you the manual gymnastics use the `repeat` function, which is bound to `C-x z`.

Due to Emacs's age, it does not comply with the Common User Access (CUA) standard for user interfaces, which was first developed in 1987 (Berry, 1988). This standard defines the familiar keyboard shortcuts such as `C-c` and `C-x` to copy or cut something to the clipboard. Emacs uses these as prefix keys, which would create conflict. Other standard keys, such as `C-z`, are already used for different functionality. You can configure Emacs to recognise these common keyboard shortcuts by enabling CUA Mode, but *Emacs Writing Studio* (EWS) sticks to the Emacs shortcuts.

If you want to find out which command binds a specific shortcut, use `C-h k` and enter the key sequence. A popup window describes the relevant function. Press `q` to kill this window. Emacs displays a message at the bottom of the screen when you enter a key sequence that has no associated function, e.g., "`C-c k` is undefined." You also see this message when you activate an undefined sequence without the help function.

One more prefix key needs mentioning. Some commands have alternative states, meaning the same function can be used in multiple ways. You activate an alternative state by adding `C-u` (the universal argument) before the regular key sequence.

Emacs repeats the function four times when a function does not have an alternative state for the

universal argument. So using `C-u <up>` moves the cursor four lines up. Using a double universal argument makes it sixteen, and so on. So typing `C-u C-u C-u #` Emacs inserts sixty-four hashtag symbols. You can also repeat keystrokes by adding a number after `Control` or `Alt` repeats the next keystroke. For example `M-80 *` adds eighty asterisks to your text.

This detailed description of how Emacs uses the keyboard might dazzle you. Don't worry, though. By the time you complete this book, you will gradually understand its intricacies and drive the system like a virtuoso.

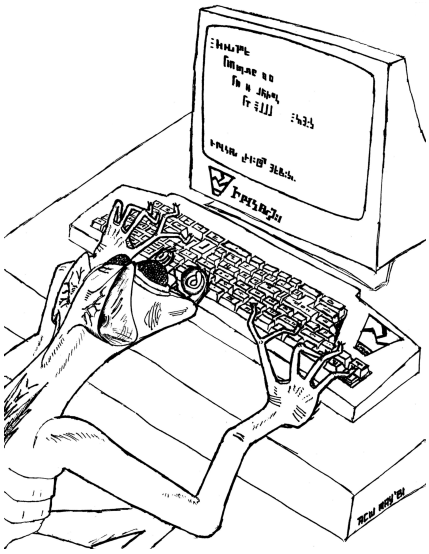

FIGURE 2.2 Cover of the 1981 version of the Emacs manual.

2.2 Issuing Commands

Emacs is largely written in the Emacs Lisp (Elisp) language, which means that almost every action activates an Elisp function. Most functions are not directly visible to the user. The functions that are interactively available to the user are the commands. Every command is a function, but not every function is a command.

The default way to execute commands is to use `M-x` and then type the command name and the Return/Enter key (`RET`). When you type `M-X`, the bottom of the screen (the minibuffer) shows `M-x`, waiting for further instructions. The minibuffer is where you enter input and instructions. For example, type `ALT-x tetris RET` to play Tetris (`RET` stands for the Return or Enter key). Don't get too distracted; just press `q` a few times to exit the game and get back to the book.

Typing the full function name every time is too much work for those who seek ultimate efficiency. The minibuffer completion system helps you find the commands you seek. When typing a partial function or file name, you can hit the `TAB` key. Emacs will display completion candidates in the minibuffer. For example, to evaluate the `visual-line-mode` function to change how Emacs wraps paragraphs, you type `M-x visu` and `TAB`. Suppose you hit `TAB` after each letter. In that case, you'll notice that Emacs narrows the completion candidates as you get closer to your desired selection. This principle also works with variable names and filenames. As you start looking for a filename, use the `TAB` key to prevent having to type the whole filename. The `TAB` key is your se-

cret weapon to help you remember and discover functions, variables, file names, buffer names and other selection candidates. But that method of issuing commands is not on the efficiency frontier, so Emacs extensively uses keyboard shortcuts. The remainder of this book will just mention the names of commands without adding the `ALT-x` and `RET` parts.

M-x

2.3 Major and Minor Modes

Emacs is a versatile tool that accomplishes specialised tasks through editing modes that usefully alter it's basic behaviour. Editing modes can be either major or minor modes. An editing mode provides major and minor modes. A major mode is like opening an app within the Emacs environment. For example, Org mode provides a task management system and publication tools. Artist mode is a quirky tool in Emacs that allows you to create plain text drawings with the mouse and keyboard. A major mode determines the core functionality for an open buffer. Each buffer has at least one major mode, and each major mode has its own functionality and key bindings. All major modes share the same underlying Emacs functionality, such as copying and pasting (killing and yanking) and opening files, but they add specialised functions.

You use that word before without explanation. Maybe latest here provide some more info. The full explanation comes only later.

Minor modes provide further additional functionality, such as spell-checking, text completion or displaying line numbers. A minor mode is an auxiliary program that enhances the functionality of a major mode. While each buffer has only one major mode, a buffer can have many active minor modes. A minor mode can also apply to the whole Emacs session.

Emacs automatically selects the relevant major mode using the file's extension and displays it in the mode line below the window. Minor modes have to be explicitly enabled, either globally or hooked to a specific major mode.

The available keyboard shortcuts depend on the major and minor modes that are active at the time. Each mode can have its own keymap. Some keymaps are global and apply to the whole of Emacs. Other maps are specific to a mode. Unless a mode overrides this binding, some shortcuts remain the same for all modes (such as `M-u`, which converts a word to uppercase). Packages can change or add shortcuts, depending on the required functionality. So, a shortcut like `C-c C-c` is used by different modes for different actions, depending on the context.

If you are ever lost and want to see all possible keyboard shortcuts for the active buffer (screen), type `C-h b` to view a list of all available shortcuts. Note that some shortcuts only relate to the currently active modes.

2.4 Finding Files

Opening files in Emacs is called 'visiting a file' and occurs with the `find-file` function (`C-x C-f`). Emacs opens the file and displays its contents in the buffer. When you type a name that does not yet exist, Emacs creates a new file. If you open a directory, Emacs shows the contents of that folder in the Emacs file manager (The Directory Editor or Dired, Chapter).

typo?
(I am not a native speaker so this may well be just right.)

Emacs asks you to nominate a file or folder in the mini buffer. When you hit the `TAB` button twice, all the available files and folders appear in the mini buffer. To complete the file name, start typing the filename and hit `TAB` again. After you complete your edits, `C-x C-s` saves your buffer. To save a buffer under a new name, you can use `C-x C-w` (Table 2.2).

TABLE 2.2 Most commonly used file functions.

Keystroke	Function	Description
`C-x C-f`	`find-file`	Find (open) a file
`C-x C-s`	`save-buffer`	Save the current buffer to its file
`C-x C-w`	`write-file`	Write current buffer to a file (Save as)

2.5 Buffers, Frames and Windows

When you open Emacs, the software runs within a frame (Figure 2.3). This might sound confusing because a frame is called a window in most operating systems. To confuse matters further, you can divide an Emacs frame into windows. You can also open multiple frames on a desktop, for example, one on each monitor.

The default Emacs screen has a dropdown menus on top and toolbar with icons just below it. The window starts below the toolbar. Each window contains a buffer, which contains the text. Buffers can also contain a user interface or output from functions. The mode line below the window diplomats the name of the buffer or its associated file. Special buffers, such as `*Messages*`, are surrounded by asterisks. Most buffers, except those surrounded by an asterisk, are linked to a file. Like standard office software, you are working on the version in memory (the buffer), and the previous version is on disk (the file). You can have multiple buffers open at the same time so that you can easily switch between them. Emacs is highly stable, and some users have hundreds of open buffers because they rarely need to restart the program. The active buffer is the one you are currently working on. The `C-x b` shortcut (`switch-to-buffer`) selects another buffer as the active one. With the `C-x left` and `C-x right` key sequences (`previous-buffer` and `next-buffer`), you can move between buffers in chronological activation order.

shows

diplomat (?)

buffer names

Each frame has an echo area at the bottom, where Emacs displays feedback. Echo is a computer science term for a local display of information, such as error messages and other feedback. The bottom of the page also contains the minibuffer, where users add input when prompted. The modeline is a bar at the bottom of each window. This line displays metadata about the buffer and the system (Figure 2.3).

By default, a frame has one window. You can duplicate the current window horizontally or vertically by pressing `C-x 2` or `C-x 3` (`split-window-below` and `split-window-right`). The `C-x 0` shortcut (`delete-window`) removes your current window but the buffer stays in memory, and `C-x 1` removes all other windows (`delete-other-windows`), so you work in the full frame again. To move between windows, use the `C-x o` shortcut (`other-window`). This function cycles through the available windows.

split

When splitting a window vertically, the same buffer appears twice. Each window can have its own cursor position so you can easily refer to other parts of your writing without jumping around. Activating `follow-mode` flows the text so the two or more windows become columns of the same document. This function toggles follow mode, so you can switch it on and off with the same function.

...with M-x follow-mode

I was confused because I think this descriptions fits normal C-x 3 too. I had to try to understand it

The bottom of each window has a mode line, and the bottom of the frame contains the echo area and mini buffer. The mode line provides information, such as the open file's name, the current line number and other helpful information. The echo area and mini buffer occupy the same space. The echo area is a single line where Emacs displays feedback. The minibuffer is an expandable part of the bottom of the screen where Emacs seeks your input when, for example, selecting a buffer or a file.

TABLE 2.3 Buffer and window functions.

Keystroke	Function	Description
`C-x b`	`switch-to-buffer`	Select another buffer
`C-x <left>`	`previous-buffer`	Move to the previous active buffer
`C-x <right>`	`next-buffer`	Move to the next active buffer
`C-x 0`	`delete-window`	Delete the current window
`C-x 1`	`delete-other-windows`	Delete all windows except the active one
`C-x 2`	`split-window-below`	Split the current window horizontally
`C-x 3`	`split-window-right`	Split the current window vertically
`C-x o`	`other-window`	Move to the next window

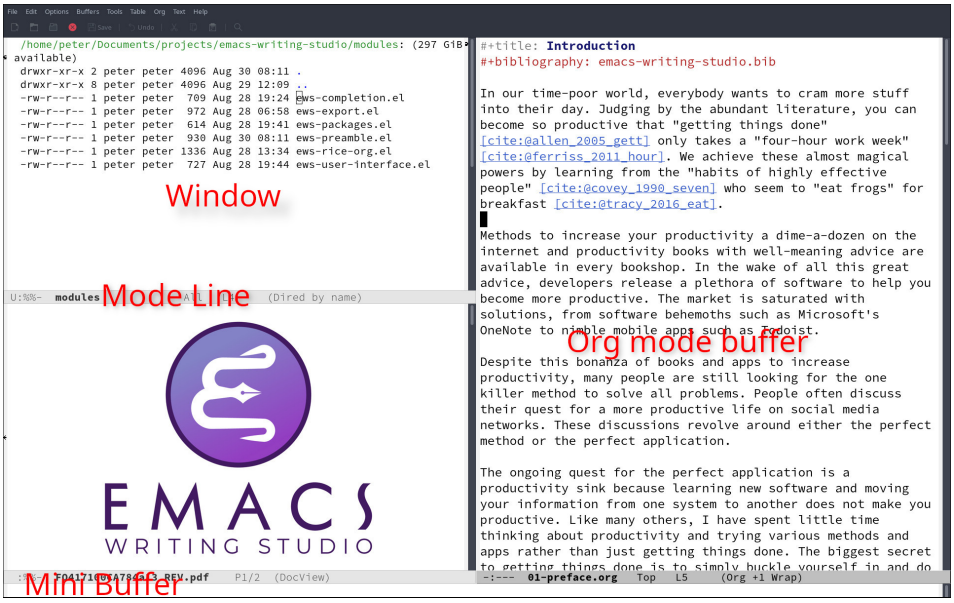

FIGURE 2.3 Emacs frame with three windows, a Dired buffer, image buffer and Org mode buffer.

2.6 Writing in Emacs

You now know enough to start writing. Either visit an existing plain text file or create a new one and start typing. But to be fully productive, you need to understand some of the basic principles of Text Mode, the foundational major mode for writing prose.

The Emacs documentation describes Text Mode as the mode for writing text for humans, in contrast to Prog Mode, which is for writing code that computers read. Text mode forms the foundation for all other prose formats. This means that all major modes for authors use the same basic functionality for writing.

This section summarises the most common commands for writing text. Chapter 8 and 26 of the Emacs manual provide a detailed description of all functionality for writing human languages (as opposed to computer languages), which you can read with `C-h r`.

2.6.1 Moving Around in a Buffer

the point

You can move the cursor with arrow keys and other standard navigation keys. Emacs documentation sometimes refers to the cursor as a point. The cursor is the character displayed on the screen (a line or a box), and the point indicates where the next typed character will appear. Point is more critical when you write Emacs functions, so this book focuses on the cursor, as that is where the writing action happens.

In addition to the standard methods for moving around a buffer, Emacs provides additional functionality to help you navigate your project. For example, `C-p` (`previous-line`) does the same as the up key (see Table 2.4). Some people prefer these keys so their hands can stay in the default position for fast touch-typing. However, writing is more about thinking than maximising keystrokes per minute, so feel free to try them out.

TABLE 2.4 Moving around a buffer in Emacs.

Keystroke	Function	Direction
`C-b`, `<left>`	`left-char`	Left
`C-f`, `<right>`	`right-char`	Right
`C-p`, `<up>`	`previous-line`	Up
`C-n`, `<down>`	`next-line`	Down
`M-b`, `C-<left>`	`backwar-word`	Previous word
`M-f`, `C-<right>`	`forward-word`	Next word
`C-v`, `<PageDown`	`scroll-down-command`	Scroll down
`M-v`, `<PageUp>`	`scroll-up-command`	Scroll up
`C-a`, `<home>`	`move-beginning-of-line`	Start of line
`C-e`, `<end>`	`move-end-of-line`	End of line
`M-<`, `C-<home>`	`beginning-of-buffer`	Start of buffer
`M->`, `C-<end>`	`end-of-buffer`	End of buffer

Getting lost in a sea of words on your screen is easy. Some simple keystrokes can help you focus your eyes quickly. Keying `C-l` (`recenter-top-bottom`) moves the line that your cursor is on to the centre of the screen. If you repeat this keystroke, the cursor will move to the top of the screen. If you do this three times in a row, the cursor will move to the bottom of the screen.

You will undoubtedly experience moving from one part of a document to another and then like to jump back but lose your place. You search through the document to get back to where you left off. You can do this more efficiently by setting a mark. A mark is a bookmark for a position (point) within your text. Setting a mark is like dropping a pin on a map. You can set a mark to remember a place you want to jump to, which is incredibly handy when editing large files. You set a mark with `C-SPC C-SPC` (`set-mark-command`), which stores the cursor's current location in the mark ring. The mark ring is the sequence of marks for the current buffer. You can now move to another part of your document and edit or read what you need. You jump back to the previous mark with `C-u C-SPC`. While `C-SPC` (`set-mark`) stores the current location in the mark ring, adding a universal argument extracts that position and jumps to it. Repeatedly pressing `C-u C-SPC` moves through all the marks stored in the ring. If you get to the first stored value, you return to the last one, hence the name mark ring.

2.6.2 Search and Replace

While jumping around the text with arrow keys and other functionality is great, sometimes you know exactly when you need, which is when you search. The search and replace functionality in Emacs is extremely powerful and this section only reveals the tip of the iceberg.

Emacs' most common search method is incremental search (`isearch-*`). An incremental search

(C-s) begins as soon as you type the first character of the search term. As you type in the search query, Emacs shows you where this sequence of characters is found. Repeatedly pressing C-s steps through the matches in the buffer. When you identify the place you want, you can terminate the search explicitly with ENTER or C-g. Big difference between C-g and RETURN and most other editing commands which also terminate.

The C-s shortcut (isearch-forward) searches incrementally from the cursor. You cycle through the search results by repeatedly pressing C-s. Using C-r (isearch-backward) searches the text before the cursor. Emacs saves search terms in the search ring. Typing C-s C-s will show the previous search term. Using C-p and C-n lets you scroll through previous search terms in the ring.

To search and replace text in a buffer, use M-% (query-replace). This function highlights all instances of the text to be replaced and provides a range of options at each instance. Type space or y to replace the marked match and delete or n to skip to the next one. The exclamation mark replaces all instances without further confirmation. If something goes wrong, use u to undo the most recent change or U to undo all changes made in this search. The Return/Enter key of q quits the replacement process. More options are available, which you can glean by hitting the question mark. or

2.6.3 Copy and Paste Text

Writing is fun, but sometimes it is more efficient to copy something you wrote previously or comp a citation from somebody else (referenced of course). The system for copying and pasting text works a bit different from modern system and has a bit more functionality. ?

To select (mark in Emacs speak) a piece of text, you first set a mark with C-space and then move to the end of the section to highlight the desired section. To select a complete paragraph, use the M-h key. Repeatedly pressing M-h will select subsequent sections. Using C-x h selects all text in a buffer, and C-g cancels the selection. Once the text is marked, you can act on it by deleting, copying, or moving it.

In some modes you can select with shift and arrow keys, but it is disabled in some modes because these key combinations are used for other functionality. Shift-selection also behaves differently with respect the mark ring describes in the previous section.

In modern computing language, copying and pasting are handicraft analogues for moving text from one place to another. Emacs terminology is more evocative. Copying a text is the same as saving it to the 'kill-ring' and yanking a text retrieves it from that seemingly bleak location. While the clipboard in most systems only retains the last entry, the kill ring provides access to your 'killing spree' history. In other words, Emacs stores a history of all text you copy and cut from a buffer to the kill ring. The length of this history is stored in kill-ring-max, which is sixty entries by default. Once the kill ring is full, the oldest item vanishes.

The kill* commands copy or move text to the kill ring and the system clipboard. The yank* commands copy an entry from the kill ring to the current buffer. The yank-pop (M-y) command cycles through the contents of the kill ring so you can access the history. Use the keyboard shortcuts in Table 2.5 to copy and move text from and to the kill ring.

TABLE 2.5 Copying and pasting in Emacs.

Keystroke	Function	Description
M-w	kill-ring-save	Copy a selection to the kill ring
C-W	kill-region	Move a selection to the kill ring
C-y	yank	Paste the most recent entry in the kill ring to the buffer
M-y	yank-pop	Replace the previously yanked text with the next kill ring entry

2.6.4 Correcting Mistakes

An ancient Roman proverb tells us that it is human to make mistakes, bu to keep making them is diabolical. Emacs does not care about these sensibilities and provides ample options to let you correct your mistakes.

but

The most convenient aspect of writing on an electric screen is that it is easy to change your mind or correct a mistake without resorting to correction fluids or other archaic methods. A series of editing commands are available to modify text and fix your typos (Table 2.6). Commands that start with `kill` store the deleted text on the kill ring so you can yank the deleted text back into the buffer if needed.

TABLE 2.6 Emacs deletion commands.

Keystroke	Function	Action
`C-d`, `<delete>`	`delete-char`	Delete character after point
`<backspace>`	`delete-backward-char`	Delete character before point
`C-x C-o`	`delete-blank-lines`	Remove blank lines below the cursor
`M-d`, `C-<delete>`	`kill-word`	Delete the next word
`C-k`	`kill-line`	Delete to the end of the line

Besides removing unwanted characters, you can also swap them with a series of transposing commands. When you accidentally reverse two letters in a word, you can switch their order with the `transpose-char` command with the cursor between them (`C-t`). Swapping words is quickly done with the `transpose-words` (`M-t`) command.

Emacs can assist you when you make a mistake when capitalising a word. The three commands below change the word under the cursor from its position. If you are in the middle of a word, move first to the start. Adding a negative argument (`M--`, ALT and the minus key) before these commands modifies the letters before the cursor. This addition is valuable when you have just finished typing a word and realise it needs to start with a capital letter. Typing `M-- M-c` fixes it for you without jumping around the text or grabbing a mouse. Using any of these commands in succession converts a sequence of words in a sentence.

- `M-l`: Convert following word to lower case (`downcase-word`).
- `M-u`: Convert following word to upper case (`upcase-word`).
- `M-c`: Capitalise the following word (`capitalize-word`).

The Emacs `undo` command is mapped to `C-/`. If you need to undo the step, use `C-?` (`undo-redo`). Emacs behaves differently from other software concerning undoing and redoing edits, which requires some explanation. In standard word processors, the text you undid is lost if you undo something and make some changes but then change your mind. For example, type “Socrates”, change it to “Plato”, and then undo this edit to revert back to Socrates and add more text. In standard word processors, you cannot move back to the state where the text mentioned Plato (State B in Figure 2.4). In Emacs, all previous states are available. You can always return to any prior state with consecutive undo commands in Emacs. Subsequent undo commands follow the chain in Figure 2.4, never losing anything you typed.

I think what describes it concisely is that Emacs undoes the undo or leaves the undo on the list of commands which can be undone. I have not seen this in any other editor. And even a savvy developer using Emacs once told me that this is the one thing in Emacs that he finds quite confusing.
I absolutely love it.

Lastly, there is also a nuclear undo option. You can discard all the changes since the buffer was last visited or saved with `M-x revert-buffer`, which reloads the file from the disk, erasing all edits since the last saving of the file.

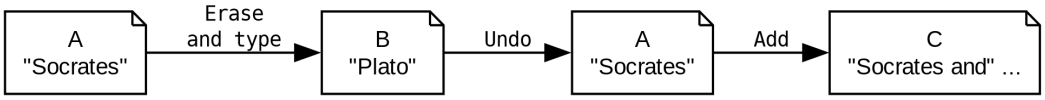

FIGURE 2.4 Emacs undo states.

2.6.5 Counting Words

A good author always tracks the word count. For this book, I aim to write between 5,000 and 10,000 words per chapter. To count the number of words in a highlighted part of the buffer, use `M-=` (`count-words-region`). If no selection is active, it counts the next near the cursor. This function displays the number of lines, sentences, words, and characters in the echo area. Adding the `C-u` prefix counts the words for the hole buffer (`C-u M-=`). Please note that Emacs defines a sentence as a sequence of characters that end with a full stop and two spaces, so if you use a single space, the number of sentences is incorrect. The `count-words` function, which has no keyboard shortcut, counts all word in the buffer or the marked region.

2.6.6 Modifying the Display

The way the buffer looks on the screen depends on the major mode, the theme, and specific configurations and packages. You do have some interactive control over the size of the text. To temporarily increase the height of the text in the current buffer, type `C-x C-+`. To decrease it, type = C-x C– = (text-scale-adjust). To restore the default (global) face height, type `C-x C-0`.

formatting

Everyone in the world calls it 'font' :-) May need a translation to Emacsish here.

The default Text Mode in Emacs does not truncate lines like a regular word processor but keeps going until you hit enter. In Emacs, a logical line is a paragraph separated by empty lines. A visual line relates to how it is displayed in Emacs. The default setting is that visual lines continue beyond the screen boundary. While this is useful for writing code, it is confusing when writing prose. Emacs has several line-wrapping functions, of which Visual Line Mode is the most useful for writing long-form text. To activate this mode, evaluate `visual-line-mode` in the mini buffer. We need to configure the system to change the line wrapping and modify other behaviours by default to make Emacs behave a bit more like a text processor.

execute (?)

2.7 Configuring Emacs

The previous sections explained how to use Emacs in its naked, unconfigured state, more commonly called vanilla Emacs. The software can do anything you need to be an author without any configuration, but that is not ideal. As a malleable system, Emacs is almost infinitely configurable, so you can make it behave how you see fit. Emacs users have shared their configurations and published thousands of packages to add functionality. This chapter discusses the principles of configuring Emacs and how to install the *Emacs Writing Studio* configuration.

Most software is good at doing one thing well. You can write documents in LibreOffice or MS Word. You can create presentation slides in PowerPoint or Keynote and manage tasks with Trello or Todoist. The problem with using different applications is using various software packages and other methods whenever you switch contexts in your workflow. If you are lucky, the developers let you change the configuration to modify the software's behaviour and optimise your workflow. However, in most cases, you are stuck with the choices that developers made for you. While using commercial software is like renting a fully-furnished house, using Emacs is more like owning a

house. However, your digital home needs some paint, new carpets, and furniture to make it your home.

Emacs does not have the limitations that are common to most software. You undertake almost every task in one program, and nearly everything in the system is configurable. This article explains how to configure Emacs to create a fully personalised productivity suite. Please note that this configuration assumes that you are using the latest version of Emacs, which at the time of writing is 29.3.

2.7.1 Configure Emacs with Starter Kits

Some Emacs users use pre-configured systems, such as Doom Emacs, Spacemacs, or other starter kits. While these configurations are helpful, they sometimes provide 'everything but the kitchen sink'. On the other side of the spectrum, you can configure your system from scratch, which can become a productivity sink as you wade your way through a myriad of options. The EWS configuration is a starter kit with a minimum configuration to get you started as an author. The basic idea is to use this configuration as a starting point and modify it to meet your preferences. But before installing the EWS configuration, let's first introduce the principles of configuring Emacs.

2.7.2 Emacs Lisp

Commercial software provides graphical menus to define how it operates. For example, in Figure 2.5, you might tick a box, select an item in a list, or enter a value in a text box to configure the program according to your wishes.

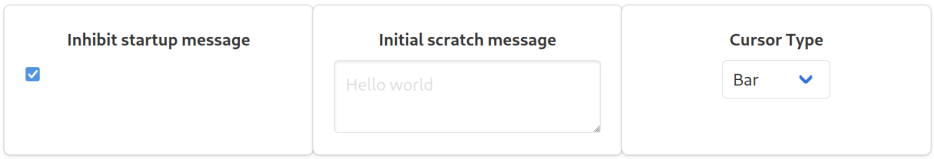

FIGURE 2.5 Typical graphical configuration screen.

Being a plain text program, Emacs does not have such facilities but uses the Emacs Lips programming language. The code below is equivalent to the form shown in Figure 2.5. Compare the code with the image to reverse engineer the Elisp code.

```
(setq inhibit-startup-message t
      initial-scratch-message "Hello world"
      cursor-type 'bar)
```

A Lisp program consists of expressions, which are instructions nested between parentheses. Each expression starts with the name of a function (`setq` in the example above). In most cases followed by one or more parameters. The `setq` function sets the value of a variable. For example, `(setq inhibit-startup-message t)` has the same effect as ticking a box called 'inhibit startup message', while `inhibit-startup-message nil` is the same as removing the tick from that box. In Emacs Lisp, `t` means the same as TRUE and `nil` is equivalent to FALSE in other computer languages. Confusingly Emacs documentation often mentions to set a value to non-nil, which is a double negative to suggest setting the variable to true. The expression in this example determines whether Emacs will show a startup message when you first open it. The second line sets the initial scratch message. The scratch buffer is a place to write temporary notes. In this case the parameter is a string, nested

between quotation marks. The last line sets the cursor type to a bar. This variable has various predefined options, such as bar or hollow. To prevent Emacs from confusing this word with another variable, it uses the single quotation mark (also called a tick mark) before the text.

If you like to know more about these variables, the use `C-h v` and the variable name. The help documentation explains what the variable does and in some cases the available options.

While on the surface, the text-based method seems more complex than ticking and writing in boxes and picking a dropdown list, it is far more potent than a graphical interface. However, once you learn how to write simple Emacs Lisp, you will realise that Emacs is, in reality, the most user-friendly system possible because of the power it gives you over your computer. Using Emacs Lisp is the epitome of user-friendliness. You decide how your computer behaves instead of some software company controlling your behaviour. But with this immense power comes great responsibility and a learning curve.

Section 2.6.6 showed how to modify how Emacs wraps long lines by activating `visual-line-mode`. The code snippet below shows what this would look like in your init file. In this case, we hook visual line mode to text mode. All modes derived from Text Mode, such as Org Mode or Markdown, will inherit this property. The line that starts with two semi-colons is a comment intended to render the code easier to read and navigate.

```
;; Sensible line-breaking
(add-hook 'text-mode-hook 'visual-line-mode)
```

2.7.3 The Initialisation File

When you start Emacs, it loads the initialisation file, or init file in short. This file contains Lisp code that loads additional packages and configures when Emacs starts. You can run Emacs without an init file, but you will undoubtedly want to change the defaults. The first time you start Emacs, it will create the configuration folder. The init file lives in this folder, which also contains the packages you need to personalise your system. Emacs looks for a file called `.emacs`, `.emacs.el` or `init.el`. The dot in front of the file means that it is hidden from view to prevent clutter in your directories. Most Emacs documentation talks about your `.emacs` file.

2.7.4 Customisation System

Besides crafting your personal configuration or using a starter kit, Emacs has a customisation menu that lets you configure Emacs without writing code. When you make changes using the customisation functionality, the relevant Emacs Lisp code will be written in the init file. For example, if you want to remove the toolbar from view, you type `M-x customise-variable RET tool-bar-mode RET`. A new window pops up with the customisation options for this variable (Figure 2.6). This variable is a boolean, meaning it can be either true (`t`) or false (`nil`). You can change the state of this variable with the toggle button. The 'Apply' button brings this change to immediate effect. When you click 'Apply and Save', the new value is saved to the init file, which looks something like this:

Stallman is American :-) s->z

```
(custom-set-variables
 ;; custom-set-variables was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(tool-bar-mode nil))
```

The customisation screen depends on the type of variable. The input can be a boolean (true or false), a string or a number, a dropdown box, a Lisp expression and some other variable types.

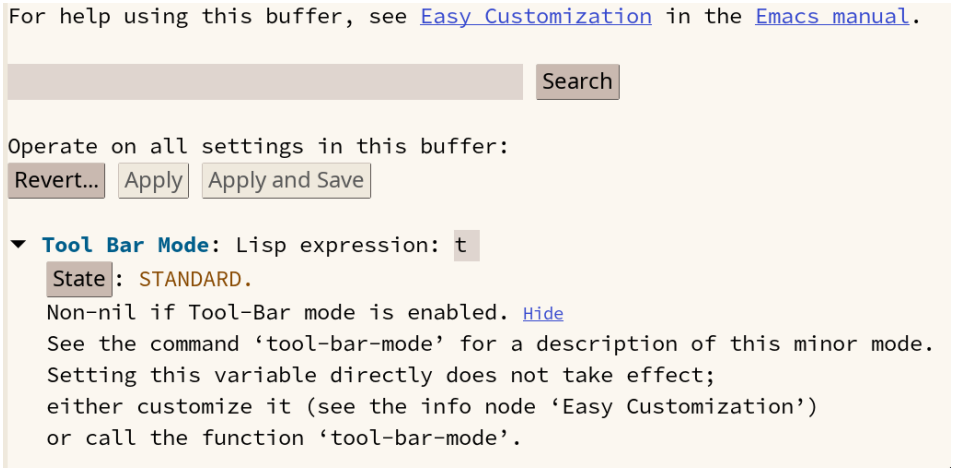

FIGURE 2.6 Customisation screen for `tool-bar-mode`.

2.7.5 Emacs Packages

The Emacs base system provides extensive functionality, but you can enhance its capability with any of the thousands of external packages. Many people develop and share packages in Emacs Lisp to improve or extend what the system can do. Developers of these packages mostly distribute them through a public package repository, which are websites that let you easily download and install packages. The two most important ones are:

- ELPA: GNU Emacs Lisp Package Archive — the official package archive, enabled by default (`elpa.gnu.org`).
- MELPA: Milkypostman's Emacs Lisp Package Archive — Unofficial archive (`melpa.org`).

The differences between these two repositories relate to who holds the copyright. In ELPA packages, the Free Software Foundation holds the copyright. For MELPA packages, the copyright remains with the author. The end result for the user is the same as all packages are licensed as free software. You can explore the list of packages with the `list-packages` function.

Packages are constantly updated by their developers. To ensure you get the latest version, use the `package-upgrade-all` function, which has no default keybinding. This naming convention might seem back to front, and using `upgrade-all-packages` seems more linguistically sound. However, the convention for naming Emacs Lisp functions is that the first word is the package name, which in this case is package.

The easiest method to install packages is with the Use-Package macro, which downloads, installs and configures a package. The example below downloads and installs the Dracula theme, a popular colour theme available for hundreds of software packages. The first line identifies the package. The `:ensure t` in the second line ensures that Emacs downloads the package if not already available. Any code under `:config` will run when the package is activated. In this case, we load the theme without asking for confirmation.

```
(use-package dracula-theme
  :ensure t
  :config
  (load-theme 'dracula :no-confirm))
```

These are the basic principles of installing and configuring packages. The Use-Package methodology has many more options, as discussed in the Appendix.

2.7.6 Modifying Key Sequences

Emacs ships with a range of predefined keyboard shortcuts for its core functionality and the built-in packages. Most external packages don't define key keyboard shortcuts to prevent conflicts with your configuration.

You can change the keyboard's behaviour at three levels: programmable keyboards, the operating system/window manager, and Emacs.

Some high-end keyboards are programmable and let you define the output of each key. For example, you could map the right control key as the Hyper key. At the second level, your operating system interprets the input from the keyboard. In Windows, `s-E` (Windows and E) opens the file explorer. You can erase this binding to make it available in Emacs. Each operating system has its own methods to change keyboard maps (keymaps). Some experienced Emacs users remap the caps lock key to act as the control key to make it easier to use.

Last but not least, you can define key sequences within Emacs itself. The example below binds `C-t` to toggling whitespace mode. This minor mode indicates whitespace in the current buffer with characters. The `#'` characters before the function name are a technical requirement to instruct Emacs not to evaluate this function but only to store its value. If you like to unset a keystroke, just use `nil` as the function.

Uff, here we have Emacs 29 stuff.As mentioned earlier, having the very latest maybe som

```
(keymap-global-set "C-t" #'whitespace-mode)
```

This example uses the global keymap, meaning the shortcut is available in all modes. You can also define a shortcut for a specific mode, which is only available when that mode is active. The example below sets the same shortcut but only applies when Org mode is active.

```
(keymap-set org-mode-map "C-t" #'whitespace-mode)
```

Some people don't like the Emacs keyboard defaults much because they require frequent use of the modifier keys. These people suggest that repetitive use of these keys causes strain injury, the dreaded' Emacs pinky'. Several packages, such as Evil Mode and God Mode, exist within the Emacs ecosystem that change the default keybindings to a different model. *Emacs Writing Studio* follows the standard conventions and does not modify default keybindings.

2.7.7 Installing *Emacs Writing Studio*

You don't have to learn to program in Elisp to configure Emacs. You can start with the *Emacs Writing Studio* configuration to get you going. Many Emacs users share their configuration so that you can copy parts of their code as an active learning experience. The copy-paste method is a shortcut to learn more about Emacs Lisp, as you can change the settings and explore the results of your changes.

To install the EWS configuration, download the init file from the GitHub repository (`github.com/pprevos/emacs-writing-studio`) and save it in the configuration folder. The location of the configuration folder depends on your operating system and Emacs version. Type `C-h v` to obtain help about variables, and type `user-emacs-directory`. The popup help buffer provides the correct folder name. You can close this buffer by pressing `q`. Copy the `init.el` file from the EWS repository to this directory. The Appendix to this book provides a detailed explanation of the EWS configuration. Your init file will become active after you evaluate the `restart-emacs` function or the next time you start the program.

Hmm, anyone having done minimal config in their Emacs will overwrite this. You could say, if they have done so, they know it, but if someone only ever did a customize-... they might not be really aware that they overwrite those with your init.el.

2.8 Exiting Emacs

After your first practice round, it is time to close (kill) Emacs. The `C-x C-c` kills the Emacs session, checking for unsaved buffers first (`save-buffers-kill-terminal`). There are a few options to ensure you don't lose anything when you have unsaved buffers. This function displays any unsaved files in the echo area and provides some options for dealing with it. You can answer `y` or `SPC` to save the file mentioned in the echo area or `n` / `DEL` not to save it. Keying `C-r` lets you look at the buffer in question before deciding. If you would like to know the difference between the open buffer and the saved file, use `d` to view the differences. The safest option is to key `!` and save all buffers that have changes without any further questions. Use the trusted `C-g` chord to exit this function without existing Emacs. Don't stress if you can't remember all this. Using `C-h` displays a help message describing these options.

2.9 Learn More

This chapter only discussed the basic principles of using Emacs. If you want to know more, follow the built-in Emacs tutorial, which you activate with `C-h t`.

The Emacs help system contains a voluminous library of extensive manuals for Emacs itself and some packages. You access these manuals with `info-display-manual` system (`C-h R`) and choose manual. To directly jump to the Emacs manual, use `info-emacs-manual` (`C-h r`).

When reading a manual in the info system, the space bar scrolls the screen up so you can walk through the manual and read it page by page (`info-scroll-up`). The backspace button returns you to the previous screen (`info-scroll-down`). The manual contains hyperlinks in the table of contents and sprinkled throughout the text. You can click these with the mouse or hit the enter key. To jump to the previous or the next chapter, you can use `info-up` and `info-down` functions bound to `u` and `d`. The Emacs manual is not the type of literature you like to read from cover to cover. The `Info-goto-node` function, bound to the `g` key, lets you search for the relevant section. You can start typing the name of a chapter and hit the tab key using the standard minibuffer completion. If you don't quite know where to look then `Info-search` (`s`) lets you search for specific terms.

3

Using Emacs Writing Studio

The previous chapter described how to use and configure vanilla Emacs to make it behave as you want. The *Emacs Writing Studio* (EWS) configuration converts vanilla Emacs to a bespoke tool for authors. This chapter explains how to use the additional functionality that the *Emacs Writing Studio* configuration provides and introduces a workflow from ideation to publication.

EWS uses a minimalist interface without any typical graphical software elements. This austere look minimises distractions from your screen so you can focus on what's important—writing words into a buffer. The second change to vanilla Emacs is the completion system. Emacs Writing Studio uses the Vertico / Orderless and Marginalia stack to provide enhanced completion in the minibuffer, making it easier to find functions, files, and other stuff you need.

The EWS configuration is an opinionated set of choices and might not suit everybody. Within the Emacs world, there is sometimes discussion about what constitutes a sensible default configuration. However, such a state does not exist. One person's sensible default is another computing nightmare, so feel free to change anything in EWS to suit your ideal worldview. The Appendix contains the complete annotated configuration.

3.1 Minimalist Interface

A minimalist interface means that your Emacs becomes a place of rest and contemplation away from the cacophony of contemporary software filled with buttons and functionality you don't need. The default Emacs user interface is a hybrid between keyboard and mouse-driven operations. While using the mouse might seem convenient, a keyboard-driven system is more efficient because you don't have to move your hand that much and switch contexts when finding icons.

This configuration removes the toolbar, menu bar and scroll bars. While drop-down menus are a valuable tool to discover functionality, there is no need to constantly keep them on the screen. You can access the menu with `F10` (`menu-bar-open`) and select menu options with the arrow keys and `RET` to choose an item. You exit the menu with `C-g` (`keyboard-quit`). But after using Emacs for a while, you'll quickly build muscle memory and revert to keyboard shortcuts. Note that the menu items show the relevant keyboard shortcuts when available.

This configuration also installs the Spacious Padding package by Protesilaos (Prot) Stavrou, which increases the margins around the text so the screen is not too cramped with symbols.

3.1.1 Themes

A theme is a set of instructions describing the colours of defined text parts. Colours in a text editor play a different role than in a word processor. Colours in Emacs are semantic, meaning they indicate the type of text, not how it looks when published. A heading might have a different colour than the text or metadata, which helps you find your way through the document.

Two basic classes of themes exist for text editors: light and dark. Light backgrounds, common with most modern word processing software, can cause asthenopia (eye strain) after you stare at the

screen for a while. Dark colour schemes increase visual acuity and reduce visual fatigue, especially in low-light physical environments with complex backgrounds (Kim et al., 2019). Many text editor users, therefore, prefer dark themes. Light themes are not bad as they are effective when you work in a brightly lit room.

The EWS configuration installs and activates the most recent version of Prot's Modus themes. The Modus themes have two primary versions: the `modus-operandi` theme is the primary light theme, while the `modus-vivendi` theme is its dark counterpart. The primary Modus themes maximise contrast between background and foreground following the Web Content Accessibility Guidelines (WCAG). The Modus themes comply with the triple-A standard of the WCAG, which specifies a contrast ratio between background and foreground of 7:1. This high contrast ratio is legible for people with moderately low vision. Each of the primary themes has three modified versions: two versions for red-green and blue-yellow colour blindness (deuteranopia and tritanopia) and a more colourful variety (tinted).

Emacs Writing Studio uses the tinted versions as default. These versions have a slightly lower contrast ratio and are suitable for people with normal vision. The Modus themes do not prescribe keyboard shortcuts, so EWS defines some. The `C-c w t` shortcut toggles between the light and dark side (insert Star Wars pun here). Using `C-c w T` provides a selection menu of all Modus themes. *Emacs Writing Studio* uses `C-c w` as its default prefix key for its specific functionality, where the `w` is a mnemonic for writing and `t` for themes.

If you would like your configuration to default to the high-contrast versions or one of the two colour blindness-safe versions, customise the `modus-themes-to-toggle` variable in the init file to override this default. Refer to the Appendix for details.

The Modus Themes package includes an extensive manual that explains in detail how to customise the look and feel of its collection of themes. This manual is available through Info Mode with `C-h i R modus ENTER`.

Vanilla Emacs includes various themes, which you can activate with the `customise-themes` function. When you click the *Save Theme Settings* button, the theme is added to the `custom.el` file. You must remove the Modus themes code in the init file to ensure your new theme is not overridden. When a theme is used for the first time, Emacs might ask you to confirm that it can run Elisp code. Just say yes and keep this setting for future sessions. s->z

Emacs users have developed a ragtag collection of themes. To pick your favourite, you can browse the Emacs Themes Gallery (`emacsthemes.com`). If the theme is available in the ELPA or MELPA package repositories, you can install it with a Use-Package declaration, as explained in section 2.7.5.

3.1.2 Setting Fonts

The default font in Emacs is a fixed-pitch (mono-spaced) font designed for writing code. In a fixed-pitch font, all characters have the same width. An `i` or an `m` will use the same amount of space, just like the traditional typewriters. This type of letter is ideal when writing code because it helps to align the text. Mono-spaced fonts are also necessary to create old-school ASCII art.

A variable-pitch font is easier on the eye when writing prose. Not all characters have the same width in a variable-pitch font, as is common in natural writing. Ideally, we want the best of both worlds and configure Emacs to use the most suitable font for each situation. Emacs can define a different font for certain parts of the text, for individual buffers, or for a major mode.

The EWS configuration uses Alex Branham's Mixed-Pitch package, which assigns fixed and variable pitch fonts to the relevant elements in the buffer. For example, tables in Org mode become fixed pitch so that all columns are the same size.

The *Emacs Writing Studio* configuration does not specify any particular fonts and uses your system's defaults. You can configure your favourite fonts, provided they are available on your computer. You need to define three font variables:

- `default`: The default face settings (a fixed-pitch font).
- `fixed-pitch`: The font used for computer code.
- `variable-pitch`: The settings for prose.

In Emacs, a 'face' is a collection of attributes to display text. It defines the font, foreground colour, background colour, optional underlining, etc. Various face attributes are available for configuration. The ones we use here are:

- `font`: The name of the font
- `height`: The font height as an integer in units of 1/10 point.

You can use the customisation menu by evoking `customise-face` and selecting `default`, `fixed-pich` or `variable-pitch` and entering the font name in the *Font Family* box. Click *Apply and Save* for each font. This action saves the font settings to the `custom.el` file, which Emacs evaluates at the beginning of the startup sequence. Please note that anything you configure here will override theme settings, so ideally, only customise font family and size.

is this new in 29, mine safes in init.el, or it is my historic setup from the last millenium :-)

To see which fonts are available, you can set the fonts in a graphical window by running `menu-set-font`. When you use the GUI to set the default font, the change is immediate but transient. Use `menu-bar-options-save` to save your default font and size to the `custom.el` file.

Alternatively, add the following three lines, with your fonts and sizes of choice, to your `init.el` file.

```
(set-face-attribute 'default nil :font "DejaVu Sans Mono" :height 130)
(set-face-attribute 'fixed-pitch nil :font "DejaVu Sans Mono")
(set-face-attribute 'variable-pitch nil :font "DejaVu Sans")
```

3.2 Minibuffer Completion

Even with the advent of speech-to-text software, the keyboard is still the most common method to convert thoughts to text. While computers might one day even read our minds, there is something to be said about using your fingers to do the talking. Who would want their 'ums' and 'ahs' or their uncensored stream of consciousness committed to text? Writing is as much about thinking and crafting a stream of words as it is about maximising keystrokes per minute.

Completion systems are like predictive text on a mobile phone. You start typing some characters, and the computer lets you complete your choice. Emacs has an extendable completion system that helps you complete long words, find files, remember function names and other menial tasks. This article explains the basic minibuffer completion engine and introduces some packages extending this functionality. Emacs has three types of completion systems:

1. *Minibuffer completion* assists with picking choices in the minibuffer, such as function names and files.
2. *Keychord completion*: Systems to help with keyboard shortcuts.
3. *Text completion* helps you complete words you type in the buffer (see Chapter).

The minibuffer is where you find files, evaluate functions, and enter other information. The minibuffer completion system aims to make it easier to find what you need by providing a search

mechanism that provides a list of possible options. The standard minibuffer Emacs completion system focuses on entering functions, filenames, buffer names and any other selection process in the minibuffer.

The minibuffer completion system is highly configurable, and several packages extend the vanilla functionality. The EWS configuration uses a stack of connected packages developed by Daniel Mendler that provide a seamless experience. This approach showcases the extensibility of Emacs as a modular computing environment.

The Vertico package enhances minibuffer completion. This extension uses incremental search, meaning the list of candidates is shortened to match your entry as soon as you type one or more characters. For example, when opening a file with `C-c C-f`, you can start typing any part of the filename to locate the file you seek. Use `C-backspace` keys to move to a higher directory.

The `savehist` package remembers your selections and saves your minibuffer history when exiting Emacs. This package ensures that your most popular choices remain on top for further convenience.

To further refine our ability to find completion candidates, the orderless package enhances Vertico and matches patterns, irrespective of the order in which they are typed. For example, typing `emacs writing TAB` provides the same results as `writing emacs TAB`.

Emacs is a self-documenting computing environment, meaning every function and variable includes a text describing what it does. The Marginala package displays the first line of these texts next to your completion candidates. This package also shows any keyboard shortcut for relevant completion candidates (Figure 3.1). When you type `M-x`, you will see a list of functions and a brief description of what they do, making it easier to find what you need.

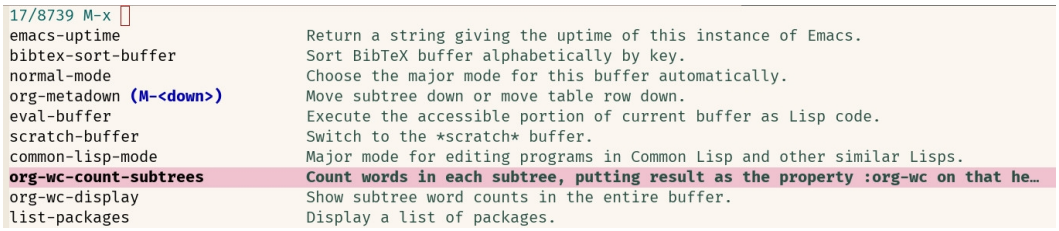

FIGURE 3.1 Minibuffer completion with Vertico, Orderless and Marginalia.

3.2.1 Keyboard Shortcuts

Completion shortens the amount of text you must type and is ideal for discovering functionality you did not yet realise existed. However, as explained in the previous chapter, we usually don't type function names but use keyboard shortcuts.

Remembering which keyboard shortcut you need takes some effort. The Which-Key package by Justin Burkett is not so much a completion system but a great help when trying to remember which keyboard shortcut to use. This package provides a minor mode that displays the keybindings following the currently-entered incomplete command (a prefix) in a popup.

Many keyboard shortcuts have multiple parts, such as `C-x C-f`. The which-key package shows a popup menu that lists all the available options. When, for example, you press `C-x`, the menu will list all follow-up keys and the function they are bound to. Where it says `prefix` in the popup, this means that there is a deeper level. So, by pressing `C-c w`, the EWS prefix, you see a list of the available functions.

If the shortcuts are too large to fit in the popup window, you can move to the next page with `C-h n` and the previous page with `C-h p`. Just typing `C-h` inside the Which-Key popup displays additional options to navigate the list of key bindings.

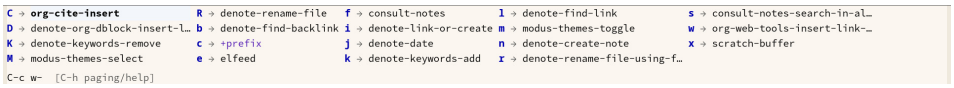

FIGURE 3.2 Which-Key popup window for C-x-w (Emacs Writing Studio).

3.3 Introducing Org Mode

The previous chapter explained how to write a plain text file. Now, we add a new layer of complexity by introducing Org mode, a powerful major mode that comes with Emacs by default. This software was initially developed in 2003 by Carsten Dominik, professor of astronomy at the University of Amsterdam. Since then, countless other developers have continued to advance Org mode. many people, like myself, use Emacs because of Org mode as it is a perfect environment for writing.

You can use Org mode to publish websites, articles and books, keep a diary, write research notes, manage your actions, and more. And on top of all that, it is intuitive to use. This section shows you the basics of writing prose in Org mode. The remainder of the book explains the more specialised functionality of this extensive package.

Start by creating a file with a `.org` extension and start writing, for example, `C-x C-f test.org`. Emacs automatically enables Org mode for any file with the `.org` extension. Org mode is based on Text Mode, so all functionality from the previous chapter also applies.

Each Org document starts with a header that contains metadata and settings relevant to the buffer. The Org mode metadata and settings start with `#+` followed by a keyword and a colon, and the metadata, for example, `#+title: Romeo and Juliet`. The document header can also contain metadata such as a subtitle or a date and other bits of information. Emacs packages can use this information when publishing the text and other functionality. If Shakespeare had used Org mode, the front matter could be:

```
#+title:   Romeo and Juliet
#+author:  William Shakespeare
#+date:    [1597-05-08 Thu]
```

This section only provides a short introduction to using Org mode to write prose. The extensive Org mode manual is available in the info system with `C-h R org ENTER`. Subsequent chapters explain more specialised functionality in Org mode, such as managing projects and exporting.

3.3.1 Document Structure

Books have chapters, sections and paragraphs; articles have headings; poems have verses; and so on. Almost all forms of writing have a hierarchy. One of the rules of writing I like to practice is to set the structure before the content. Org mode has a flexible set of commands to quickly define the structure of your writing project. Defining headings is easy. Write an asterisk as the first character to start a new header, followed by a space. To create deeper levels, add more stars:

```
,* Heading 1
,** Heading 2
,*** Heading 3
```

leading commas?

When you press `M-RET`, the following line becomes a new heading. With `C-RET`, the following line is added after the text in the current section. You can also promote a standard paragraph to a

heading using `C-c *`. Org mode also makes it easy to move and promote or demote existing headings and associated subheadings and text (which in Org mode is a subtree). Just use the with the ? sentence `ALT` and arrow keys. You can also use these keys to move paragraphs of text around.

You can collapse or expand a heading with the `TAB` key. Pressing `S-TAB` collapses the whole buffer, showing only the level one headings. Pressing `S-TAB` once again will show the headings, and repeating it for a second time reveals the entire buffer. You can keep cycling through these modes with the `S-TAB` key.

TABLE 3.1 Org mode structure editing.

Shortcut	Function	Description
`TAB` / `S-TAB`	`org-cycle`	(Un)fold headings
`M-up` / `M-down`	`org-metaup` / `org-metadown`	Move a heading or paragraph up or down
`M-left` / `M-right`	`org-metaleft` / `org-metaright`	Promote or demote a heading
`M-RET`	`org-meta-return`	Insert a new heading

Org mode numbers every heading when the `#+startup: num` option is set in the front matter. The numbering appears in the Org file. Whether this numbering also appears in any published output depends on your configuration (see Chapter).

3.3.2 Text Formatting

To change how Org Mode displays text, you surround it with special characters: `/italic/`, `*bold*`, `_underline_`, `+striketrough+` and `=verbatim=`. In Vanilla Emacs, these markers remain visible but disappear when exporting the document to its published format. You can remove the markers by setting the `org-hide-emphasis-markers` variable. The only problem with hiding emphasis markers that way is that rich text becomes hard to edit because it is unclear whether your cursor is on the marker or the first or last character. EWS installs the Org-Appear package by Alice Hacker (perhaps a synonym?), which displays the rich text markers while the cursor is on a the word but hides them otherwise.

Pretty entities in Org mode relate to special symbols, such as superscript and subscript (`x^{2}` or `x_{2}`) and other LaTeX special characters, such as `\pi`, which display as x^2, x_2 and π. You can toggle this behaviour with the `C-c C-x \` keystroke (`org-toggle-pretty-entities`). For more complex mathematical expressions, see section 3.3.6. By default, Org mode does not require curly braces for sub- and superscripts. But this can cause confusion if you like to write something using $\text{snake}_{\text{case}}$. The EWS configuration limits applying sub- and superscripts to characters between curly braces by setting the `org-use-sub-superscripts` variable to `{}`.

3.3.3 Lists

Sometimes, writing lots of prose in long paragraphs can make the content hard to understand, so non-fiction authors extensively use lists to create clarity in writing. Writing lists in Org mode could not be more accessible. Start a line with a dash and complete the entry with `M-RET` to create the next entry. Using the `TAB` and left or right arrow keys changes the depth of the item and, with the up and down arrows, moves the line up or down in the hierarchy. Change the list prefix with the `SHIFT` and left/right arrow keys. The default options are: `-`, `+`, `1.` or `1)`.

To enable alphabetically ordered lists and numerals, you need to customise the `org-list-allow-alphabetical` variable to `t`. This adds `a.`, `A.`, `a)` and `A)` as additional options to number a list.

```
- Item
```

```
  + next item
    1. The following
    2. And another
       a. Down, down, deeper and down
```

Numbered lists start at one by default as expected. You can add a cookie to instruct Org mode to start the list at a different number. For example, to start the list at number 3, add `[@3]` at the start of the list text, as shown below. The cookie is invisible when exporting this file.

```
3. [@3] First line
4. Second line
```

3.3.4 Tables

A table is another mechanism widely used in technical documents to structure information. Tables in Org mode are plain text with intuitive functionality to add, remove and move columns and rows.

To create a table, start with a pipe `|` symbol, enter the content, and continue until you have defined all columns. Then, end the line with a terminating pipe. Every cell in an Org mode table is flanked by a pipe. You don't have to worry about aligning the text, as the `TAB` key does that for you. When you start a line with `|-` and hit `TAB`, you create a horizontal line. You navigate forward through the cells with the `TAB` or arrow up/down keys. Using `S-TAB` moves you back to one cell. Combining Alt and Shift with the arrow keys adds and removes columns and rows.

```
#+caption: Example table.
#+name: tab:example
| Column 1 | Column 2 |
|----------+----------|
| Sator    |       12 |
| Arepo    |       26 |
| Tenet    |      878 |
| Opera    |       89 |
| Rotas    |       89 |
```

Each table can also have a caption, which starts with the `#+caption:` token and a name (`#+name:`). These lines are mainly used for creating cross-references when publishing your document, which is further explained in Chapter 7.

Org mode has extensive capabilities for creating tables, including formulas for spreadsheet functionality. Consult the Emacs manual for more details.

3.3.5 Images

Although Emacs is a text editor, it can also display images. In Org mode, an image is a link to an image file. An image in Org mode is a link to the file, as shown below. To add an image, press `C-c C-l` (`org-insert-link`), which opens the link menu. Org mode understands many types of links. We want the file type, so type `file:`. Press enter and select the image filename in the mini buffer. You can skip the `file` part by adding the universal argument with the `C-u C-c C-l` shortcut.

```
#+caption: Image caption.
#+name: fig:example
#+attr_org: :width 600
[[file:path/to/image]]
```

After adding the image, you preview it with the `org-redisplay-inline-images` function or `C-c C-x C-M-v`. To toggle pictures in your document, use `C-c C-x C-v` (`org-toggle-inline-images`). The EWS configuration enables default previews in all Org mode buffers, but adding images will switch them off. Hence, you must run the command whenever you add a new image.

You open a link in Org mode with a mouse click or by pressing `C-c C-o` (`org-open-at-point`) with your cursor on the link text. Emacs has some facilities to manage image libraries through the Image-Dired package, discussed in Chapter 8.

Like tables, you can add a caption and a reference name to an image. The pictures in the buffer are all shown at 300 pixels wide. You can configure the preview size to your preference by adding a line above the image, e.g. `#+attr_org:: width 600`. Various other attributes can be added to define how the image is displayed in the published version, as explained in Chapter 7.

3.3.6 Mathematical Notation

Mathematical notation in Org mode is written in LaTeX syntax and surrounded by dollar signs. A full explanation of LaTeX formula notation is outside the scope of this book.

If you have a working LaTeX installation, then Org mode can preview LaTeX fragments as images. To preview the fragment under the cursor, press `C-c C-x C-l` (`org-latex-preview`) to, for example, convert `$\sum_1^\infty \frac{1}{x}$` to $\sum_1^\infty \frac{1}{x}$. This process converts LaTeX formulas to an SVG file stored in a subdirectory named `ltximg`.

The Org-Fragtog package by Benjamin Levy provides convenient functionality to toggle between the plain text source and the image preview of formulas. This means you don't have to use the `org-latex-preview` function repeatedly. When the cursor is in the formula, it shows the plain text, and when it is outside the formula, it shows the graphical version.

3.3.7 Ricing Org Mode

Ricing is slang term among software developed referring to heavily customising the appearance of their editor. This could involve using themes, fonts, and other visual tweaks to create a unique and often flashy look. Emacs is an ugly duckling that can be configured into a beautiful swan. The EWS configuration file contains a range of modifications to the user interface.

The main difference between a text editor and a Word processor is that in Emacs, the design of the text (font, colour and so on) is based on meaning and not the writer's choice. Your chosen Emacs theme will set the size and font for your document. The purpose of the buffer design is to help you navigate the document. This will not be the way it looks when you export the text, which is defined by a template (See Chapter).

The active theme and various configurations and packages define the display of an Org mode buffer. Emacs defines how a buffer looks through `font-lock-mode`. Font locking assigns faces to (or 'fontifies' in Emacs speak) various parts of your text using logical rules. Evaluating `font-lock-mode` toggles between the fully configured version of your Org mode file and the plain text version.

3.4 Checking Spelling

The Emacs minor mode Flyspell provides an interface to the Hunspell spell-checking software. You must ensure that Hunspell is available on your computer. Hunspell is easy to install on Linux computers and is also available for Windows and macOS.

The EWS configuration enables the Flyspell minor mode for all text modes and sets Hunspell as the default checking program. The `flyspell-buffer` (`C-c w s`) function check the spelling in the

buffers. The system provides you with alternatives, which you can select with the mouse button. The bottom of the menu provides some further options, which you can also access with the arrow keys. using `@` removes the suggestions for quick access to these options.

A more productive method is to use the `flyspell-auto-correct-previous-word` (`C-;`) function. This function provides the most likely correction of the first spelling error before the cursor but visible on the screen. Emacs shows the list of possible corrections in the mini buffer. Repeatedly pressing `C-;` will cycle through the options until you return to the original. This function prevents you from having to jump to your spelling mistakes.

You must modify the configuration to set your preferred dictionary as the default. The available English dictionaries in Hunspell are:

- `en_US` (American)
- `en_CA` (Canadian)
- `en_GB` (British)
- `en_AU` (Australian)

Hunspell also supports an extensive collection of other dictionaries, so you can install and choose your own when English is not your preferred language.

If you are multilingual, you can set a different language for each buffer by adding a file variable to the relevant buffer. Ass the text below as the last lines in your Org mode file, where you replace `nederlands` (Dutch) with your preferred language:

Add
but which text?

Emacs evaluates this line and activates the dictionary when you run `M-x normal-mode` and the next time you open the file. This dictionary only applies to the relevant file; all other buffers remain in the default language. Unfortunately, a language apply to a complete buffer and not to sections of text.

applies

3.5 Other Packages and Bespoke Functionality

The EWS configuration installs many other packages, including a package with bespoke functions developed for this book. Each of the following chapters explains the relevant EWS configurations and packages. The complete *Emacs Writing Studio* configuration is displayed in the Appendix including annotations.

3.6 The *Emacs Writing Studio* Workflow

The process of writing can be chaotic as it involves many iterative cycles. But an orderly pattern emerges when we return from the daily grind. We read literature, develop new ideas, develop these ideas and publish the results. Even though the reality is never as linear as this list suggests, it is a helpful guide to organise the *Emacs Writing Studio* workflow (Figure 3.3).

The EWS configuration follows a four-step workflow for producing written creative works. The fifth step is about the overhead to manage your projects and teh final step in the process is share the fruits of your labour to the world:

the to

1. *Inspiration*: Reading, listening and watching

2. *Ideation*: Cultivating ideas
3. *Production*: Writing and editing
4. *Publication*: Create a PDF, ebook or a website
5. *Administration*: The overhead of the writing process.
6. *Communication*: Tell others about your work and exchange ideas

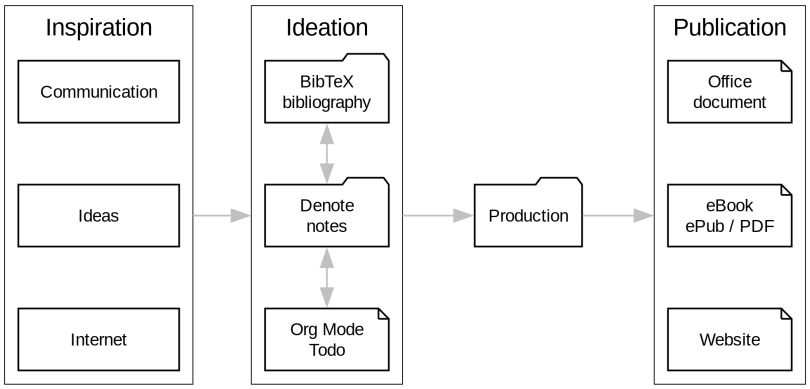

FIGURE 3.3 Emacs Writing Studio workflow.

The basic principle of this workflow is that the author collects information from literature, the web, email and so on (*inspiration*), which they process in a note-taking system. These notes are the central repository of information and are linked to a bibliography (*ideation*). These ideas and notes form the foundation of the writing process (*production*). When the work is completed, the author publishes it in its final format (*publication*). Elated with the final result, the author communicates the new work through email or social media (*communication*). At the end of a long work day, we must also do some *Administration* to keep our systems in good shape.

3.6.1 Inspiration

Ideas don't pop into your mind out of thin air. Our thoughts, plans, and inspirations derive from our lived experiences and depend on what we read, hear, or watch.

Emacs has extensive facilities to read any plain text format imaginable and display PDF files, ebooks and images. Listening to a podcast or watching a video is impossible within Emacs, but it can provide an interface to integrate with external multimedia applications.

You can also maintain a bibliography in Emacs with BibTeX files and link your electronic library to read literature inside or with another program. More recent formats, such as BibLaTeX and CLS-JSON, are also supported.

The Emacs internet browser is not fully featured but a tool that displays websites you like to read as a plain text file. While this approach creates readable websites without distracting clutter and adds, it cannot render more complex internet functionality.

Ingesting all these new ideas is only worthwhile if you keep a record of your new-found inspirations. Hence, maintaining research notes is essential to facilitate the ideation process.

Chapter discusses how to read ebooks, surf the web and consume multimedia files with Emacs.

3.6.2 Ideation

My personal ideation process starts with a physical notebook. This might be a strange thing to admit as somebody who professes admiration for the electronic virtues of Emacs. Physical notebooks have some advantages in the creative process. Firstly, you can use it everywhere. The only exception might be the shower, paradoxically where we get our best ideas. Using a notebook is also a slower process than using a keyboard. Writing with a pen forces you to think a bit more deeply about what you are writing. Writing on paper triggers different neural pathways than writing electronically. Cognitive processes for the paper notes group in this research were more profound (Umejima et al., 2021). Once ideas germinate in my notebook, some will find a place inside Emacs. As part of my weekly action list, I review my physical notes and transfer the features that require archiving to my digital system.

Emacs is an ideal tool for storing notes in plain text. Several packages are available to manage your digital brain. This step in teh EWS workflow revolves around the Denote package by Protesilaos (Prot) Stavrou.

I don't follow any specific, such as *Zettelklasten* or *Bullet Journal*, to manage my notes. My collection of notes is a primordial soup of ideas, categorised using tags and opportunistically linked. Besides extracts from my physical notebooks, Denote also collects information I find online, from my email inbox, social media or journal articles and books. Everything is worth keeping to my Denote collection. Most note-taking systems contain six types of notes:

1. *Fleeting notes*: Quickly take a note while doing something else.
2. *Permanent notes*: Contains new ideas, project files and anything else worth keeping.
3. *Bibliographic notes*: Notes about a book or article linked to the bibliography.
4. *Journal entries*: Regular musings about your life.
5. *Attachments*: Read-only files, such as images, PDFs, videos and anything else that supports the content of the written notes.
6. *Meta notes*: Notes that act as a hub to connect to other notes, for example, about the same topic.

Chapter 5 discusses how to use Org mode and the Denote package to create and manage your collection of ideas.

3.6.3 Production

Once you have gathered your thoughts, it is time to start writing. Emacs developers have published many utilities to assist with the writing process, such as auto-completion, checking spelling and grammar, dictionaries, thesauruses, and other indispensable tools.

Emacs Org mode is ideal for writing articles and books or developing websites. Another popular plain text format for writing is Markdown, which does not have the same flexibility as Org mode. Fountain mode is another plain text format designed to write scripts for theatre, film or a YouTube video.

Chapter 6 describes how to use Org mode to write articles, websites and books.

3.6.4 Publication

The glorious moment has arrived when you can publish the fruits of your labour. Emacs Org mode has powerful capabilities to export the text to various formats, most importantly word processor documents for sharing, PDF files for physical books, ePub for ebooks and HTML for websites.

Org mode exports files to PDF through the LaTeX document preparation system, which is popular with technical authors and publishers. Emacs will export your Org files to LaTeX format, and

then that software exports your file to a beautifully designed PDF file. Many journals accept LaTeX submissions, so you can also export your Org mode to a `.tex` file to send to the editor. You don't need to know any LaTeX to export to PDF, but it helps if you do when creating more complex documents.

Chapter 7 discusses how to use Org mode to convert your plain text document to an electronic or physical publication to share with the world.

3.6.5 Administration

Working through a writing project is a fantastic journey in creative expression, but there is also some overhead in managing your projects.

Emacs interfaces with other GNU software to help you manage your files using the powerful directory editor (Dired). Managing your files is a breeze with this package. You can also use Emacs to manage your photograph and image collection with the built-in Image-Dired package.

Lastly, working on a big project means tracking many tasks. Org mode has a fully functional task management system to help you keep track of your projects. You can implement your personal workflow or use a Getting Things Done (GTD) approach.

Chapter 8 discusses how to manage your files and your projects to keep you own track in your writing projects.

3.6.6 Communication

Emacs can also connect you to the world so you can communicate with other writers and (potential) readers. The software has built-in capabilities to read and write emails, and several external packages, such as *mu4e* and *notmuch*, are available to make sense of your email. Using Emacs for email is a very efficient way to grind through your daily dose of electronic mail. Other packages let you connect to IRC, Reddit, Mastodon and other communication protocols. This book does not discus how to communicate with Emacs, but does provide some hint on where to obtain further information.