Peter Prevos

Emacs Writing Studio

Research, Write and Publish Articles, Books and Websites in Plain Text

Yellow highlighting & green text = notes to myself

Orange highlighting & purple text = edits to book

Contents

Foreword

By Protesilaos (Prot) Stavrou

Preface

In the good old days all a writer needed was a typewriter, a pack of smokes and a glass of whisky, if we can believe the old movies. Writing on a typewriter is a very physical process.

Computers have replaced mechanical typewriters. I was very excited when I wrote my first document on a computer. There was no more need for correction fluid

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. A had to manage a complex web of software tools to work on a single project. Wouldn't it be nice if there was one program that could help you with almost all your tasks?

This book describes ... ?

This book is not a manual on how to use Emacs but a guided tour through Emacs to teach you how it can help you being an author. When you install Emacs it comes fully equipped with extensive documentation. This book includes regular references to the built-in manuals to provide more detailed information.

This book has been completely researched and written with Emacs. In essence this book is about how to write this book with the plain text Org mode format.

The *Emacs Writing Studio* configuration file and the source Org mode files for this book are available in the GitHub repository on `https://github.com/pprevos/emacs-writing-studio`

Acknowledgements

A full list of acknowledgements for an free software project such as Emacs would consist of a huge list of names. Emacs can only exist because of the hundreds if not thousands of volunteers who develop and maintain the code of the core system and the many packages.

My inspiration for writing this book came from the excellent work of people like Protesoleas (Prot) Stavrou, * Wilson from System Crafters, * and many other people who helped me flatten the learning curve of my Emacs journey.

Arjan Zuidhof

Big thanks to the test readers

Test readers

- ☒ Harald Kirsch: 1-3 (email)
- ☐ Harald 5–6
- ☒ Thomas Montfort: 1–3
- ☐ Thomas 5–6

☒ Ben Finney @bignose@fosstodon.org : 1:3

⊟ Bignose 5–6

☐ Prot: 1–6

who volunteered to review the content of the book after a short message on Mastodon. Their feedback has helped to make this book both easy to read and comprehensive, which is a fine balance.

Youtub channels

- LigerLearn, Emacs from Scratch Series.

1

Why Use Emacs?

Imagine needing a different tool for every step of your writing project: one for brainstorming ideas, another for taking notes, yet another for writing the actual book. Your information is scattered in many places and stored in various formats that are incompatible with other programs. You spend a lot of time switching between programs, using different methods to achieve the same goal. Also finding the information you need can be problematic as it is scattered across your drive. Unfortunately, that's how many people manage their workflows. What if you could do this differently and use the same program to create written works from ideation to publication? This book introduces Emacs, a powerful tool that can handle everything from sparking your initial ideas to publishing your finished article, book or website. Emacs lets you ditch the juggling act and focus on your writing.

This book is for authors frustrated by commercial software and are looking for a more efficient workflow and flexibility in their computing needs. This book covers everything you need to know to do research, write and publish electronically or for print. Emacs is originally developed for software developers, but you don't need to be a computer wiz to use it as an author. This book provides a gentle introduction to flatten the learning curve to get started. Most existing Emacs documentation is somewhat technical and assumes prior knowledge. Technical documentation typically delves into the depth of every aspect of the system, which his daunting for new users. This books uses a different approach and peels Emacs like an onion, revealing one aspect of this software at a time. Each chapter adds more detail and complexity, following the typical workflow for an author. This book does not require you to write any computer code because *Emacs Writing Studio* is configured to the most common needs of authors.

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 plugins 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 Stallman 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 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 does not make sense for authors. A text editor is a tool for developing software. From an author's perspective, Emacs is a *text processor* because editing is only the last step in writing an article or book. A text editor is a tool for software developers 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 and converts a text editor to a text processor.

1.2 Why use Emacs?

Authors jot down notes in a research tool, meticulously build bibliographies in a database. Then, they write in a familiar word processor. To stay on top of your deadlines, they juggle a productivity tool for managing projects. Finally, after hours of focused work, the researcher might unwind with a quick game of Tetris to take a well-deserved break from the application circus.

The problem with this perhaps familiar scene is that each program requires you to learn a new set of skills, navigate a distinct internal logic, and bend to its preordained workflow. Most software is inflexible, forcing you to adapt to the developer's vision of how you should work, besides perhaps some configuration options.

Emacs offers a revolutionary approach. You can write research notes, manage a bibliography, and yes, even play Tetris — all within a single, unified environment. Imagine the efficiency of mastering one set of commands instead of grappling with multiple programs. Emacs empowers you to configure and customise it to your exact preferences, transforming it from a mere writing tool into an extension of your personal workflow.

While Emacs might appear different from the eye-candy of modern software, there's a method to its apparent lack of sophistication. Don't be fooled by its austere facade. Beneath the surface lies a robust and contemporary computing environment, meticulously crafted that you can bend to your will.

Another advantage is the longevity of this tool. How you use Emacs 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).

Many writers have lamented the constraints of commercial word processors when tackling large documents. Working with commercial word processor software can be a frustrating experience. These programs were first developed when paper memos and reports ruled the world, and have changed little since. Word processors combine content, layout and typography in one file, focusing on printed pieces of paper. Emacs breaks free from this paradigm by separating the content from the design. This liberating approach allows you to focus on crafting your ideas without getting bogged down in the design of the end product. As an added benefit Emacs an transform the same text file effortlessly into a print-ready PDF, a website, or an ebook.

Emacs empowers you to streamline your workflow, ditch the software juggling act, and focus on what truly matters: your writing.

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

1.3 Malleable Software

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.

The first principle of malleable software is that it is easy to change.[2] With Emacs advanced users can build their own applications using the Emacs version of the LISP language, also called 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 also extend and configure Emacs with any of the thousands of freely available plugins, also called packages. EWS is a curated collection of such packages to align it with the needs of authors. 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:

> 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.

1.4 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, the graphical approach is not user-friendly at all 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. Furthermore, the *What You See is What You Get* approach is outdated as it is only relevant for printed documents. 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.

Emacs is a plain text processor that focuses on the semantic meaning of characters on the screen instead of how they will eventually look on a page or screen. Most text is just a paragraph, but adding a few symbols converts it to a heading or figure caption. 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 hides the definitions for font sizes, colours and other attributes.

Plain text most commonly has a `.txt` extension and does not have any formatting such as bold text. 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). However, there are many other plain text formats, such as HTML, Markdown, LaTeX, and Org Mode, that include a vast range of capabilities to turn plain text into a work of art.

Plain text can be read across all computer systems, so you never have to worry about locking your writing into a proprietary format or being stuck using a particular software package. Anything you

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

write in Emacs can be read with NotePad, TextEdit or any other such software. The only difference is that the other programs don't have the versatility of Emacs. Plain text is not a niche application. 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 Australian platypus (Source: asciiart.eu). However, there is no need to resort to these ancient techniques as Emacs can also display images in the most common file formats such as JPG and PNG.

```
          _.- ~~^^^'~- _ __ .,.- ~ ~ ~  ~  -. _
 ________,'       ::.                      _,-  ~ -.
((      ~_\   -s-  ::                     ,'          ;,
 \\       <.._ .;;;`                     ;           }  `',
  ``=======''    _ _- _ (   }             `,          ,'\,  `,
              (// _ _,i   ! _ ~ - -- - _ _'_-_,_,,,'    \,  ;
      cfbd       ((((____/            (,(,(, ____>        \,'
```

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.

The graphical approach distracts the mind from the content and lures the user into working on style instead of writing text. 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. Office workers around the globe waste a lot of time trying to format or typeset documents in graphical environments. In plain text, the content and most semantics are directly visible and editable by the user.

Following the plain text Emacs way helps you become more productive by not worrying about the document's design until you complete the content. As I write this book, it only takes a few keystrokes to convert the text into a fully formatted ebook or print-ready PDF copy. 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. Sections, paragraphs, illustrations and other document elements are labelled as such using various conventions (Khalili & Auer, 2015).

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. Table shows how to denote *italic text* in four popular plain text formats.

Table 1.1 Italic text in common plain text formats.

Format	Italic semantics
HTML	`<i>Italic Text</i>`
LaTeX	`\emph{Italic Text}`
Markdown	`_Italic Text_`
Org mode	`/Italic Text/`

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 one of my web pages, 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.

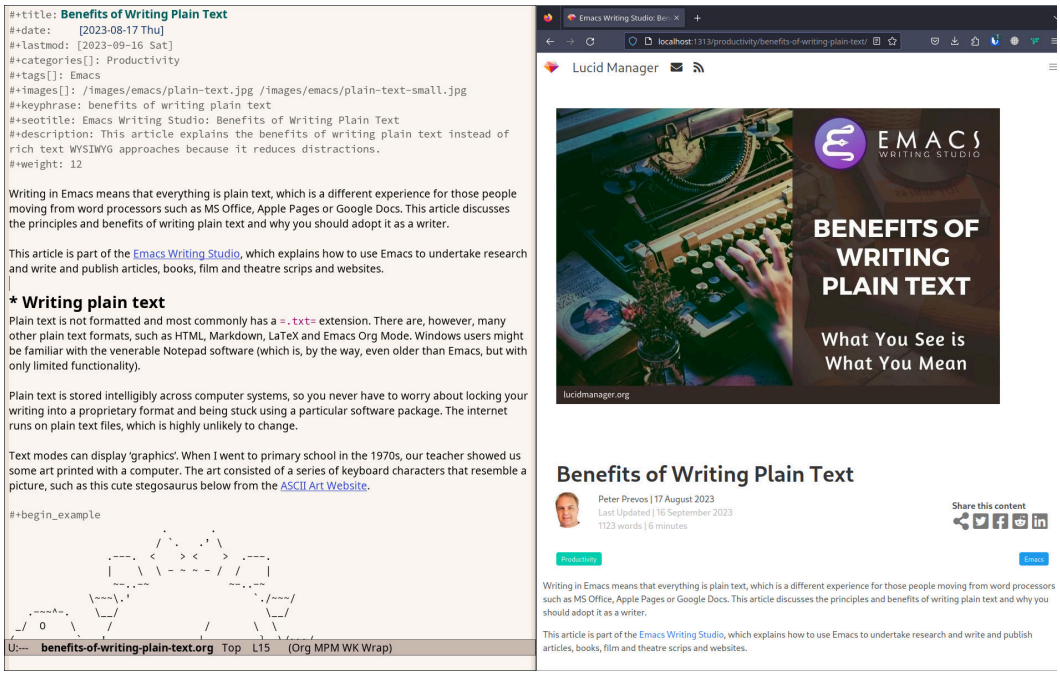

Figure 1.1 What You See is What You Mean approach to writing.

The style sheet in plain text writing is separate from the text. This means that you can easily export your document to different formats. Org mode in Emacs has a potent export engine (chapter 7) that can 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.

1.5 The Learning Curve

Emacs supposedly has a steep learning curve because its possibilities are so vast. 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

sometimes mocked. Perhaps Emacs is more complex than other plain text processors, 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.

The key to learning Emacs and flattening the curve is not to get overwhelmed by the virtually infinite possibilities and master those bits of functionality that you need to do what you need to do right now. Even without any configuration, Emacs can do a lot.

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. 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 modern handicraft terms, such as cutting, pasting, and moving files between folders as if they were pieces of paper. These differences are not only part of Emacs' charm, but also of its power. You will find that the Emacs equivalent of these familiar software tasks more potent that what is common in modern software.

After you master the techniques described in this book and start developing your personal EWS. Learning Elisp to configure the software to your wishes might sound daunting, but you can simply copy and paste (kill and yank) examples from the internet. 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 specialised research, writing, and publication engine. This book focuses on using this configuration instead of delving deeply into the technical details. The Appendix to this book describes the full configuration for readers interested in venturing into the depths of Emacs Lisp.

1.6 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. Notwithstanding this admiration, Emacs has some limitations.

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 does not 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 materially impact your output because you write with your mind, not the keyboard.

1.7 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 intrepid 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. Chapter three takes you through the principles of using EWS 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 Citar bibliography tool and Denote note-taking plugin. 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 follows the typical workflow for a writing project. Chapters four to eight describe each aspect of this workflow, taking you from research to writing and to publication. Chapter eight covers administrative tasks such as managing projects and your file system.

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 provides some advice on how to learn even more and become an Emacs Ninja. The appendix to this book contains the annotated EWS configuration with some guidance on how to read it and make changes.

The best way to read this book is by sitting in front of your computer and try things out. Experiment different options, create some files and play around. Don't worry about breaking things, just restart Emacs and have another go. Playing with something is the best way to learn anything, so boot up your computer and get ready for an adventure.

2

Getting Started with Vanilla Emacs

Start your engines; it is time to install and use Emacs. The installation process for Emacs depends on your operating system. The GNU Emacs website (`emacs.org`) contains instructions on how to install Emacs on the most common operating systems. Please note that you will need the latest version of Emacs, which at the time of writing is 29.

Once you have installed the software, 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. When the splash screen closes, you enter the 'Scratch Buffer', which you can use for temporary notes. In Emacs terminology, a buffer is an area in the memory of your computer that holds content, which can link to a file. Emacs does not save the content of the Scratch Buffer when you exit the program, so don't start writing your dissertation just yet.

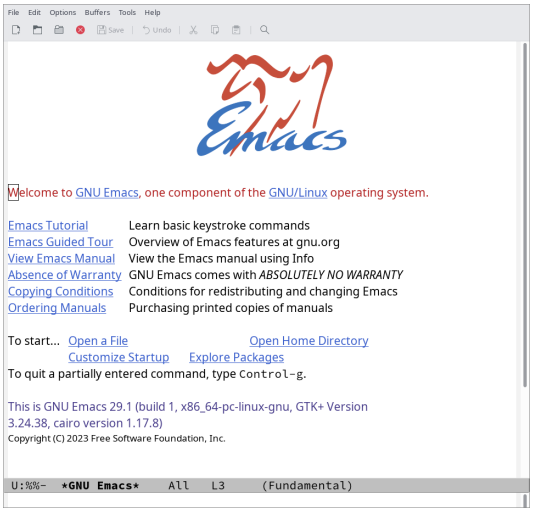

Figure 2.1 Emacs 29 splash screen.

2.1 Working with the Keyboard

Emacs is by and large a keyboard-driven application. You can use the mouse for occasional tasks, but there is no need for a mouse. There is no agreement on whether using a keyboard or a mouse is most efficient (Tognazzini, 1992; Omanson et al., 2010). Arguably, clicking on an icon in a menu bar seemingly 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 functional-

ity. 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 and you can use it in Emacs for some tasks, like selecting text or moving the cursor. Vanilla Emacs also contains drop-down 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 drop-down menu.

You have probably used a keyboard for years and wonder why this section appears in this book. As Emacs was developed before standardisation of computer interfaces and the way it interacts with the keyboard is slightly different to what you are perhaps used to. Lets start at the basics. A standard computer keyboard has five types of keys:

1. Alphanumeric: Letters, numbers and punctuation.
2. Editing: such as arrow keys and backspace
3. Function and multimedia
4. Escape
5. Modifier keys: Shift, Control, Alt, Windows/Command

Pressing an alphanumeric keys adds the character to the computer's memory and displays it on the screen. This is a complex way of saying that they are used in typing. Editing keys, such as arrow keys, page up and down, delete, and backspace do what it says on their labels. Function and multimedia keys perform more complex tasks. For example, pressing `F3` records a macro. Multimedia keys are usually defined by the operating system and activate tasks such as increasing the screen brightness or playing music. The escape key is the most potent member of the keyboard. Like Dorothy's Ruby Slippers in the *Wizard of Oz*, pressing it three times gets you out of trouble when you are stuck.

In principle, these are the only keys you ever need to write prose, but we want to do more than just insert and edit 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 have the same modifier keys but there is no 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 some 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 unused but still available as a modifier key in Emacs.

Emacs documentation and this book abbreviate key sequences. For example, `C-a` stands for pressing the Control and a keys at the same time. The dash indicates that the first key modifies the second key, while a space between keys indicates that they are typed consecutively. Actual spaces are shows and `<space>`. 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

"... are shown as..."?

and Shift a), which requires the nimble fingers of a sleight-of-hand artist to execute smoothly. The shift modifier is mostly not indicated because `C-M A` is the same as `C-M-S a`. The escape key can also act as a modifier key. Pressing escape once is the same as holding the meta key. So `ESC x` is the same as `M-x`.

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

All keystrokes in Emacs execute a function, which means they perform a task. Functions that are visible to the user are called commands and this book with use these words interchangeably. Most technical books display the names of functions in `typewriter-font` to distinguish them from normal text. Emacs functions are always written with dashes instead of spaces between words, which hackers sometimes refer to as kebab-case. Not all functions have a keyboard shortcut, but when a shortcut is available, it is also shown in typewriter text. Knowing the names of functions and the keyboard shortcut helps to better understand how Emacs works. You also need to know the function name because keyboard shortcuts can change as they are fully configurable.

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. After pressing a prefix key, Emacs displays it in the echo area, awaiting further input. The length of key sequences is theoretically unlimited, but they are usually nor more than three or four keys in practice, for example `C-c w s d`. Some packages also use prefix keys. *Emacs Writing Studio* (EWS) uses `C-c w` as a prefix to store all its keybindings. This means that you can group key bindings for easy memorisation. The standard prefix keys are:

- `C-c`: Mostly used by packages
- `C-h`: Help functions
- `C-x`: Mostly used for built-in Emacs commands
- `M-x`: Execute commands (discussed in the next section)

Due to Emacs's age, it does not comply with the Common User Access (CUA) standard for user interfaces, 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. Other standard keys, such as `C-z`, are already used for different functionality. You can configure Emacs to recognise these common keyboard shortcuts, but (EWS) sticks to the Emacs version.

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, by the time you complete this book, you gradually understand its intricacies and drive the system like a virtuoso. 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).

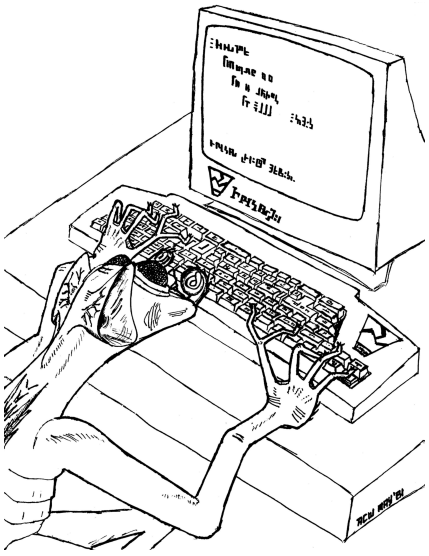

Figure 2.2 Cover of the 1981 version of the Emacs manual.

2.2 Issuing Commands

The modifier and prefix keys provide an abundance of shortcuts to issue commands to Emacs, but the number of keys is not unlimited so some functions don't have a shortcut. If a function does not have a default keybinding then you can provide your own, just be careful not to create conflict between existing shortcuts. The Appendix explains how to do this. Functions without a keybinding need to be called by name.

The standard 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 `M-x tetris RET` to play Tetris. Don't get too distracted; just press `q` a few times to exit the game and get back to your work.

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 execute the `visual-line-mode` function and change how Emacs wraps paragraphs, you type `M-x visu` and `TAB`.

To see how this completion works, enter `TAB` after each letter you type into the minibuffer. You will notice that Emacs narrows the completion candidates as you get closer to your desired selection, until there is only one option. This principle also works with variable names and filenames. The

TAB key is your secret weapon to help you remember and discover functions, variables, file names, buffer names and other selection candidates. You can also access the menu and tool bars with the mouse, but they only contains a small selection of the available functionality as the screen is simply not large enough to hold them all.

The remainder of this book only mentions the names of commands without adding the M-x and RET parts. So when the text suggest to use a function or command called example-function-or-command, you do so with M-x example-function-or-command RET. Any available keyboard shortcuts are also indicated, in which case you can use the short way to access the function.

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. An editing mode provides major and minor modes. A major mode is like opening an app within the Emacs environment, just like you open an app on your phone. 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. go ahead and try, issue the artist-mode command and drawing with the mouse. A new *Artist* item will become available in the menu bar to provide some options.

A major mode determines the core functionality for an open buffer. A buffer is the part of the memory that holds the text of a file you just opened, or other content. More about buffers in section 2.5. Each buffer has at least one major mode, and each major mode has its own functionality with specific key bindings and drop-down menus. All major modes share the same underlying Emacs functionality, such as copying and pasting (killing and yanking) and opening files, but they add specialised tasks, for example exporting to a PDF file.

Minor modes provide further 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 (the keymaps) and drop-down menus depend on the major and minor modes that are active at the time. Some keymaps are global and apply to the whole of Emacs. Other maps are specific to a mode. Unless a mode overrides it, 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 in which it is used.

2.4 Opening Files

Opening files in Emacs is called 'visiting a file' and uses the find-file function (C-x C-f). So effectively, finding, opening and visiting a file have the same effect. Emacs opens the file and displays its contents in the buffer, ready for editing. 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', see chapter). Alternatively, you can open a file with the toolbar icon or through the menu bar.

Emacs asks you to select a file or folder in the minibuffer. Typing the complete path to the file you seek would be tedious, so Emacs assists with auto completion, explained in section 2.2. Please note that a file path in Emacs is separated by forward slashes and not by backslashes, as is the case in Windows (`C:/Users/Wittgenstein/` and not `C:\Users\Wittgenstein\`).

When finding a file, Emacs starts in the folder of the currently active buffer. You can simply remove the text before the cursor to move to higher levels in the directory tree. If you like to find a file in your home directory, ignore the text in the minibuffer and type a tilde followed by forward slash (`~/`) and `TAB`. To start searching in the root folder or your drive, type two forward slashes (`//`). On Windows computer the best method is to type the drive letter, followed by a colon and a slash (`c:/`). When you hit the `TAB` button twice, all the available files and folders appear in the minibuffer.

Create a file with a `.txt` extension to get some practice and start writing into the buffer. After you have added some text, you might wat the save your work to the file. The contents of the file stays the same until you save the buffer. After you complete your edits, `C-x C-s` saves your buffer to its associated file. To save a buffer under a new name, you can use `C-x C-w` (table 2.2). You can see whether a buffer is different from the associated file in the mode line at the bottom. If it contains two asterisks at the start, then you file needs saving. two dashes, means that the content of the file is the same as the buffer.

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 menu bar on top and toolbar with icons just below it. The window starts below the toolbar. Each window contains a buffer, which holds the contents of a file. Buffers can also contain a user interface or output from functions. The mode line below each window displays the name of the buffer or its associated file and other metadata. Each frame has an echo area at the bottom, where Emacs displays feedback. Echo is a computer science term for displaying information, such as error messages and other feedback. The bottom of the page also contains the minibuffer, an expandable part of the bottom of the screen where Emacs seeks your input when, for example, selecting a buffer or 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. The active buffer is the one you are currently working on. The names of special buffers, such as `*Messages*`, are surrounded by asterisks. Most buffers, except those surrounded by an asterisk, are linked to a file.

Emacs is highly stable, and some users have hundreds of open buffers because they rarely need

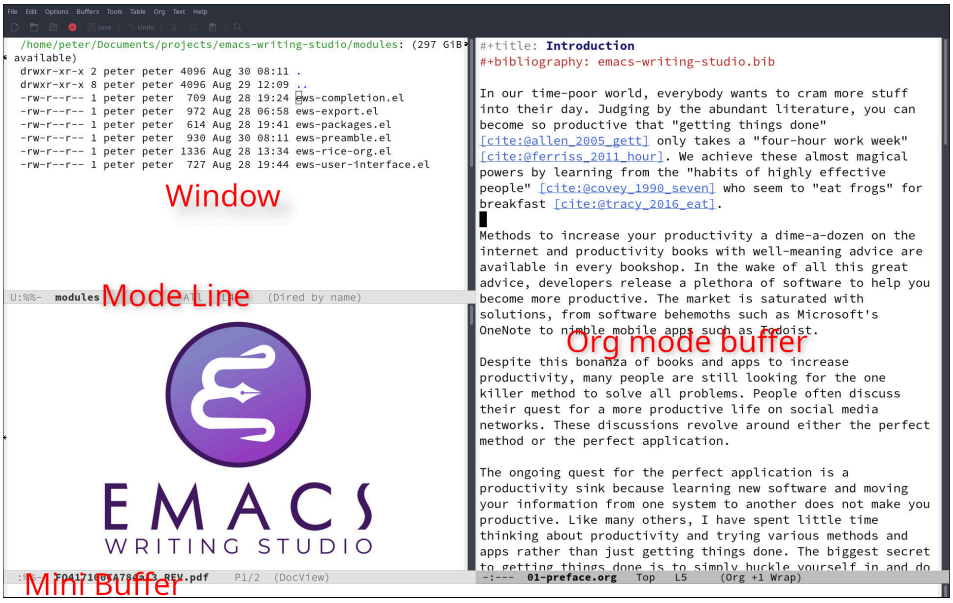

Figure 2.3 Emacs frame with three windows, a Dired buffer, image buffer and Org mode buffer.

to restart the program. 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.

By default, a frame has one window. You can split 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.

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 and loose focus. Activating follow mode with `follow-mode` flows the text of the buffer so the two or more windows become columns of the same document. When the cursors moves below the bottom of the left window, it appears again in the right window, so all windows share one cursor. To deactivate follow mode, run the same function again.

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

2.6 Finding Help

Emacs has an extensive built-in help system with different ways to access information, accessible with the `C-h` prefix key. The complete Emacs manual is available with `C-h r` (`info-emacs-manual`). This manual opens in Info mode, which is a specialised mode for manuals. The full Emacs manual is not bedtime reading but more a pool of knowledge to dip your toe into when the need arises. The `g` (`Info-goto-node`) key lets you jump to a chapter or section of the text, using minibuffer completion discussed earlier. For example, `C-h r g help` takes you to the page about the help system.

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 or `S-space` 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`. If you are looking for something specific, then `Info-search` (`s`) lets you search for specific terms. As always `q` quits the screen.

Some packages in Emacs have their own manuals. You can view a list of the available manuals with `C-h R TAB` (`info-display-manual`). Also here you can use minibuffer completion to find a manual. Not all Emacs packages have an extensive manual. Another method to find out information about a package is the `describe-package` function (`C-h P`). This function extracts information from the source code and provides a summary of the package and a link to it's home page.

The help system also has other commands to find more specific descriptions. If you want to find out which command binds a specific shortcut, use `C-h k` and enter the key sequence. 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". To find out more about a variable, use `C-h v` (`describe-variable`) and type its name. And to learn more about a command use `C-h x` (`describe-command`). A popup window describes the relevant variable or command, which you can close with `q`.

The remainder of the book provides references to the relevant Emacs help system for readers who like to know more details about the system. You don't need to read the manuals because this book contains everything you need to know to get started as an Emacs author. The documentation in the manuals is technical and concise and as such can be difficult to understand for beginners. The references to Emacs documentation are for people interested in knowing more details about how the software works.

2.7 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. 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. The Emacs manual provides a detailed description of all functionality relevant for writing human languages (as opposed to computer languages), which you can read with `C-h r g basic` and `C-h r g text`.

2.7.1 Moving Around in a Buffer

You can move the cursor with arrow keys and other standard navigation keys. Emacs documentation sometimes refers to the cursor as '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, but 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>`	`backward-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.

2.7.2 Search and Replace

While jumping around the text with arrow keys and other functionality is great, sometimes you know exactly what you need, which is when you use 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. An incremental search (`C-s`) begins as soon as you type the first character of the search term. As you type the search query, Emacs shows you where it finds this sequence of characters. Repeatedly pressing `C-s` steps through the matches in the buffer. When you identify the place you want, you can terminate the search with `C-g` and the cursor jumps back to the original location. When exiting the search with the Enter key or an arrow key stops the cursor at the current location so you can edit the text.

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` recycles 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 enter key or `q` quits the replacement process. More options are available, which you can glean by hitting the question mark.

2.7.3 Copy and Paste Text

Writing is fun, but sometimes it is more efficient to copy something you wrote previously or copy a citation from somebody else (referenced of course). The system for copying and pasting text works a bit different from modern systems 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. In a plain text context, a paragraph is a line of text separated by blank lines. 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 described 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'. 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 kill ring entry to the buffer
`M-y`	`yank-pop`	Replace previously yanked text with the next kill ring entry

2.7.4 Correcting Mistakes

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

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”, erase it with `M-d`, change it to “Plato”, and then undo this edit to revert back to Socrates and add some more text. In standard word processors, you cannot return to the state where the text mentioned Plato (State B in Figure 2.4). In Emacs, all previous states are available. You can 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. This behaviour can be confusing at first, but you will learn to love it after a while because you never loose any edits.

Another feature of the Emacs undo system is that it can apply only to a selected region. Lets say that you have just completed the first chapter and have started writing the chapter two. You then realise that you need to undo some of the edits in the chapter one. If you use the undo function, it will first undo all your work on chapter two before changing the first chapter. You can solve this problem by selecting the relevant region of text in chapter one and then issue the `undo` command over just that region.

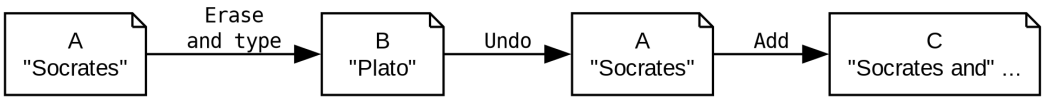

Figure 2.4 Emacs undo states.

2.7.5 Languages Other than English

For the majority of the world, English is not their first language. When you set the keyboard settings in your operating system to another language, Emacs can get confused when using modifier keys. Typing `M-x` on a Russian computer results in `M-`, which Emacs cannot compute.

Emacs supports a large range of input methods to type the rich variety of languages of the world. To see an overview of the various languages that Emacs supports run `view-hello-file` (`C-h h`). An input method either converts keyboard characters to a different one or it converts a sequence of characters into one letter. For example, with the Balinese input method, the letter D becomes a . Using one of the various methods to type Chinese, you start keying and a menu appears in the minibuffer from where you can select the desired character. So 'san' can become , or one of the other characters in the menu.

To choose an input method for the current buffer use `C-x <RET> C-\` (`set-input-method`), which lets you select the preferred method in the minibuffer. The current input method is indicated at the start of the mode line at the bottom of the window. You can temporarily disable the chosen method with `C-\`. Using this key again takes you back to the selected input method.

For more specific information on how to use your keyboard to write another language, use `C-h I` which runs the `describe-input-method` function. To view a list of all available input methods run the `list-input-methods` command and a new buffer pos up with a long list of the languages of the world. The Emacs manual provides detailed information on the various input methods with `C-h r g input`.

2.7.6 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) font height, type `C-x C-0`.

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 sequence of characters that finishes with a return. A visual line relates to how it is displayed in Emacs. The default setting is that logical lines continue beyond the screen boundary. While this is perhaps 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, execute `visual-line-mode` in the minibuffer. Doing this every time when working on a buffer is a bit tedious and this is where configuration comes in. We need to configure the system to enable line wrapping for all text modes by default.

2.8 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.

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 box of building block to modify to meet your preferences. But before installing the EWS configuration, let's first introduce the principles of configuring Emacs.

2.8.1 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.

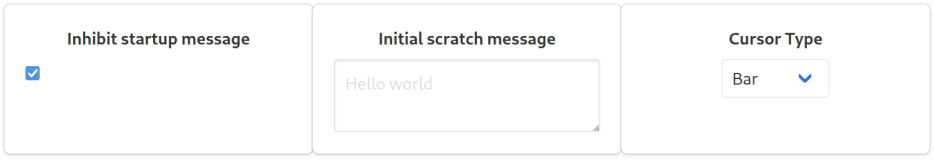

Figure 2.5 Typical graphical configuration screen.

Being a plain text program, Emacs does not have such facilities but uses the Emacs Lisp 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. Fun fact, 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 suggestion to setting a 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. In this case the parameter is a string of letters, nested between quotation marks. The last line sets the cursor type to a bar. This variable has other predefined options, such as 'bar' or 'hollow'. To prevent Emacs from confusing this option with a variable, it uses a single quotation mark (also called a tick mark) before the text.

While on the surface, the text-based method seems more complex than ticking and writing in boxes and picking a drop-down 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 2.7.6 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)
```

2.8.2 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 configurations when Emacs starts. You can run Emacs without an init file as show in the previous chapter, but you will undoubtedly want to modify the defaults. The first time you start Emacs, it will create the configuration folder which is where the init file lives. This folder 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.

2.8.3 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 can be written in the init file. For example, if you want to remove the toolbar from view, you type `M-x customize-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`). Other variables will ask for a different type of input. You can change the state of this variable with the toggle button. The 'Apply' button brings this change to immediate effect. When clicking 'Apply and Save', the new value is saved to the init file.

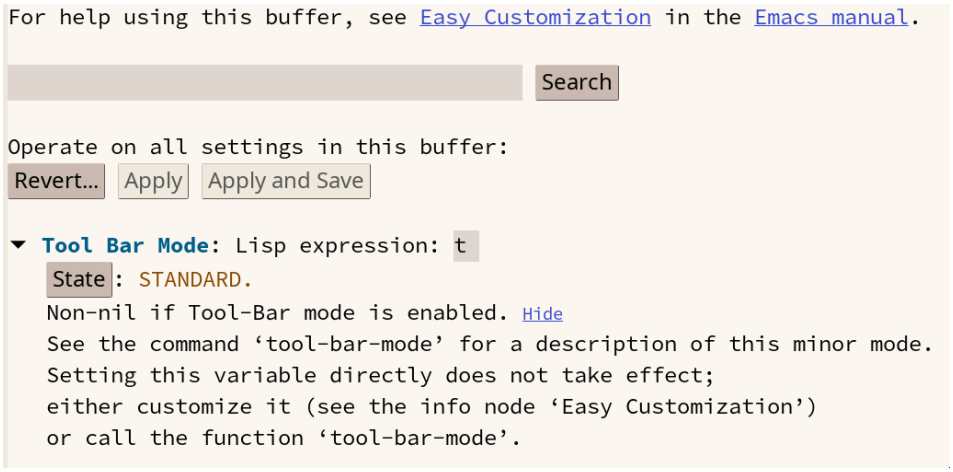

Figure 2.6 Customisation screen for `tool-bar-mode`.

2.8.4 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. This naming convention might seem back to front, as using `upgrade-all-packages` is linguistically better. However, the convention for naming Emacs Lisp functions is that the first word is the package name, which in this case is package. This naming convention makes it easy to group functions by package.

2.9 Exiting Emacs

Working with Emacs is so much fun you might never wan to shut it down. But all good things come to a temporary end, so we might need to shutdown (kill) Emacs occasionally. The `C-x C-c` shortcut (`save-buffers-kill-terminal`) kills the Emacs session, but not before checking for unsaved buffers. 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 options for dealing with each or all of them. You can answer `y` or `SPC` to save the file mentioned in the echo area or `n` / `DEL`

to abandon it. Keying `C-r` lets you look at the buffer in question before deciding. 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 exiting Emacs or loosing any text. Don't stress if you can't remember all this. Using `C-h` displays a help message describing these options.

Alternatively you can issue the `restart-emacs` command to reboot your configuration.

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 major change to vanilla Emacs is the completion system. *Emacs Writing Studio* uses the Vertico / Orderless and Marginalia packages 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 that might not suit everybody. Within the Emacs world, there is occasionally some discussion about what constitutes a sensible default configuration. No matter how interesting these debates can be, such a state does not exist. One person's sensible default is another person's computing nightmare, so feel free to change anything in EWS to suit your ideal worldview. The Appendix contains the complete annotated configuration and suggestions for making changes.

3.1 Installing *Emacs Writing Studio*

You don't have to learn to program in Elisp to use Emacs. You can start with the *Emacs Writing Studio* configuration to get you going. The next step is to install EWS and start using it. The remainder of the book does not explain any methods to configure Emacs but explains how to use the functionality in EWS to research, write and publish articles, books and websites. The Appendix to this book provides a detailed explanation of the EWS configuration.

To install the EWS configuration, download the `init.el` and `ews.el` files from the GitHub repository (`github.com/pprevos/emacs-writing-studio`) and save them in the configuration folder. The location of the configuration folder depends on your operating system and Emacs version. Type `C-h v user-emacs-directory` to identify its location in the popup help buffer. You can close this buffer by pressing `q`. Copy the bot files from the EWS repository to this directory. EWS will activate after you evaluate the `restart-emacs` function or the next time you start the program.

If you an experienced Emacs user, then you can try the EWS configuration by creating a separate folder for the init file and start Emacs with `emacs --init-directory <folder-path>`. That way you can try EWS without clobbering your existing configuration.

The EWS configuration also installs a package from GitHub with some bespoke variables and additional functionality. All such variables and functions start with `ews-`. You will need to configure these variables to prevent errors as you work through this book.

Emacs can also in integrate with various other free software to extend its functionality. Some of the features provided by EWS requite you to install additional software that Emacs will control. You don't need to know how these packages work as Emacs will control them for you. Each chapter in this book outlines which software is required for which purpose.

The `ews-missing-executables` function checks if external software is available on your system. Emacs writes a message in the minibuffer if any of the recommended tools is missing. You can jump to the `*Messages*` buffer with `C-h e` to review the output. If packages are missing, then Emacs will function normally, but some features might be unavailable. The relevant chapters in this book provide more details which software is required and the tasks it undertakes.

This book was written and published with the configuration it describes, so it is fully tested to real-life conditions. The GitHub repository for EWS also contains the `documents` folder which contains the Org mode files for this book. You can download these files to see an example of a book written in Emacs.

3.2 Minimalist Interface

Emacs is 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.

The EWS 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.

3.2.1 Themes

A theme is a set of instructions that describe the colours of defined text parts. Colours in a text processor play a different role than in a word processor. Colours in Emacs are semantic, which means that they indicate the function of the 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 processors: 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 processor users prefer dark themes. Light themes are not bad intrinsically 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 t` shortcut toggles between the light and dark side (insert Star Wars pun here). Using `C-c w t s` 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 theme.

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 R modus`. The Appendix provides some more information on how to customise the theme for your personal settings.

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 as explained in the Appendix.

3.2.2 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 `w` will use the same amount of space, just like mechanical typewriters. This type of letter is ideal when writing code or tables because it helps to align the text.

- `Fixed pitch font`
- Variable pitch font

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 variable pitch mode for the ideal mix between font types.

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

In Emacs lingo, 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 main ones to use 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-pitch` 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 customise this way overrides any theme settings, so ideally, only customise font family and size.

To see which fonts are available, you use the 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.

3.3 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 Mender to provide a seamless experience.

The Vertico package 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 Marginalia package displays the first line of these texts next to your completion candidates. This package also shows available keyboard shortcuts 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 and whether there is a keyboard shortcut to access it.

3.3.1 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.

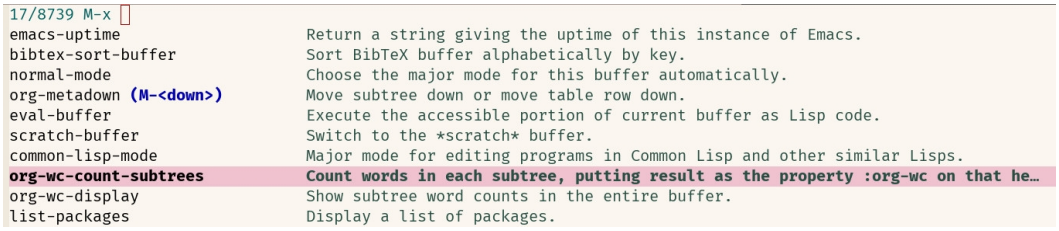

Figure 3.1 Minibuffer completion with Vertico, Orderless and Marginalia.

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 submenus and 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.

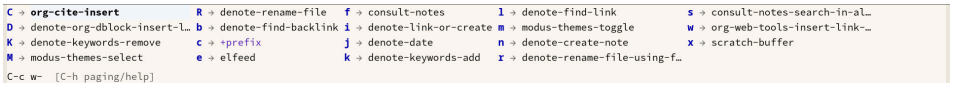

Figure 3.2 Which-Key popup window for C-c-w (Emacs Writing Studio).

3.4 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 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 derived from text mode, so everything explained in 2.7 also applies to this section.

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. Subsequent chapters explain more specialised functionality in Org mode, such as managing projects and exporting. The extensive Org mode manual is available in the info system with `C-h R org`.

3.4.1 Document Structure

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

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

When you press `M-RET`, the following line becomes a new heading. With `C-RET`, the new line is added after the text in the current section. You can also promote a standard paragraph to a heading using `C-c *` (`org-toggle-heading`). 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 `ALT` and arrow keys to move a subtree around the document. You can also use these keys to move paragraphs.

When the cursor is on a heading the `TAB` key collapses the text. Repeatedly pressing `TAB` shows the subheadings and then again the full text. To collapse the whole document, add the Shift key. Pressing `S-TAB` collapses the whole buffer, showing only the level one headings. Pressing `S-TAB` once again will show headings, and repeating it for a second time reveals all text. You can keep cycling through these modes with the `S-TAB` key (figure and table). You can recognise folded headings by the ellipses (`...`) at the end of the line. The Org-Modern package (section 3.4.7) also changes the asterisks to triangles. When the triangle points to the right, the heading is collapsed and when it points down, the heading is open.

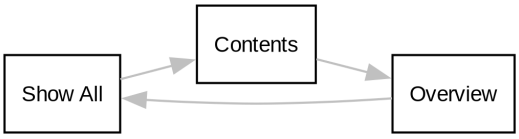

Figure 3.3 Global cycling in Org mode with `S-TAB`.

Org mode also provides a set of commands to make it easier to jump between headings. These commands let you move between headings of the same level and move up in the hierarchy. Table lists some the available commands related to the structure of Org mode documents.

3.4.2 Text Formatting

Writing all words in the same style can be boring and some text needs some emphasis. To change how Org Mode displays text, you surround it with special characters: `/italic/`, `*bold*`, `_underline_`, `+strikethrough+` and `=verbatim=`. In Vanilla Emacs, these markers remain visible but disappear when exporting the document to its published format.

The EWS configuration hides these markers. The only problem with hiding emphasis markers

Table 3.1 Org mode structure editing.

Shortcut	Function	Description
C-c *	org-toggle-heading	Convert paragraph to heading
TAB / S-TAB	org-cycle	(Un)fold headings
M-<up> / M-<down>	org-metaup / org-metadown	Move a heading or paragraph
M-<left> / M-<right>	org-metaleft / org-metaright	Promote or demote a heading
M-RET	org-meta-return	Insert a new heading
C-c *	org-toggle-heading	Convert paragraph or vice versa
C-C C-n	org-next-visible-heading	Move to next heading
C-c C-p	org-previous-visible-heading	Move to previous heading
C-c C-u	outline-heading-up	Move to the higher level
C-c C-f	org-forward-heading-same-level	Move next at the same level
C-c C-b	org-backward-heading-same-level	Move previous at the same level

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 therefore uses the Org-Appear package by Alice Hacker, which displays the rich text markers while the cursor is on a the word but hides them otherwise.

3.4.3 Lists

Writing lots of prose in long paragraphs can make content hard to understand, so non-fiction authors use lists to create clarity in writing. Writing lists in Org mode could not be easier. Start a line with a dash and complete the entry with M-RET to create the next entry. Using the Alt and left or right arrow keys changes the depth of the item. The Alt key with the up and down arrows moves the line up or down in the hierarchy. You can change the list prefix with the SHIFT and left/right arrow keys. The default options are: -, +, 1. or 1). You can convert a paragraph to a list with C-c - (org-toggle-item). Repeatedly using this command change the bullet type, just like shift and the arrow keys. To demote a list item back to a paragraph, simply remove the list characters.

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

Numbered lists start at one by default but you can add a cookie to start the list at a different number. For example, to start the list at number 3, add [@3], as shown below.

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

Org mode can also order your list with the org-sort function (C-c ^). You will be prompted to select a sorting method, which can be either numerically, alphabetically or more advanced options.

3.4.4 Tables

A table is another mechanism in technical publications to structure information in lieu of prose. Creating tables in Org mode uses an intuitive plain method to add, remove and move columns and rows. To create a table, start a line with a pipe (|) symbol, enter the content, and continue until you have defined all columns. Then, end the line with another pipe. Every cell in an Org mode table

is flanked by a pipe. You don't have to worry about aligning the text because the `TAB` or `C-c C-c` automatically add spaces to adjust the column sizes.

Navigate forward through the cells with the `TAB` or arrow up/down keys. Using `S-TAB` moves the cursor back one cell. Combining Alt and Shift with the arrow keys adds and removes columns and rows. When you start a row with `|-` and hit `TAB`, you create a horizontal line across the table. You can also add a horizontal line below the cursor and move to the next row with `C-c RET`, which in this context calls the `org-table-hline-and-move` function.

```
#+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 can only handle simple tables without spanning columns or rows. To create more complex tables, Org mode integrates with the builtin table package by Takaaki Ota. These tables have a slightly different syntax to Org mode, as illustrated below in this table of German articles. Note that Org mode syntax inside cells in these types of tables is not recognised. Because of interference with other Org mode functionality, Takaaki Ota tables cannot be edited directly in an Org buffer. To edit such a table use `org-edit-special` (`C-c '`). To convert a standard Org mode table to the more complex format use `org-table-create-with-table.el`, bound to `C-c ~`. To learn more about the syntax for this package, read the manual with `C-h P table`.

```
+------------+-----------+----------+----------+-------------+
|            |             Singular            | Plural      |
|            +-----------+----------+----------+-------------+
|            | Masculine | Neuter   | Feminine | All genders |
+------------+-----------+----------+----------+-------------+
| Nominative | der       | das      | die      | die         |
| Accusative | den       | das      | die      | die         |
| Dative     | dem       | dem      | der      | denen       |
| Genitive   | dessen    | dessen   | deren    | deren       |
+------------+-----------+----------+----------+-------------+
```

3.4.5 Images

Although Emacs is a text processor, it can also display images. In Org mode, an image is a link to an image file, so the text and the images remain separate files. To add an image, press `C-c C-l` (`org-insert-link`), which opens the link menu. Org mode understands many types of links and for images we need a file link, so type `file:`. Press enter and select the image filename in the minibuffer. You can skip the `file` part by adding the universal argument with the `C-u C-c C-l` shortcut, from where you can start selecting the image file. Your buffer will now contain a link that starts with `file:`, the directory and file name. Under the hood, Org mode wraps the link between double square brackets, so it looks like: `[[file:path/to/image]]`.

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

After adding the link, you can preview the image with the `org-redisplay-inline-images` function or `C-c C-x C-M-v`. To toggle previewing pictures in the whole document, use `C-c C-x C-v` (`org-toggle-inline-images`).

The EWS configuration enables default image previews in all Org mode buffers, but adding a new image removes the preview so you run the redisplay command. 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, for example: `#+attr_org:: width 600`.

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.

Image links are links to other files without a description. You can use the same process to link to any type of file and add a description, so it looks like a regular hyperlink. The structure of a complete file link is: `[[file:path/to-image][Description]]`. Org mode hides the structure of links and only shows the description. You can change this behaviour with `org-toggle-link-display`.

Like tables, you can add a caption and a reference name to an image. Various other attributes can be added to define how the image is displayed in the published version, as explained in Chapter 7.

3.4.6 Mathematical Notation

Technical authors often rely on mathematical notation, which in Org mode is written in LaTeX syntax. You can use simple LaTeX commands in your text, known as pretty entities, or write fully-formatted mathematical notation.

Pretty entities (more precisely UTF-8 characters) in Org mode relate to special symbols, such as superscript and subscript (`x^{2}` or `x_{2}`) and other characters, such as `\pi`, which display as x^2, x and . Org mode also understands other LaTeX characters, such as Greek letters (`\alpha` to `\omega`), superscripts and subscripts. You can write these plainly inside an Org buffer. EWS converts these to their 'pretty' version by default. The underlying text remains the same. You can switch this behaviour on and off with the `C-c C-x \` keys (`org-toggle-pretty-entities`). 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 'snake$_{case}$'. The EWS configuration limits applying sub- and superscripts to characters between curly braces.

For more complex formulas you need to use the full LaTeX formula notation. A formula is surrounded by one or two dollar signs. A single dollar sign indicates an inline formula, while a formal with double dollar signs the formula has its own paragraph. To give you a taste of what LaTeX formulas look like, this is Ramanujan's formula for π, both graphically and in LaTeX notation. A full explanation of LaTeX formula notation is outside the scope of this book. You can reverse-engineer this example as it contains what you need for 80% of mathematical formulas.

$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801} \sum_{n=0}^{\infty} \frac{(4n)!}{(n!)^4} \times \frac{26390n + 1103}{396^{4n}}$$

```
$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801}
\sum_{n=0}^{\infty} \frac{(4n)!}{(n!)^4} \times
\frac{26390n + 1103}{396^{4n}}$$
```

If you the pretty entities functionality is enabled, then some LaTeX symbols are converted to mathematical notation, so you might want to disable this when writing mathematical expressions with `C-c C-x \`.

Org mode can preview LaTeX fragments as images if the `dvipng` program is available. To preview the fragment under the cursor, press `C-c C-x C-l` (`org-latex-preview`). 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 LaTeX fragments and the image preview. When the cursor is inside a formula, Emacs shows the plain text, and when outside a formula, it shows the graphical version, preventing the need for manual switching.

3.4.7 Ricing Org Mode

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

The main difference between a plain text processor and a WYSIWYG word processor is that in Emacs the design of the text (font, colour and so on) communicates meaning and rather than design. Your Emacs theme sets the colours and perhaps also fonts for your document. The purpose this styling is to help you navigate the document. The way your document looks in the buffer is not what it looks like when exported to the final product.

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. Run this function on an existing Org mode file to see the difference between pure plain text and a fontified text. To take it a step further, you can open an Org mode file and run `text-mode` to disable all Org mode functionality and see the file in its raw beauty. To jump back to safety, simply run `org-mode` to restore the file.

EWS uses parts of the Org-Modern package *

3.5 Checking Spelling

Writing with a spell checker has become the ultimate security blanket for authors. Without the squiggly red lined underneath our text, the final product would be littered with typos, at least in my case. The Emacs Flyspell minor mode provides an interface to the Hunspell spell-checking software, so you must ensure that it is available on your computer. The EWS configuration enables the Flyspell minor mode for all text modes.

There are basically two ways to correct your writing. Either just keep the juices flowing and check the complete text when your complete the session, or fix typos as detected.

The `ispell` function (`C-c w s s`) walks through possible spelling mistakes in the buffer or in a selected region. This function displays the proposed corrections at the top of the window. You can select the preferred version by entering the relevant number. The minibuffer provides a menu to ignore the typo with the space bar, accept it for this session (`a`), insert in your personal dictionary with `i` and other options which `C-h` reveals. Any key not in the menu quits the spell checking session.

To fix typos on the fly use the `flyspell-auto-correct-previous-word` (`C-;`) function. This function replaced the first detected spelling error visible on the screen before the cursor with the

most likely correction. Repeatedly pressing `C-;` cycles through the options until you return to the original version. The echo area shows the list of possible corrections. Typing any other key breaks the chain. All options are also stored in the undo system, so you can still go back if you accidentally break the chain and still need changes.

This function prevents you from having to jump to your spelling mistakes and back to where you came from. To store the supposedly mistyped word in your personal dictionary, move to the word and use `ispell-word` (`M-$`) and use the options in the menu.

The Hunspell software has access to a wide collection of dictionaries, including variations of English, which you have to install as a separate download. The default dictionary for EWS is the Australian English dictionary. See the Appendix for instructions on how to set your preferred default dictionary.

If you are multilingual, install multiple dictionaries and set a different language for buffers in the non-standard language using a file variable. Add the text below as the last lines in your Org mode file, where you replace `nederlands` (Dutch) with your preferred dictionary:

```
# Local Variables:
# ispell-local-dictionary: nederlands
# End:
```

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

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 stand back from details of the daily grind. We read literature and get inspired, develop new ideas, produce new works and publish the results. Even though reality is never as linear as this list suggests, it is a helpful guide to organise the *Emacs Writing Studio* workflow and the content of the rest of this book (Figure 3.4).

The basic principle of this workflow is that the author collects information from literature, the web, movies, 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*). However, there is a fifth step. At the end of a long day of writing and editing, we must also do some *Administration* to keep our systems in good shape.

3.6.1 Inspiration

Ideas don't pop into minds out of thin air. Our thoughts, plans, and inspirations derive from our lived experiences and 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 to organise and access your collection of electronic literature. Emacs can also browse the internet, although the built-in browser is not fully featured but a tool that displays websites as plain text. Chapter 4 discusses how to read ebooks, surf the web and consume multimedia files with Emacs.

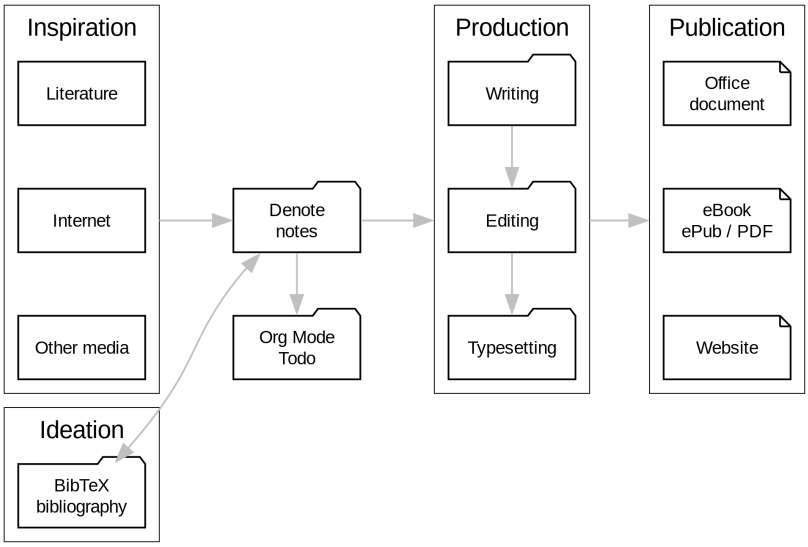

Figure 3.4 Emacs Writing Studio workflow.

3.6.2 Ideation

Ingesting all these new ideas is only worthwhile if you keep a record of your new-found inspirations. Hence, maintaining notes is essential to facilitate the ideation process. Emacs is an ideal tool for storing notes in plain text. Several packages are available to manage your digital brain. This step in the EWS workflow revolves around the Denote package by Protesilaos (Prot) Stavrou.

You don't need to follow any specific note taking methods such as *Zettelklasten* or *Bullet Journal*. My personal collection of notes is a primordial soup of ideas, categorised using organically grown tags and opportunistically linked. Besides digital musings, You can add anything worth keeping to Denote, including binary files such as PDFs or photographs. Chapter 5 discusses how to use Org mode and the Denote package to develop a personal knowledge management system.

3.6.3 Production

Once you have gathered your thoughts, it is time to start writing. Org mode is ideal for writing articles and books or developing websites. Emacs developers have also published many additional utilities to assist with the writing process, such as completion, grammar checking, a dictionary, thesaurus, and other indispensable tools. During production you also might want to collaborate with other authors, which requires some control over different versions. Chapter 6 describes how to use Org mode to write articles, websites and books and manage large projects.

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 print-ready PDF files through the LaTeX document preparation system, which is popular with technical authors and publishers, but can be used for any type of physical book.

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 of 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). You can also use Emacs to manage your photographs and images 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.

4

Inspiration: Read, Listen and Watch

A common interview question for authors and other creative people is where they get their ideas from. The responses to this simple question are often mystical and vague, but a big part of the answer is straightforward. Authors need three resources for their inspiration: imagination, writing skills and the works of other writers.

Therefore, the first step in *Emacs Writing Studio* workflow (EWS) is consuming information created by other people, either by reading, listening or watching content. The thoughts and ideas of other people are the foundation upon which authors build their works. In 1675, Sir Isaac Newton wrote a letter to Robert Hooke in which he penned the now-famous phrase, "If I have seen further, it is by standing on the shoulders of Giants". This quote is perhaps a cliché, but it still holds true, even in the age of generative AI. Depending on other creators' works is essential not only in academic writing with its strict conventions on referencing. All authors need to read to be able to write.

Even though Emacs is a text processor, it can evoke other software to display ebooks. Besides being malleable software, Emacs is also an interface to other software. This means that we can use Emacs for tasks beside writing and editing text. You can as such also use Emacs to listen to music or podcasts, and watch videos, but only with the assistance from some additional software. Keeping track of your collection of physical and electronic literature collection is another important task of a scholar. Emacs can also assist with creating a searchable bibliography for easy access to your resources using BibTeX files. You can access electronic files and hyperlinks from this bibliography and attach notes to each item. This chapter explains how to use Emacs to read electronic documents in the most common open formats, surf the web, and consume multimedia files. This chapter also explains how to create a plain text database to manage your collections of electronic books.

4.1 Reading eBooks

In the pre-digital era, people shared information physically, from clay tablets to paper books and other tangible formats. Before the internet the only way to get information was to visit a library and spend hours and days wading through catalogues and books. Digital documents have become the norm in the last two decades and visits to the local library have been replaced by internet searches.

Most ebooks are not distributed in plain text but either PDF, ePUB or DjVu. A wonderful exception is Project Gutenberg (`gutenberg.org`), a vast electronic library of over 70,000 copyright-free eBooks. This library started in 1971 and distributes books in plain text and other open formats.

Emacs can only render ebooks in open formats. An open format, as opposed to a proprietary format, has a publicly available specification so anyone can develop software that can create and read these files. The greatest advantage to an open format for ebooks is that your book can be displayed on any device. The most common proprietary formats such as Apple ibook and any Kindle books cannot be read with Emacs. Proprietary file formats pose a grave risk to cultural continuity. Some governments therefore require open documents formats to enforce digital sovereignty and liberate themselves from the dominance of large internet corporations (Pohle & Thiel, 2020).

Emacs needs assistance from external software to display ebooks and word processor docu-

ments. The diagram below shows the various external software packages necessary to render pages from the supported file formats. This external software converts each page to a plain text buffer or a PNG image, which Emacs can display. The following external software is required to read the most common ebook formats:

- Ghostscript (`gs`) and/or MuPDF (`mutool`): These two packages can convert PDF files to images.
- Poppler (`pdftotext`): Another tool to render PDF files and convert them to plain text.
- LibreOffice (`soffice`): Office productivity software suite to view office documents.
- DjVuLibre (`djvulibre`): Reading DjVu ebooks
- `unzip`: Extract files from ZIP files to read ePub books.

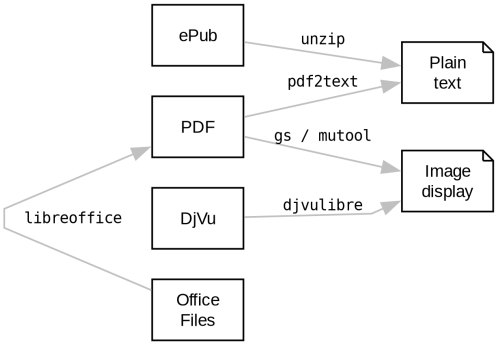

Figure 4.1 Document conversion in Doc-View.

The *Emacs Writing Studio* configuration issues a warning when any of these packages are unavailable. Emacs will work normally, but some functionality might be unavailable. The warnings are displayed in the `*Messages*` buffer, which opens in another window with `C-h e` (`view-echo-area-messages`). If you are a Linux or Chromebook user, these packages will be available through your system's package manager. Windows users can use the Chocolatey package manager (`chocolatey.org`) or MSYS2 (`msys2.org`) to get this software. Apple users can install Linux tools using the Homebrew package manager (`brew.sh`).

4.1.1 PDF Files

Portable Document Format (PDF) is a versatile system developed by Adobe in the early 1990s. PDF presents documents consistently, regardless of the software, hardware, or operating system used to view them. PDF is codified in an international standard (ISO 32000) and has become the main open format for electronic literature.

Each PDF file includes a complete description of a fixed-layout document. The file contains the text, fonts, layout and typography, images and other information needed to display the content. A PDF document is not a real ebook because the layout is fixed, contrasting with other electronic formats that adjust to the screen. A PDF file follows a traditional physical layout and typography, assuming that documents are printed.

Emacs can display PDF files with the built-in DocView major mode with some assistance from external software. GhostScript or MuPDF convert PDF files to images for DocView to display. This conversion means that DocView can be sluggish when reading large documents, and the system issues a warning when a file is larger than fifty megabytes.

To navigate a document use the arrow and page-up / page-down keys to turn pages. Several other keyboard shortcuts are available to find your way through the document (table).

Table 4.1 Doc-View keyboard shortcuts.

Keystroke	Function	Description
`P`	`doc-view-fit-page-to-window`	Zoom to the full page
`W`	`doc-view-fit-width-to-window`	Fit width to window
`H`	`doc-view-fit-height-to-window`	Fit height to window
`+` / `-`	`doc-view-enlarge` / `doc-view-shrink`	Zoom in and out
`M-g g`	`doc-view-goto-page`	Jump to page
`M-<` / `M->`	`doc-view-first-page` / `doc-view-last-page`	Jump to first or last page
`k`	`image-kill-buffer`	Kill (close) the file
`?`	`describe-mode`	Help file

When `mutool` is available on your system, you can also use the `imenu` (`M-g i`) command to view a table of contents and jump to a chapter, assuming the PDF file has bookmarks.

To enable searching through a PDF file you need the `pdftotext` tool, part of the Poppler software. Fun fact: this utility is named after an episode from the *Futurama* cartoon series. You can search within a document with `doc-view-search` (bound to `C-s`), which creates a list of all matching pages and shows how many pages contain the search query. After the search, you can jump to the next page containing a match with an additional `C-s`. DocView does not highlight the searched term, but pressing `C-t` (`doc-view-show-tooltip`) shows the search results for this page in a tooltip. Poppler also allows you to view a PDF file as a plain text file with the `C-c t` shortcut (`doc-view-open-text`). This option makes searching and copying relevant text to your notes easier. To return to the graphical view of the text, press `C-c C-c` twice.

4.1.2 Office Documents

The DocView package can also read Microsoft Office and LibreOffice files (text documents and presentations). To enable this functionality, you must install the LibreOffice software package, a free and open-source office productivity software suite. When opening an office document, Emacs invokes LibreOffice to convert the file to a PDF and display it in DocView (Figure 4.1), leveraging the functionality explained in the previous section. You can use this method to open not only word processor files but also presentations and spreadsheets, all of which are converted to PDF.

Office documents are compressed XML files, so when you open them with an archiving utility, you can view their content in plain text. Unfortunately, two standards for office documents exist. One standard is used by open-source software, while commercial software vendors embrace the other version. Differences between these standards can result in minor formatting issues when reading files created with commercial software.

4.1.3 DjVu Books

DJVU (pronounced *déjà vu*) is a file format intended for scanned books. Since a DJVU file can contain high-quality colour images, photographs, text, and drawings, it's often used for archival documents. DjVu files use the `.djvu` or the `.djv` file extension. DocView can read DjVu files when DjVuLibre is available on your system.

DocView displays PDF, office documents and DjVu documents, so the same functionality that is available when viewing PDF files also applies to other formats. You can read more details about this package in the Emacs Manual, which you can quickly find with `C-h r g docu`.

The DocView program within Emacs can be a bit slow when dealing with large documents. The Appendix provides some alternative solutions to this problem.

4.1.4 ePub Files

An ePub file (Electronic Publication) is a widely used open format for digital books, magazines, and other written content. Unlike PDF and DjVu formats, The content adapts to the geometry of the screen, making it ideal for e-readers, tablets, and other devices. An ePub file is a website in a box. You can see the raw content of an ePub file when opening it with an archiving utility. The file consists of a collection of XML files that define the content and design of the book, and any image files for illustrations. This file format thus needs the `unzip` package to enable viewing the content.

The `nov` package by Vasilij Schneidermann provides useful functionality for viewing ePub books inside Emacs. Open an ePub file and scroll with the space bar, the arrow keys or the `page-up` / `page-down` keys. Several keyboard shortcuts are available to move through the book (Table 4.2).

Table 4.2 Keyboard shortcuts in the Nov package.

Keystroke	Function	Description
`t`	`nov-goto-toc`	Table of contents
`n`	`nov-next-document`	Next chapter
`p`	`nov-previous-document`	Previous chapter
`q`	`quit-window`	Quit
`?`	`describe-mode`	Help buffer

To increase or decrease the text size, use the `C-x C-+` and `C-c C--` shortcuts (`text-scale-adjust`). When enlarging the font, it might move parts of the text outside the window. To reset the length of the lines, press `g` to re-render the document (`nov-render-document`).

You can copy and paste text from ePub files into your bibliographic notes (Chapter 5) using the `kill-ring-save` command (`M-w`). You can also copy images from an ePub file. Open the file as an archive with the `a` key (`nov-reopen-as-archive`), which shows the document's internal structure. From here, you can navigate to the relevant image file, copy it with the `C` key (`archive-copy-file`), and select the new location and name. Unfortunately, there is no functionality to preview images inside an archive file.

4.2 Managing Your Digital Library

Maintaining a large collection of literature can quickly lead to some chaos so most scholars use an electronic bibliography to keep track of what they read. Emacs can help you to build a plain text library catalogue to easily access your bibliography. This bibliography can also link to notes (chapter 5) and link to scholarly citations (chapters and 7).

Creating and managing a bibliography requires three Emacs packages that seamlessly integrate with each other. The built-in BibTeX Mode assists with creating and maintaining a plain-text bibliography. The Biblio package by Clément Pit-Claudel searches online scholarly databases and inserts relevant items to you bibliography. Bruce D'Arcus' Citar package provides easy access to your bibliography using the minibuffer completion system. These three packages turn Emacs into a fully featured literature management system.

4.2.1 Getting Started with Emacs BibTeX Mode

BibTeX Mode is a major mode to create and manage bibliographies. As the name suggests, this mode uses the BibTeX file format as your default literature database. BibTeX is a plain text format to organise literature and citations. It is typically used for typesetting scholarly publications with LaTeX (Lamport, 1994). A typical entry for a book will look like this:

```
@article{stallman_1981_emacs,
  title      = {EMACS the Extensible, Customizable Self-Documenting Display Editor},
  author     = {Stallman, Richard M.},
  year       = 1981,
  journal    = {ACM SIGOA Newsletter},
  volume     = 2,
  number     = {1-2},
  pages      = {147--156},
  doi        = {10.1145/1159890.806466},
  keywords   = {Emacs}
  file       = {computing/stallman-1981-emacs.pdf}
}
```

Each entry starts with an `@`-sign and the publication type (book, article or other types), followed by a curly brace and a unique citation key. The following lines contain the relevant data about this entry. BibTeX can process different types of literature, such as articles and conference papers, each of which has its own field types. BibTeX ignores unrecognised fields, which provides opportunities to use the format for other purposes, such as attaching files and keywords. BibTeX is a plain text file with the `.bib` extension.

You can store one or more bibliography files in a central folder to refer to them from anywhere in Emacs. In EWS, this central folder is defined by the `ews-bibtex-directory`. You need to customise this variable to the desired location, which by default is `~/Documents/library/` (refer to section 2.8.3 on customising variables). The tilde at the start is the path to your home directory, which you can view with `C-h v ews-home-directory`.

Your BibTeX files have to reside in this directory so that the system knows where to find them. Attachments can live in subdirectories. Any file paths for BibTeX entries start at this location. For example, when the BibTeX entry states: `file={topology/article.pdf}`, the attachment is stored at: `~/Documents/library/topology/article.pdf`. You can have more than one attachment per entry, separated by a semi-colon. BibTeX attachments have no formal file naming convention, so you can use your favourite method.

You don't need to install external software to get started. Just create an empty file in your bibliography directory with a `.bib` extension, and Emacs enables BibTeX mode when you open the file.

4.2.2 Adding New Entries

Emacs BibTeX mode uses templates to add new entries. To add a new reference, use the `bibtex-entry` function (`C-c C-b`) and select the relevant publication type. Emacs also provides a shortcut for each kind of literature. You can read a list of these commands when inside a BibTeX file with `C-c C-e ?`.

Most fields are optional, but each literature type has at least one compulsory field. Optional fields start with `OPT`. You must complete at least one field that begins with `ALT`, such as author or editor. The EWS configuration adds fields to categorise literature with keywords and to attach one or more files. The template below shows the book template. The title, publisher and year fields are compulsory and you have to complete the author or the editor field, or both. All other fields are

optional. Each type of literature has its own template, the example below shows the template for a book.

```
@Book{,
  ALTauthor    = {},
  ALTeditor    = {},
  title        = {},
  publisher    = {},
  year         = {},
  OPTvolume    = {},
  OPTnumber    = {},
  OPTseries    = {},
  OPTaddress   = {},
  OPTedition   = {},
  OPTmonth     = {},
  OPTnote      = {},
}
```

Jump from field to field with `C-j` (`bibtex-next-field`) and complete all required fields and one of the `ALT` fields. When done, press `C-c C-c` (`bibtex-clean-entry`) to check the syntax and remove empty fields. This function also assigns a unique citation key to the entry using some configurable rules. You can override this citation key and set one manually, as long as it is unique. BibTeX mode issues a warning when it finds duplicate keys. To clean-up the buffer and align the fields, use the `bibtex-fill-entry` function (`C-c C-q`). This command also removes redundant curly braces to create a clean look.

To enter author or editor names, place the family name first, followed by a comma and the first name or initials. Separate additional authors by "and", e.g. "Hawking, S. and Penrose, R.". If you copy and paste an author name the first and family name might be the wrong way around. A nice Emacs hack is to use the `org-transpose-words` function, bound to `M-t`, which swaps the order of two words left and right of the cursor. For example, transform "Stephen Hawking" to "Hawking Stephen" with `M-t` and add a comma after the last name to finish it off.

Table 4.3 summarises the most salient keyboard shortcuts and functions in Bibtex Mode.

Table 4.3 Overview of keyboard shortcuts to add and edit entries.

Keystroke	Function	Description
`C-c C-b`	`bibtex-entry`	Add an entry for selected type
`C-c d`	`bibtex-empty-field`	Empty the current field
`C-j`	`bibtex-next-field`	Jump to next field
`C-down`	`bibtex-next-entry`	Jump to the next entry
`C-up`	`bibtex-previous-entry`	Jump to the previous entry
`C-c C-c`	`bibtex-clean-entry`	Clean the entry
`C-c C-q`	`bibtex-fill-entry`	Align the fields

This section is only a short summary of this package's capabilities. The documentation for this package is a bit sparse. Jonathan Le Roux (`jonathanleroux.org`) publishes a comprehensive manual on his website that explains the functionality provided by this package in great detail.

The EWS package provides two convenience functions to assure the integrity of the links between the BibTeX files and the attachments. The ideal state is that the files mentioned in the BibTeX entries do actually exist and vice versa, all files in your bibliography folder are listed in the bibliography. The `ews-bibtex-missing-attachments` function lists all missing attachments in the `*Messages*` buffer. To fix this discrepancy, you need to either remove or edit the `file` field in the

relevant BibTeX entry, or fix the name of the file in your collection. The `ews-bibtex-missing-files` function lists any attachments in your bibliography directory that are not registered in your BibTeX files. To fix any issues, either rename the relevant file or add it to the associated BibTeX entry in the `file` field. These two functions help you to ensure that you can always access your electronic literature through the Citar menu.

4.2.3 Using Biblio to add New Entries

BibTeX mode requires you to type all entries manually, which is inefficient and could easily lead to errors. Clément Pit-Claudel's Biblio package lets you browse and import bibliographic references from online sources to undertake a systematic literature review. Currently, the package enables you to search CrossRef, DBLP, arXiv, doi.org, and Dissemin.

Crossref interlinks millions of items from a variety of content types, including journals, books, conference proceedings, research grants, working papers, technical reports, and data sets. Linked content includes materials from scientific, technical, and medical (STM) and social sciences and humanities (SSH) disciplines. DBLP is a computer science bibliography website with more than seven million publications. arXiv (pronounced "archive") is an open-access repository of pre-prints and post-prints approved for posting after moderation but not peer review. In mathematics and physics, almost all scientific papers are self-archived on the arXiv repository before publication in a peer-reviewed journal.

Most electronic publications have a Digital Object Identifier (DOI), a persistent identification code that links to metadata about the publication. The DOI system ensures that publications can be found, even when the address changes. DOIs are widely used to identify academic, professional, and government information, such as journal articles, research reports, data sets, and official publications. The DOI is shown on screens and in print as `doi:10.1142/9789812777171_0020` or as a URL.

To use the Biblio package, open the relevant BibTeX file, run `biblio-lookup`, select the appropriate database and enter a search query. Once the search results are available, a new buffer opens. Select your target with the arrow keys or search in the buffer with `C-s`. Once you find the needed literature, insert its BibTeX record into the buffer where you called the function with `i`. Alternatively, you can copy the BibTeX record with `c` and paste it into place later. You quit the search results with `q`. To see all possible commands in this buffer, use the `h` key.

The `biblio-doi-insert-bibtex` function inserts a BibTeX record based on a DOI number into the current buffer. You can enter just the identifier in one of the two formats mentioned above. You need to activate this command from within a BibTeX buffer with the cursor on the location you like the new entry to appear.

Unfortunately, large corporate publishers still hold the world's academic knowledge behind lock and key. However, open access and pre-print publications are slowly becoming the norm. The Dissemin website searches for openly accessible copies of papers in an extensive collection of open repositories and websites. To use this service with Emacs, evaluate `dissemin-lookup` to show information about the open access status of a paper using a DOI number. You can also press `x` in the search menu for the `biblio-lookup` function to check for open access version.

The Biblio package is useful, but one minor inconvenience is that you must jump to the relevant bibliography file before inserting a new entry. It also provides two search functions that can be combined into one. The bespoke `ews-bliblio-lookup` (`C-c w b b`) function lets you select the BibTeX file where you would like to store the search results before choosing one of the available sources. This function also combines searching for DOIs with the other sources, removing a few steps from your workflow.

4.2.4 Using Citar to Access Bibliographies

Maintaining one or more BibTeX or BibLaTeX files to store your library is a good start, but the content is not easy to search and access, especially when you have multiple bibliographies. We need an interface that makes it easy to find literature on your computer and access its resources, such as links, attachments, and notes.

The Citar package uses minibuffer completion to access your bibliographies. It provides access to hyperlinks, notes, attachments, and the source bibliography file for selected items. Citar also integrates with `org-cite`, Org mode's citation module (Chapter 6) and can provide access to notes for your literature (Chapter 5).

Within Org mode, bibliographies can have a global or a local scope. The global bibliography is a set of BibTeX files available from anywhere within Emacs, located in the `ews-bibtex-directory`. You can also link one or more project-specific local BibTeX files to an Org mode file. This might be useful when working on a project that only references literature relevant to the work at hand. Add a local bibliography to an Org mode file with something like `#+bibliography: ews.bib`.

Citar processes all bibliography files in the global folder and any files referenced in an active Org mode buffer. If you add another global bibliography file, then you need to let Citar know with the `ews-bibtex-register` function (`C-c w b r`). This function registers all bibliography files in the nominated directory for Citar to use and displays them in the echo area. You only use this function to register new files, it is not required when adding new literature to an existing file.

You activate Citar with `citar-open`, which in EWS is bound to `C-c w b o`. A menu pops up in the minibuffer where you can search your collection. The first three columns in the menu indicate which entries include a hyperlink (`L`), one or more attached files (`F`) and an associated note (`N`). The remainder shows the author, year, title, citation key and keywords (Figure 4.2).

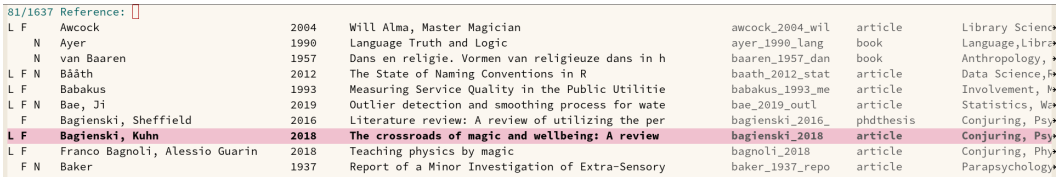

Figure 4.2 Example of the Citar menu.

Finding literature with Citar is easy due to the power of the Vertico and Orderless packages. After every keystroke, Citar narrows the list of options to relevant matches. Select your candidate with the arrow keys, or use `TAB` to select more than one entry, and hit the Enter key. You can filter the Citar completion menu for entries with an attachment using `":f"` and with links with `":l"`. Chapter 5 explains attaching notes to literature and chapter 6 shows how to insert citations. After hitting the Enter key, Citar provides a popup menu in the minibuffer where you can open attachments or follow any hyperlinks listed in the BibTeX entry.

4.3 Surf the Web

Emacs has a built-in web browser called the Emacs Web Wowser (EWW). This package focuses on readability over functionality by displaying websites as a plain text. It can display images but does not render any CSS or run JavaScript. A wowser is somebody with moral views for temperance and abstinence. The plain text approach to browsing is as such an ethical stance on the World Wide

Web and its security and privacy issues. Although some people suggest that the name stands for the reaction you might have when you first see a website rendered in plain text.

You can open a URL or search the web with the command `eww`. If the input doesn't look like a URL, EWW will search the web with DuckDuckGo, a privacy-focused search engine that doesn't track your online behaviour. After the page loads, use the arrow and page-up / page-down keys to navigate the page. Several keyboard shortcuts are available to navigate the webpage.

- `<`, `>`: Beginning and end of the page
- `R`: Readable format (only display the main text)
- `G`: New search or website
- `H`: Browsing history
- `M-I`: Toggle images
- `l` / `n`: Previous and next page
- `q`: Quit the window
- `w`: Copy the URL under the cursor or the URL of the page
- `&`: Open the page in the external browser
- `?`: Help file with list of other keyboard shortcuts

The enter button opens links (`eww-follow-link`). If you want the new page to open inside a new buffer, use `M-ENTER` (`eww-open-in-new-buffer`).

The most useful option pressing `R` to ignore most of the navigation parts of the page and focus on the content. If the page does not render in EWW or you are warned about needing JavaScript, use ampersand (`&`) to escape to your system's default web browser. When opening a link to a website from inside a non-EWW Emacs buffer, it is opened in the default browser for your operating system.

If you find a website you like, bookmark it with `eww-add-bookmark`, bound to the `b` key inside EWW. The `eww-list-bookmarks` function lists all stored bookmarks, from which you can select one and visit the page. Keeping a collection of bookmarks is helpful, but you have to regularly visit these sites to see if anything new has been published. Many blogs and podcasts use RSS feeds to notify their readers of new content, which are the topic of the next section.

Once you get used to browsing the internet in plain text then you might like to set EWW as the default browser to follow links in Emacs. To make this change, customise the `browse-url-browser-function` variable and select 'Emacs Web Wowser' in the value menu.

You can read the complete EWW manual with `C-h R eww`.

4.3.1 Read RSS and Atom Feeds with Elfeed

Finding interesting content on the internet can be like sifting through piles of garbage to discover something valuable. Social media can be fun and engaging, but the cacophony of irrelevant and abusive content driven by dark algorithms is disheartening. RSS and Atom feeds enable subscriptions to the websites and blogs you enjoy. A feed is an XML file containing recent content from a website, either the complete text or just an excerpt.

RSS (Really Simple Syndication) is an elegant mechanism for consuming content because you only see the blogs or podcasts you subscribe to. When you use RSS, no algorithm decides what you can and cannot see. Subscribing to RSS feeds is anonymous, so you will not be spammed with

email funnels trying to sell you stuff or services. Some websites have multiple feeds, so readers can subscribe to specific topics.

Atom feeds are a newer feed format that clarifies some of the ambiguities in RSS. Both feeds are a form of XML, and you need an aggregator to display their content. Unfortunately, RSS and Atom feeds have lost importance due to social media dominance and website owners' preference to collect email addresses. However, the technology is still alive and used in almost all websites, including podcasts and YouTube. Browsers no longer link to the feeds automatically, and websites rarely prominently link to them like they used to, but the feeds still exist. For example, the RSS feed for Emacs articles on the accompanying website for this book is:

```
https://lucidmanager.org/tags/emacs/index.xml
```

The Elfeed package by Christopher Wellons aggregates your favourite RSS feeds. You can list and categorise your favourite feeds. The Elfeed browser helps you navigate your unread articles, YouTube feeds, or podcasts. You must install the cURL program, which stands for 'Client for URLs'. This program assists with downloading files from the internet. If cURL is unavailable, then Elfeed uses the slower built-in Emacs method to extract data, which does not work on Windows computers.

The package creates a database to store the feeds. EWS sets the location of the downloaded content to your Emacs configuration folder instead of your home folder. The EWS keyboard shortcut to start Elfeed is `C-c w e`. But before reading feeds, you must first find some and add these to a configuration file and download the data.

Finding RSS feeds used to be easy, but large internet companies prefer to rely on their black-box algorithms to feed content to users so RSS feeds are all but invisible. Some websites still use the RSS logo to offer feeds. When this is not the case, you can still find what you need. Almost half of the world's of websites use WordPress. You can find the feed for these sites by adding `feed` to the end of the URL. If all else fails, you can find the feed by looking at the page source (use the `v` key when viewing the page in EWW). Don't let the HTML code scare you. Search for `rss-xml` and copy the URL in the `href` specification. To add YouTube channels to your feed, add the channel ID to the URL `https://www.youtube.com/feeds/videos.xml?channel_id=<ID>`.

The basic configuration for Elfeed includes setting the `elfeed-feeds` variable to a list of RSS feeds. However, there is a more convenient way to manage your collection of RSS feeds. The Elfeed-Org package lets you configure your list of favourite websites in an Org mode file. The package reads the nominated Org mode file(s) and collects internet addresses or links from the headers with the `:elfeed:` tag. You set a tag for an Org mode header with `C-c C-q`. The example below shows how you can structure your Elfeed Org mode file. Note that a tag applies to all headings at a lower level, so the `:elfeed:` tag also applies to the Emacs and news headings. You can also add text comments, as Elfeed only reads headings.

```
#+title: Elfeed configuration

* Feeds                                    :elfeed:
** Emacs                                   :emacs:
Emacs-related information.
*** https://lucidmanager.org/tags/emacs/index.xml
*** http://www.reddit.com/r/emacs/.rss
*** https://www.youtube.com/feeds/videos.xml?channel_id=UCEqYjPJdmEcUVfHmQwJVM9A

** News                                    :news:
*** [[https://www.abc.net.au/news/feed/2942460/rss.xml][ABC Australia]]
```

You can either use a plain URL or an Org mode hyperlink. A hyperlink in Org mode consists of

a nested set of square brackets `[[link][description]]`. In an Org mode buffer, the link looks like a traditionally-formatted hyperlink. You can insert a link with the `org-insert-link` function (`C-c C-l`), paste the feed URL with `C-y` and give it a name. For further convenience, The Org-Webtools package inserts fully formatted hyperlinks into Org mode. The `org-web-tools-insert-link-for-url` function (`C-c w w`) constructs an Org mode link from a web address copied into the kill-ring and extracts the link title from that website. When using the EWW browser you copy the current address to the kill ring with `w`.

The only configuration you need for `elfeed-org` is to customise the name of the Org file(s) you like to use to store your feed links. In EWS, the location of the Elfeed configuration is stored in the `rmh-elfeed-org-files` variable, which by default is set to `"~/Documents/elfeed.org"`.

You need to customise this variable to match the file you like to us for Elfeed and then restart Emacs. To add or remove a feed, edit this file and update the database with `elfeed-update`. You are now ready to read your RSS feeds.

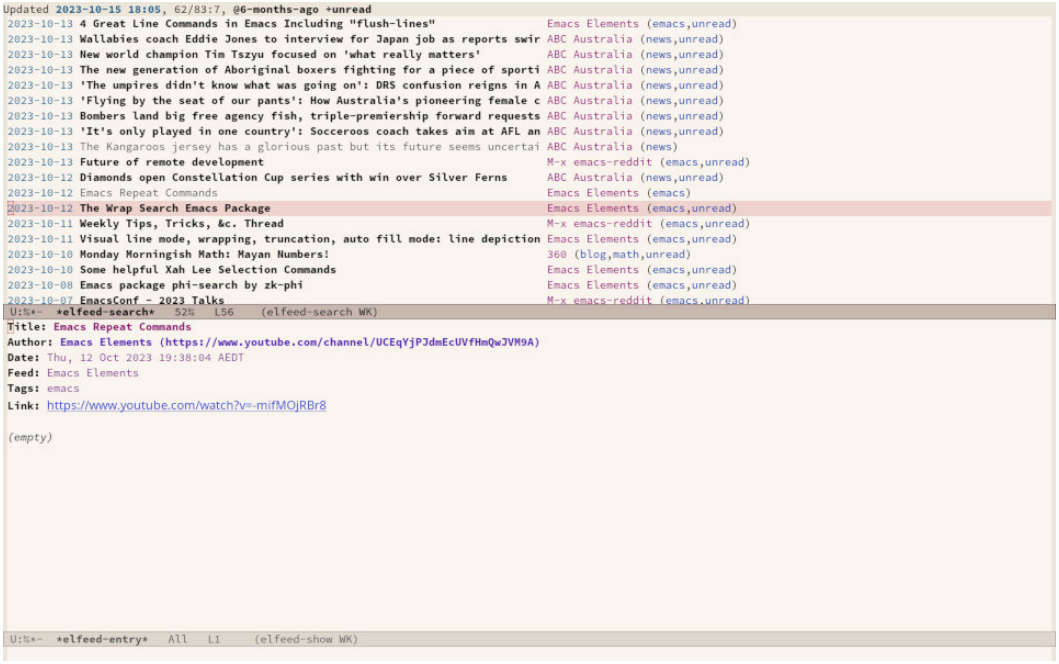

Figure 4.3 Elfeed screendump.

Press `C-c w e` to start the Elfeed browser, which shows a list with the date and title of each entry, the feed's name and any tags. When you hit `enter`, Elfeed displays the webpage or a summary with a hyperlink to the web version in another window. You can use the following keystrokes to manage your feed:

- `G`: Fetch feed updates from the servers
- `b`: Open the article in the system browser
- `c`: Clear the search filter
- `g`: Refresh view of the feed listing (remove unread items)
- `q`: Quit Elfeed
- `r` Mark the entry as read

- `s`: Update the search filter
- `u`: Mark the entry as unread

All new entries are tagged as unread by default. The other tags derive from your list of RSS feeds. When you remove a feed from your list, all articles that you previously downloaded will remain in the database and will show on your list until you read or remove them. Elfeed also has a powerful search filter that can be used to filter by tag, feed name, and dates.

4.4 Emacs Multimedia System

Music is a great tool for boosting productivity. Playing J.S. Bach' transcendental fugues or Sepultura's polychromic metal soothes the soul while writing you next great work of art. Psychologist Sara Bottiroli and her colleges studied the psychological effect of background music and found that it can improve episodic memory, intelligence, and verbal and visual processing speeds (Bottiroli et al., 2014).

Emacs might be a humble text processor, but it can also facilitate playing background music while you write, listen to podcasts or your field interviews. You might already have a great music player on your computer, but not having to switch applications to play music helps to retain your focus. The Emacs Multimedia System (EMMS), is a comprehensive music and video player for Emacs. It contains an intuitive browser displaying album covers and metadata, converting your Emacs system into a personal jukebox.

Emacs needs access to an external music player to produce sound and access to image software to convert album covers to thumbnails. To play sounds with Emacs, you must install one of the compatible sound players (ogg123, mpg321, MPlayer, MPV, or VLC). When you run the Emacs Writing Studio configuration, the system checks whether one of these players is available. Any missing software is listed in the messages buffer, which you can access with `C-h e`.

The last step in playing music is caching the music files. Evaluate the `emms-add-directory-tree` command to scan your collection. EMMS creates a cache in your Emacs configuration directory. EMMS reads metadata in music files for Ogg Vorbis, Opus, FLAC and MP3 files and some video file types. This process also caches thumbnails and might take a while, depending on the size of your collection. You can start playing music when `EMMS: All track information loaded` shows in the minibuffer. You can see the progress of the scan in the Messages buffer.

The basic principle to play music files is that you move tracks to the playlist buffer. The browser is the most convenient way to select the music of your liking. You start the browser with `emms-browser` (`C-c w m b`). Loading the browser for the first time in an Emacs session might take a moment if you have an extensive music collection.

When in the EMMS browser, use the `b` key followed by a number to browse by artist, album, genre, year, composer or performer. The browser is collapsed by default, showing only album covers, artist names, or whatever you select as the browsing category. The browser is hierarchical. For example, when browsing by genre, the hierarchy has four levels: *Genre* > *Artist* > *Album* > *Track*. Use the `1` to collapse and the `2-4` keys to expand the categories at levels 1 to 3.

As is the case in most Emacs applications, press `?` for a list of available keystrokes, such as:

- `r`: Jump to a random entry
- `s`: Search by album, artist, title, performer or composer
- `ENTER`: Add selection to playlist

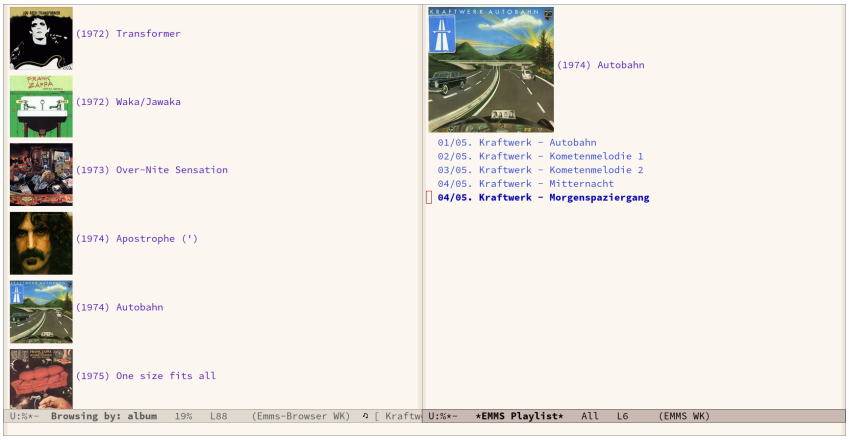

Figure 4.4 Screenshot of the EMMS browser.

- `C-j`: Add selection to playlist and play
- `W`: Lookup entry on Wikipedia

Being an Emacs buffer, standard search commands will also work for visible entries. When the cursor is on a category, such as an album name or a composer, it will add all tracks belonging to that category to the playlist. So when the cursor is on an album, it will add all tracks. When the cursor is on a single track, it will only add that track to the playlist. Jump to the playlist with `emms` (`C-c w m e`), from where you can manage what you play. Press `?` for a list of keyboard shortcuts, some of which are:

- `n` / `p`: Next or previous track
- `r`: Play a random track.
- `D`: Remove selected track from playlist
- `C`: Clear playlist

The EWS configuration also sets the multimedia buttons on your keyboard (play, pause, next, previous). The MPRIS (Media Player Remote Interfacing Specification) extension ensures that these buttons also work when playing music with EMMS but are not inside Emacs.

Once you have curated a funky playlist, keep it for future reference in `m3u` or `pls` format for use in Emacs or other multimedia players using the `save-buffer` command (`C-x C-s`). The `emms-play-playlist` function (`C-c w m p`) lets you load and play a saved playlist.

EMMS has many more advanced features that allow you to control your sound collection. Some other useful EMMS functions are:

- `emms-play-directory`: Add a directory to the playlist.
- `emms-play-find`: Plays all files in the music directory that match a given search criterion.

The Emacs Multimedia System has a plethora of additional options to fine-tune your listening experience. Read the EMMS manual with the info browser (`C-h R emms ENTER`) for detailed information about the various options.

Now that you know how to inspire yourself with Emacs, it is time to solidify your thought by writing them into an electronic notebook. The next chapter explains how to use *Emacs Writing Studio* as your personal knowledge management system.

5

Ideation: Record and Manage Ideas

Before the invention of writing, people memorised everything they needed to know. People from these prehistoric cultures recited thousands of lines of text from the Iliad, the Mahabharata, and other epics from memory, aided through songs and rhyme. Words that rhyme are much easier to remember than plain prose. Think about the song lyrics you can remember flawlessly but struggle to memorise a shopping list.

When writing first became common, Greek philosopher Socrates lamented in his dialogue with *Phaedrus* that written language erodes our memory. Socrates' argument was perhaps correct. However, writing has freed our minds from being a storehouse of factual knowledge to being a creative machine. The development of humanity accelerated when writing allowed people to free their minds from facts and use this brainpower to create new ideas. Productivity guru David Allen expressed it succinctly when he wrote, "The mind is for having ideas, not for holding them" (Allen, 2005). Philosophers refer to Allen's intuition as the *extended mind thesis*, which suggests that the mind does not exclusively reside in the body but extends into the physical world (Clark & Chalmers, 1998), which led to the idea of the second brain.

Taking notes to extend the fragility of human memory is as old as the art of writing itself. Since the 1970s, humanity has transitioned from writing words on paper to keeping them in electric storage. In recent years, the need to keep notes has resulted in a cottage industry of blogs, books, YouTube channels and electronic note-taking applications to help people organise their mind and life. Whether Luhmann's *Zettelkasten*, Carroll's *Bullet Journal*, or Forte's *Second Brain*, they all profess to solve personal knowledge management challenges by following a specified method. The best note-taking method is the one you develop yourself and that organically grows in complexity. An electronic second brain doe snot automatically lead to success, your organic brain should dictate the second one and not vice versa (Stavrou, 2024).

In my three decades working with both paper and digital journals, I've learned that the structure of notes is not the most critical aspect. What truly matters is the authenticity and originality of the recorded thoughts. Even Luhmann, the creator of the Zettelkasten method, saw his system as a "septic tank for ideas"[1]. However, we can apply some logic to our workflow to create order in the chaos, and Emacs provides powerful tools to assist.

Using Emacs for taking notes, or any other text processor, does have limitations. My personal ideation process starts with a physical notebook. This might be a strange thing to admit by somebody who professes admiration for the electronic virtues of Emacs. Physical notebooks have some advantages in the creative process. Firstly, you can use them everywhere without batteries. The only exception might be in the shower, paradoxically, the place where we get our best ideas. Using a notebook is also a slower process than using a keyboard, which might seem a disadvantage, but using a pen forces you to think more deeply. Writing on paper enhances creativity by triggering deeper neural pathways than writing electronically (Mueller & Oppenheimer, 2014; Umejima et al., 2021). Writing on paper also makes it easy to combine graphics with text. Doodling and sketching are great ways to conceptualise knowledge, which is one thing a text processor such as Emacs cannot provide.

Putting the disadvantages of textual notes aside, electronic notes definitely have a use. Once

[1] Niklas Luhmann-Archive, ZK II Zettel 9/8a2 (`niklas-luhmann-archiv.de`).

my notebook's ideas germinate, some find a place inside my Emacs filing system. My workflow also includes scans of sketches in my notebook, photographs, videos and electronic files emailed to me. All files live in a system which makes it is easy to find information and germinate new ideas. This workflow allows me to embrace the flexibility of working on paper and harness the power of electronic information.

The ample literature on taking notes comes with the empty promise of guaranteed success when you follow their approach. These rigid workflows inevitably lead to failure because everybody has different needs. When taking notes creatively, the formal workflow only matters little. As you develop your notes, your own personal methodology emerges organically. So, rather than worrying about some method promoted on the internet, start writing and create a process that best suits your needs.

The practical reality of taking notes is that we can distinguish between two categories: fleeting and permanent notes (Ahrens, 2017). A fleeting note is a quick notation that might be only temporary. You take fleeting notes on a napkin or the back of your hand. Most fleeting notes have a short lifespan, but some are promoted to a permanent status. As the name suggests, a permanent note is information you like to keep in perpetuity. You permanent notes form your personal wiki, second brain, Zettelkasten, digital garden or whatever neologism you might like to use.

The next section of this chapter explains how to use Org mode as a frictionless note-taking tool, using a single file to capture your thoughts. The remainder of the chapter is dedicated to the Denote package by Protesilaos (Prot) Stavrou. This package provides flexibility in managing an extensive collection of notes and other files to develop an interconnected second brain, using your chosen workflow.

5.1 Fleeting Notes

Fleeting notes capture those unexpected ideas that flit through our minds — a sudden insight, a movie quote, or a to-do list reminder. These fleeting notes are temporary parking spots for thoughts. They might be ideas for a future project, tasks to complete, or something interesting to revisit later. We need a frictionless capture system, like a trusty paper notebook, a phone app, or even the back of our hand to prevent these from disappearing.

the need to take fleeting notes also arises while using Emacs. You can leverage Org mode's capture feature for this purpose. Imagine writing a book when you suddenly remind yourself that you need to buy some milk; Org mode in Emacs lets you capture this fleeting thought with just a few keystrokes, saving it for later review without derailing your current focus.

Capturing a fleeting note with Org mode's capture feature is a simple process. Just press `C-c c` (org-capture), and a selection screen pops up. Select 'Fleeting Note' with `f`, write your thoughts into the popup buffer, and press `C-c C-c` to save the note under the 'Notes' heading in your inbox as a list item. The capture system adds new fleeting notes below the previous ones. If you decide it was not worth storing this thought after all, press `C-c C-k` to cancel the input. Once you're done, Org mode returns to where you left off, and you can happily proceed with your work with minimal disturbance. The capture menu also has an option to add an item to your todo list, which is stored in the same file, but under a different heading. Chapter discusses how to manage projects and action lists in Org mode.

The `org-default-notes-file` variable defines the inbox. By default, this variable is `~/Documents/inbox.org` in EWS. You can customise this variable to match your preferred inbox file. The file is automatically created when you first use the capture mechanism. You are, of course, free to directly add other information to this file. The capture mechanism will search for the defined headers and place any new items underneath them.

As you create more fleeting notes, your inbox steadily fills with random musings. To keep your inbox as empty as possible, it's a good habit to develop a weekly review. This review involves converting promising thoughts to a permanent note or trashing them after they expire. Ideally, your Inbox should trend towards zero content, as discussed in chapter . You can use the universal argument with the org-capture command (`C-u C-c c`) to jump to the file for your chosen template.

The Org mode capture functionality is a versatile system that allows you to craft templates for different types of notes. You can develop bespoke templates that capture information into various files and include metadata, such as timestamps and other properties. The Appendix explains how to modify the capture system to meet your needs.

5.2 Permanent Notes

Your notes are a treasure trove that slowly develops over the years. Permanent notes in Emacs form an external electronic storehouse of information that you can structure and search to create new insights.

Many different ways exist in Org mode to store permanent notes. Don't worry too much about which method to use when you start your collection of electronic notes. You can start with a single file and start writing. Your ideal workflow will emerge from your personal needs as they arise. The key to writing good notes is not worrying about the second brain, as your first brain is much more important in the creative process (Stavrou, 2024).

First, we explore using a single Org mode file to store permanent notes. Create an Org mode file, give it a suitable title, and start writing. Use a heading for each note to add a succinct title. You can also group your notes by using level one headings as categories and lower levels for the actual notes. If you want to add a timestamp to record when you took the note, use the `org-time-stamp` function. Calling this function with `C-c .` adds a date, and with a universal argument, the time is also included (`C-u C-c .`). A note under the philosophy category could look something like this:

```
* Philosophy
** Socrates against writing
   <2024-04-20 Sat>
   In the Phaedrus, ...
```

A few tools in Emacs and Org mode help to manage an extensive collection of notes in a single file. To view the table of contents of a file when you open it, add `#+startup: content` to your front matter. With this keyword, Org mode only shows headings when the file is first opened. Org mode's ability to fold and unfold headings with `S-TAB` lets you focus on what is essential.

Another method to create focus within large files is to narrow the buffer to show only the section you are working in. The `C-x n` prefix key brings you to the narrowing functionality. The `org-narrow-to-subtree` function (`C-x n s`) narrows the current buffer to only show the content of the section that the cursor is in. The other text is not erased; it is just hidden from view. To revert to the complete buffer, use the `widen` command, bound to `C-x n w`. The narrowing functionality has a few other options, which you can explore through the popup menu when you invoke the prefix key.

Another method to focus on relevant parts of your document is to construct a sparse tree with the `org-occur` function, evoked with `C-c / /`. Sparse trees provide filtered views based on search criteria and highlight relevant text while hiding unrelated content. After entering the search criterion, Org mode collapses all headings, highlights the requested words and only shows the sections where the search term occurs. Two shortcuts let you jump between the matches: `M-g n` jumps to the next match and `M-g p` to the previous one. Using any editing command or `C-c C-c` exits the search.

The main difference between a sparse tree and the regular search functionality (section 2.7.2) is that a sparse tree collapses your document to only show the sections where the search occurs.

Adding notes to categories by structuring headings is helpful but limited because a note can only be part of one category. Org mode can also add tags to each heading to relate notes to each other. A tag is a label or a category for a headline that helps to group related headings. Tags appear after heading text, nested between colons. Tags are inherited properties, meaning any tag at a level one heading also belongs to the relevant subheadings. So, in the example below, all subheadings under the philosophy heading inherit the `:philosophy:` tag. Any subheadings under the note about Socrates will also inherit both the `:philosophy:` and `:writing:` tags.

```
* Philosophy                          :philosophy:
** Socrates against writing           :writing:
   <2024-04-20 Sat>
   In the Phaedrus, ...
```

You add a tag to a note with `C-c C-q` (`org-set-tags-command`). Type the name of the new tag in the minibuffer. Any tags already used in the document are displayed in the minibuffer completion list. You can also set a library for each file by adding something like this to the front matter of the Org file: `#+tags: philosophy(p) writing(w)`. The letters between parentheses become a shortcut in the minibuffer menu for fast selection. To create a new tag, just type free text into the minibuffer. Once you have a file with tagged entries, you can use this to search notes by category using the *sparse trees* functionality. To select one or more tags for a sparse tree, use `org-match-sparse-tree` (`C-c \`). This function collapses the whole document and highlights the sections where the selected tags occur.

The Consult package by Daniel Mendler includes a convenient function to move around large Org mode files. The `consult-org-heading` function (`C-c w h`) provides al list of all headings in the current Org mode file in the minibuffer, from where you can navigate to the desired location. The Consult package provides a broad range of search and navigation commands to improve the way you use Emacs.

A further tool to manage large files is the `org-refile` function, bound to `C-c C-w`. This command lets you easily move sections to another section in your document. When evoking this function, a list of names of chapters (level 1 headings) in your document appears. The subtree that the cursor is currently in will move to the selected chapter. To jump to the relevant entry, use the `C-u C-u C-c C-w` shortcut (two universal arguments before the command). Another method to refile your headings is with the Alt and arrow keys, as explained in section 3.4.1.

Lastly, you might want to create links between notes in a file. we have already seen file links in section 3.4.5, but we can also link o a heading within an Org mode file. The easiest way is to create an internal link with `C-c l` and enter the name of the heading you need to link to and add a description. The link now looks something like this:

```
[[* Heading name][Description]]
```

The problem with this approach is that the name of the heading might change or perhaps you misspelled it. When following a link to a non existing target Org mode does not throw an error but asks whether you like to create a new heading. A better approach to linking is giving the heading a unique ID. EWS is configured so that Org mode create a unique ID for a heading. To create a link between notes in a single note document, move the cursor to the heading you like to link to and press `C-c l` (`org-store-link`). This function create a drawer underneath the heading. , A drawer is a section of collapsible text that can store metadata about a heading. Drawers are useful for a lot of tasks, and are further discussed in chapter 6. The drawer might look something like this:

```
:PROPERTIES:
:ID:       d454979b-2d40-4f95-9f85-f5d9314c28d7
:END:
```

The random string of number is a Universally Unique Identifier (UUI), which creates a random ID. The likelihood of a duplicate ID is astronomically small so we can consider it unique. A link to this ID is now stored in memory and you can insert it where you need it with `C-c =C-l` (`org-store-link`). A link to an ID looks like this under the hood:

```
[[id:d454979b-2d40-4f95-9f85-f5d9314c28d7][Example]]
```

Using one large file for your notes is a great way to commit your thoughts to Emacs. However, the file can become unwieldy after a while, and if you get really productive, a large file can slow down the system. The next section shows how to use the Denote package to write your notes as a collection of interconnected notes.

5.3 The Denote Package

Most note-taking systems use separate files for linked individual notes to create a network of ideas. Emacs users have developed a slew of packages to write and manage collections of notes. EWS uses the Denote package. This package does not enforce any methodology or workflow, and it can process both written notes in plain text and binary files, such as photographs, PDF files, or anything else you would like to archive.

The Denote package categorises your files using keywords. There is also an option to add a signature, which designates a sematic order for notes. Notes can also link to each other to form a network of thoughts. With these three mechanisms, you can use Denote to create an organic digital garden or implement a formal system, such as Zettelkasten, Johnny Decimal, PARA or methods that don't even exist yet.

The driving force of the Denote package is its file naming convention. This approach embeds metadata in the filename, so there is no need for a database or any other external dependency to navigate your jungle of notes. The Denote file naming convention has five parts (all in lowercase by default), of which only the ID and file extensions are compulsory. An example of a fully formatted Denote file is.

```
20210509T082300==9a12--simultaneous-contrast__colour_illusions.org
```

1. Unique identifier (ID) in ISO 8601 time format with a `T` separating date and time (`20210509T082300`).
2. Signature (lowercase letters and numbers), starting with a double equals sign (`==9a12`).
3. Title separated by dashes (kebab-case), starting with a double dash (`--simultaneous-contrast`).
4. Keywords separated by an underscore ($\text{snake}_{\text{case}}$), starting with a double underscore (`__colour_illusions`).
5. Filename extension (`.org`).

The timestamp is a unique and immutable identifier Denote uses to link notes. The timestamp orders our notes chronologically, but that might not always be the best option. The signature lets you order your notes just like the Dewey Decimal System orders books on the shelves of a physical library. The keywords or file tags define the categories that a note belongs to. This part of the Denote structure lets you group notes that have something in common. The signature, title and tags are flexible and can change over time. The timestamp should always stay the same to maintain the integrity of links between notes.

The file naming convention in Denote places some limitations. The Denote signature can only contain letters, numbers and the equals sign to maintain the integrity of the system. The title only

letter, numbers and dashes. Keywords can only contain letters and numbers. Denote cleans (sluggifies) file names to enforce compliance with the convention.

Denote stored new notes in the same folder, signified by the `denote-directory` variable, which defaults to `~/Documents/notes`. You can customise this variable to suit your needs and restart Emacs. Denote can store notes in subdirectories within this primary location, but there is no need to do so. When subdirectories categorise files, a part of the metadata for that file disappears as soon as you move the file to another location. Also, a file can only reside in one directory.

Modern operating systems can effortlessly manage tens of thousands of files in one directory, so there is no need to use subdirectories. Instead of subdirectories you can use file tags, which makes it easy to view files that logically belong in the same group. File tags are more flexible than subdirectories because each file can have more than one tag, but can only live in one directory.

The `denote` function, which you activate with `C-c w d n`, creates new notes as Org mode files. It first asks for a title and then for the relevant keywords. You either select a keyword from the completion list of existing notes in the minibuffer with the `TAB` key or enter new ones as free text, separated by a comma. The timestamp is automatically generated using the date and time you create the note. You can also activate this command with the Org capture system and select 'Permanent Note'.

Note that when creating a new note, it first opens as an unsaved buffer. You will need to save it to disk with `C-c C-s` to make it permanent. When you create a permanent note with the Org capture mechanism saves the note when exiting the capture popup screen with `C-c C-c`. Some functionality might not work unless you have saved the note to disk,so if you get a warning that says "Buffer not visiting a Denote file", you might have to save the buffer first so the software can recognise it.

The default EWS configuration does not require a signature or a subdirectory for new notes. You can customise the `denote-prompts` variable to define the default way Denote generates and renames files by ticking the items you like to include when creating an new note.

The note's title becomes part of the filename and becomes the title inside the Org mode file. The date and identifier are also part of the header of the file. Keywords become file tags, which are the same as tags we saw in the section about using a single file, but they apply to the whole file. The front matter for the Org mode file of the note in the example above would look like this. Now, all you have to do is fill the buffer with relevant content and save it to disk.

```
#+title:      Simultaneous Contrast
#+date:       [2021-05-09 Sun 08:23]
#+filetags:   :colour:illusions:
#+identifier: 20210509T082300
```

Not all permanent notes are created equal and the relevant workflow within Denote depends on their purpose. Broadly speaking, we can distinguish between five types:

1. *Generic notes*: General ideas.
2. *Journal entries*: Experiences related to a specific time.
3. *Literature notes*: Notes about a publication.
4. *Attachments*: Read-only notes, such as photographs or PDF files.
5. *Meta notes*: Notes that link to all notes meeting a search criteria.

The first type of note can be anything you want it to be, so there is nothing particular to discuss. The other permanent note types are discussed in the next four sections.

5.3.1 Keep a Journal or Diary

You can use Denote for personal reflection, to create a journal or a laboratory logbook, to add meeting notes, or to record any other notes related to an event. Writing a journal with Denote is easy

because the identifier for each note indicates the date and time you created it. Adding a standard tag, such as `_journal`, makes your journal entries easy to distinguish from other notes.

If you create a note for an entry in the past, use the `denote-date` function (`C-c w d d`). You enter the date in Year-Month-Day (ISO 8601) notation like `2023-09-06`. Optionally, you can add a specific time in 24-hour notation, for example `2023-09-6 20:30`. Denote uses the present date or time if no date and/or time is provided. It makes sense to tag each journal entry with a standard keyword, e.g. `_journal`, or whatever makes sense in your native language for easy reference.

5.3.2 Literature Notes

Writing helpful permanent notes is a craft in itself. Copying and pasting content from websites or other literature you read is only very useful if you need to memorise text for an exam. You should write notes in your own words for your future self to read. This means that you should include sufficient context so that you will still understand your notes when looking at them later.

A literature or bibliographic note contains a summary or an interpretation of the literature you read. For that purpose, the Citar-Denote package provides functionality to link notes to a bibliography. A bibliographic note is a special category of permanent notes that link to one or more publications in your bibliography. The Citar-Denote package integrates the Citar bibliography package with the Denote note-taking system. This package provides extended functionality to create and manage bibliographic notes. Refer to chapter 4 to find out how to create a bibliography and use Citar.

Citar-Denote allows a many-to-many relationship between notes and entries in your BibTeX files, providing a complete solution for documenting literature notes. This means you can add multiple notes per bibliographic entry or one note for more than one piece of literature. You could create a note about each book chapter or write a single literature review note for a collection of journal articles, whatever suits your workflow best.

Citar-Denote relates a note to an entry in your bibliography by using the citation key as a reference in the front matter. Each bibliographic note is also marked with the `_bib` file tag. This tag reduces the number of Denote files the system needs to track. The front matter for a bibliographic note could look something like this:

```
#+title:      Marcuse: An Essay on Liberation
#+date:       [2022-11-12 Sat 19:23]
#+filetags:   :bib:culture:marketing:philosophy:
#+identifier: 20221112T192310
#+reference:  marcuse_1969_essay
```

Open the Citar interface with `C-c w b c` (`citar-create-note`) to create a new note. Select the entry you want to write a note for, hit `ENTER` and follow the prompts (Figure 4.2). You can create additional notes for this entry if a note already exists. A note can have more than one reference. The `TAB` key lets you select multiple entries.

Once you have collected some bibliographic notes, you will want to access and modify them. You can access the attachments, links and other notes associated with the references from within via the Citar menu with `C-c w b o` (`citar-open`). Entries with a note are indicated with an `N` in the third column. The citar menu provides access to attachments, notes and links. From this menu, you can also create additional notes. To only show those entries with a note, start the search with `:n` or use `citar-denote-open-note` (`C-c w b n`) to open the Citar menu with only entries with one or more associated notes.

Several functions are available to manage the current buffer when inside a bibliographic note. The `citar-denote-dwim` function (`C-c w b d`) provides access to the Citar menu for the referenced literature in this note, from where you can open attachments, other notes, and links. The `citar-denote-add-citekey` function (`C-c w b k`) adds citation keys or converts an existing Denote file

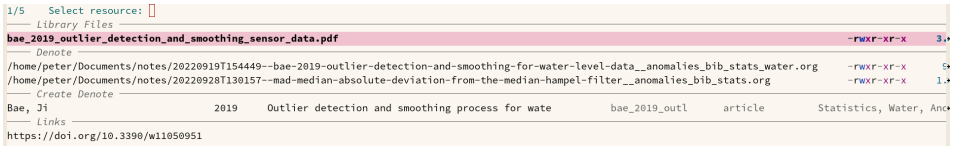

Figure 5.1 Screenshot of Citar and options for one of the entries.

to a bibliographic note. When converting a regular Denote file, the function adds the `bib` keyword to the front matter and renames the file accordingly. This function opens the Citar selection menu and adds the selected citation keys to the front matter. You remove citation references with the `citar-denote-remove-citekey` command (`C-c w b K`). If more than one piece of literature is referenced, select the unwanted item in the minibuffer first. When the note only has one reference, the bibliography keyword is removed, and the file is renamed, converting it to a generic permanent note.

What is the point of building a bibliography without citing or using each in a bibliographic note? The last two functions let you cite literature or create a new bibliographic note for any item not used in your Denote collection. The `citar-denote-nocite` (`C-c w b x`) function opens the Citar menu and shows all items in your bibliography that are neither cited nor referenced. From there, you can create a new bibliographic note, follow a link or read the file.

When writing literature notes you might have the note open in one window and the ebook or other reference in another window. You can move pages in the document you are reading without moving the cursor away from the note you are wiring. Adding the Alt key to the page-up or down key will scroll the other window up and down without needing to move your cursor.

Emacs has a large collection of commands that can act on other windows. If you like to explore other options, the type `C-h x other-window` and the completing system will show all commands that match this criteria and the Marginalia package provides a short description and any available keyboard shortcuts. This method is a productive way to explore Emacs commands that you might not know about yet.

5.3.3 Attachments

Denote's reliance on a filename to store metadata allows you to manage files other than the three types specified by Denote, which we can call attachments. An attachment in a file with a compatible filename, except those files that Denote creates. Denote recognises any file stored in the Denote directory that follows the it's file naming convention, which can be linked inside a Denote note.

Your digital notes garden can be much more than just text. You can manage your photographs with Denote and store an archive of PDF files, such as bank statements, course certificates, or scans of your paper archive. Extending Denote with attachments converts your list of notes to a complete personal knowledge management system with easy heuristics to find your documents and link them to notes.

There are numerous use cases for extending Denote to binary files. I personally save my photographs and videos in the Denote file format. I also store PDF files, such as scanned paper documents or files I receive in this format, such as invoices. You can also link to individual attachments inside a Denote file in the same way you link your notes. However, it is not possible to link back from an attachment to a note using Denote, as these files are not notes.

The first step to register an attachment in Denote is to ensure it has a compliant and correct name. You can either rename a file manually or use the denote-rename-file function (`C-c w d r`) within the Dired file manager (section). This function uses the filename as a default title, which you can modify as needed and you can add relevant keywords. The last modified timestamp of the file will serve as its identifier.

However, the creation date on the file system is not always the actual creation date. When working with attachments, there are three options for a valid timestamp, the date and time when the:

- Digitised paper document was created
- Electronic file was born (first creation date)
- Electronic file was created on the file system (Denote default)

Using the universal argument `C-u` instructs denote-rename-file to ask the user to enter a date and/or time for the attachment. So, in Emacs Writing Studio, use `C-u C-c d r` to rename a file from Dired and manually provide a timestamp. You must enter the timestamp in ISO 8601 format, for example: `2023-11-04 12:32:45`. you can also enter it as a Denote timestamp: `20231104T113245`.

The first scenario mainly relates to historical documents. Over the years, I have gradually digitised my paper archives. The earliest identifier timestamp in my Denote library is `13700623T120000`, a scan of a medieval document that holds the mortgage of my birth house. The original creation date of the document (when it was scanned) is in 2021, and the date on my file system is sometime in 2023. The Denote renaming function uses the file system date, which is not ideal. This document requires manually entering a timestamp that places the document in the distant past.

The second scenario mainly occurs with photographs. The timestamp on the file system might not be the same as when the picture was taken. So, in this situation, we need to extract the creation date from the file. Several tools, such as ExifTool, are available to extract metadata from photographs and PDF files.

5.3.4 Meta Notes

Meta notes, the last type of permanent note, serve as a gateway to other notes on a similar topic. Dynamic blocks, a versatile feature in Org mode, excel at aggregating your thoughts and linking to relevant notes. These blocks, also known as meta notes, provide a seamless entry point to your note collection and serve as a starting point for your writing projects. A meta note could contain a list of all notes within a category or an ordered list of notes that match part of a signature.

Let's say that you are working on a project to write a paper about the *Daimonion* (prophetic monitor) that spoke to the ancient Greek philosopher Socrates. You read the literature and create a bunch of permanent and bibliographic notes that use the `_daimonion` keyword. When it is time to gather your thoughts into an integrated view, you can create a new note that links to all your relevant notes.

In Org mode, a dynamic block is a section in your document that can be updated using a function. Denote offers functions to add dynamic lists of links to other documents, which are incredibly handy as they always reflect the latest version of your note collection. To use this feature, simply invoke the denote-org-extras-dblock-insert-links function (`C-c w d D`), and provide a search string that matches the notes you want to list (in this case, it's `_daimonion`). Denote will insert a block in your note that lists all notes matching this search criterion. The dynamic block is enclosed between two lines: `#+BEGIN:` and `#+END:`. If you add new notes with the Daimonion tag, simply place the cursor on the beginning or end lines and use `C-c C-c` to update the list. To modify the search criteria, edit the text between the quotation marks after the `:regexp` part of the first line and update the dynamic block.

```
#+BEGIN: denote-links :regexp "_daimonion"
- Plato: Apology
- Socrates and Plato
- Plato: Crito
#+END:
```

A regular expression is an advanced search term much like using wildcard in filename but with much greater power. This approach could, for example, collate your journal notes for a particular month. Using the regular expression `^202309.*_journal` lists all journal entries for September 2023. This regular expression lists filenames that start with `202309` and include the `_journal` keyword. The tilde (`^`) denotes that you are searching at the start of the filename. The `.*` in the middle of the regular expression indicates that any character (`.`) can appear multiple times (`*`). Regular expressions are a powerful tool for searching, but detailed discussion is outside the scope of this book.

5.3.5 Linking Notes

The Denote signature and keyword offer a unique way to categorise and rank notes, ordering and grouping similar ideas. Additionally, Org mode can also become a personal wiki by linking notes. While the term 'personal wiki' may seem contradictory, given that wikis are by definition collaborative, the power of linking notes allows for the creation of an interconnected web of ideas.

Org mode features a versatile system of links. In the previous chapter, we explored how to add hyperlinks to the World Wide Web and links to images (section 4.3.1 and chapter 3). Org mode can recognise various types of hyperlinks, including links to other files, internal links, and other special formats.

To create a hyperlink from an Org mode file to another file use `C-u C-c C-l`, navigate to the target file and enter a description. The link is now active and You can follow it with `org-open-at-point`, bound to `C-c C-o`. Linking to other documents is a great way to structure your file system, but this method has a problem because the link breaks as soon as the target file changes name or location.

Denote enhances Org mode's functionality by creating stable links between notes. A denote link only stores the identifier of the target file, so the signature, name and keywords can change freely without the risk of creating dead links.

You can link notes and attachments with the `denote-link-or-create` function (`C-c w d i`). This command lists all available notes using the minibuffer completion system, from which you can select one and hit enter. To modify the link's label, press `C-c C-l` (`org-insert-link`) while the cursors is on the link and follow the prompts. The source of a Denote link looks something like this:

```
[[denote:20210208T150244][Description]].
```

If you enter a name for a note that does not yet exist, Denote will first let you create a new note and then link to it. Denote links are indicated with italics in EWS to distinguish them from links to other resources, such as websites.

Denote can also create links to headings inside of other Org mode notes. The `denote-org-extras-link-to-heading` function (`C-c w d h`). This function will assk for the Org mode file to link to and then shows a menu of the headings inside this file. When you now click on the new link, it will take you directly to the heading inside the other file.

You don't need to search through a document to find relevant links. Jump to any linked note without placing the cursor on it with `denote-link-find-file` (`C-c w d l`). This function shows all notes linked from the open note in the minibuffer where you can select the one you like to jump to. To find out which notes link to the one you are currently reading, use the `denote-find-backlink` function (`C-c w d b`).

5.3.6 Finding Notes and Attachments

When collecting thousands of notes and attachments, it is important to have pathways in the system to find the information you need or to make new connections between ideas. The most straightforward method to find files is opening one with the standard `find-file` function. The minibuffer completion system helps you to find what you need.

If, for example, you like to filter notes tagged as 'economics', type `C-x C-f`, move to your notes folder, and type `_economics`. Minibuffer completion narrows the available options. If you need a note with economics in the title but not as a tag, use `-economics`. If you type `economics` without a prefix, then you see all posts with the search term in the signature, title or as a tag. You can also introduce regular expressions (section 5.3.4) to increase your search power. As the minibuffer completion uses the Orderless package, a space effectively acts as an AND expression. So using `^2022 ===9a _art` searches for all notes with a file name that starts with (`^`) 2022 and include a signature that starts with 9a (`=9a`) and have the `_art` file tag.

The Consult-Notes package by Colin McLear merges the capabilities of Denote and Daniel Mendler's Consult package to help you find notes without remembering which directory they live in. This package also provides facilities to search through the content of your notes, with assistance from external software.

To find a note by any part of its filename, use the `consult-notes` function bound to `C-c w f`. You can use all the above methods to narrow your quest for the perfect note. The Consult package provides live previews of the files that match the search. These previews display the files in their raw form, without any image previews, font enhancements and so on, so previews load quickly.

The Consult-Notes package only searches in the nominated directories. The configuration for Consult-Notes can include your main Denote files folder and one or more other silos of files, giving you full access to all your content in a convenient interface. You can, for example, also add your photo or video archive to access all your content in one convenient interface.

To customise the sources that Consult-Notes uses, customise the `consult-notes-sources` variable. Click on the `INS` button to add a new entry. You need to enter a name for the file silo, a shortcut character and the directory. For example, adding `Photos`, `p` and `~/Pictures` adds your photographs to the collection. Click `Apply and Save` to store these settings. You can filter each section of your file collection by using `:n` for notes and, in the example above, `:p` for photos or any other letter you assign.

Searching for titles, tags, and other metadata is a powerful way to access your information. While that is a good start, sometimes, you will want to search through the content of your notes instead of just to titles and metadata. The `consult-notes-search-in-all-notes` function (`C-c w g`) activates a deep search. For this purpose, the package uses the Grep software, which needs to be available on your system.

The search is incremental, just like minibuffer completion. As you type your search criterion, a list of results appears. The results show the filename and the matching lines within each file. The search term starts with a hashtag; when you type another such symbol, for example, `#topology# ball`, the next part will be searched within the results.

5.4 Managing Your Digital Garden

Your collection of notes needs regular maintenance as ideas and structures of thought evolve over time. The names, keywords and signatures of notes can change over time as your digital garden grows and blossoms.

Denote Org mode files hold metadata in both the file name and the file's front matter. ideally you like the file's name and the front matter to be in sync. You can also change the title and the keywords by simply editing the text. For more convenience, use the `denote-keyword-add` (`C-c w d k`) and `denote-keyword-remove` (`C-c w d K`) functions to change tags with minibuffer completion. These last two functions will also rename the file. Using `denote-rename-file-using-front-matter` (`C-c w d R`) changes the filename using the data in the front matter. This function leaves the identifier unchanged, even when edited in the front matter.

The Denote-Explore package provides convenience functions to manage your collection of Denote files. You can find the shortcuts for the Denote-Explore package with the `C-c w x` prefix. This package provides four types of commands:

1. *Summary statistics*: Count notes, attachments and keywords.
2. *Random walks*: Generate new ideas using serendipity.
3. *Janitor*: Manage your denote collection.
4. *Visualisations*: Visualise your Denote files as a network.

5.4.1 Summary Statistics

After a day of working hard in your digital knowledge garden, you might like to count the notes and attachments in your collection. Numbers are great, but a graph is worth a thousand numbers. The built-in `chart.el` package by Eric M. Ludlam is a quaint tool for creating bar charts in a plain text buffer. Two commands are available in Denote-Explore to visualise basic statistics using `chart.el`:

1. `denote-explore-keywords-barchart`: Visualise the top *n* keywords.
2. `denote-explore-extensions-barchart`: Visualise used file extensions.

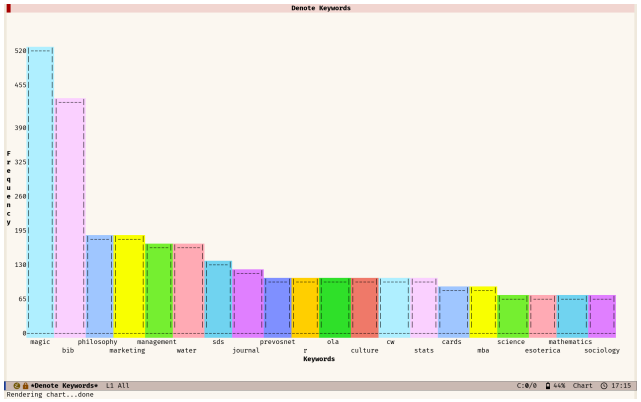

Figure 5.2 Example of a bar chart of top-twenty keywords.

5.4.2 Random Walks

Creativity springs from a medley of experiences, emotions, subconscious musings, and connecting random ideas. Intellectual serendipity transforms unrelated concepts into potentially new insights. Introducing random elements into the creative process generates avenues of thought you might not have travelled otherwise. This method can be beneficial when you're stuck in a rut or just like to walk through your files randomly. A random walk is an arbitrary sequence of events without a defined relationship between the steps. You take a random walk by jumping to an incidental note, connected or unconnected to the current buffer. The Denote-Explore package provides three commands to inject some randomness into your explorations:

1. `denote-explore-random-note`: Jump to a random note or attachment.
2. `denote-explore-random-link`: Jump to a random linked note (either forward or backwards) or attachments (forward only).

3. `denote-explore-random-keyword`: Jump to a random note or attachment with the same selected keyword(s).

The default state is that these three functions jump to any Denote text file (plain text, Markdown or Org-mode). The universal argument (`C-u`) includes attachments as candidates for a random jump.

When jumping to a random file with the same keyword(s), you can choose between one or more keywords from the current buffer. When the current buffer is not a Denote file, choose any available keyword(s) in your Denote collection. The asterisk symbol `*` selects all keywords in the completion list.

5.4.3 The Janitor

Just like any building needs a janitor to keep it clean and do some minor maintenance, your digital home also needs help. After using Denote for a while, you might need a janitor to keep your collection organised.

The Denote package prevents duplicate identifiers when creating a new note. However, sometimes, you might have to manually create a date and time for an old document where the creation date is different from the date on the file system, as explained in section 5.3.3. Adding the Denote identifier manually introduces a risk of duplication. Duplicates can also arise when exporting Denote Org mode files, as the exported files have the same file name but a different extension.

The `denote-explore-identify-duplicate-notes` command lists all duplicate identifiers in the minibuffer. Open the `*Messages*` buffer (`C-h e`) to copy the identifiers and correct the relevant files. Be careful when changing the identifier of a Denote file, as it can destroy the integrity of your links, so please ensure that the file you rename does not have any links pointing to it. Before changing an identifier, you can use `denote-find-backlink` (`C-c w d b`) to check whether a file has any links pointing to it.

With Denote-Explore, managing keywords is a breeze. The `denote-explore-single-keywords` command provides a comprehensive list of file tags that are only used once, making it easy to identify and address any issues. The list of single keywords is presented in the minibuffer, from where you can open the relevant note or attachment, streamlining your note management process.

You can also find notes or attachments without keywords with the `denote-explore-zero-keywords` command. This command presents all notes and attachments without keywords in the minibuffer, so you can open them and consider adding a keyword or leaving them as is.

You can remove or rename keywords with `denote-explore-rename-keyword`. Select one or more existing keywords from the completion list and enter the new name of the keyword(s). This function renames all chosen keywords to their new version. It removes the original keyword from all existing notes when you enter an empty string as the new keyword. This function cycles through all notes and attachments containing the selected keywords and asks for confirmation before making any changes. The new keyword list is stored alphabetically, and the front matter is synchronised with the file name.

Denote stores the metadata for each note in the filename using its ingenious file naming convention. Some of this metadata is copied to the front matter of a note, which can lead to differences between the two metadata sources. The `denote-explore-sync-metadata` function checks all notes and asks the user to rename any file where these two data sets are mismatched. The front matter data is the source of truth. This function also enforces the alphabetisation of keywords, which assists with finding notes.

5.5 Visualising Denote Networks

Committing your ideas to text requires a linear way of thinking, as you can only process one word at a time. In my paper journal, I often use diagrams, such as a mind map, rather than a narrative to relate my thoughts. Visual thinking is another way to approach your ideas, and one of the most common methods to visualise interlinked documents is in a network diagram.

Linking ideas in a network is not a modern tool. Medieval monks sketches diagrams in the margins of books hey read, connecting their short notes with lines. These diagrams are the source of the curly braces symbols `{}`, which originally indicated branching an idea into two new ones (Even-Ezra, 2021).

Viewing your thoughts as a network can help discover hitherto unseen connections between your thoughts. Visualising your Denote digital garden as a network can be helpful in your creative process. A network diagram has nodes (vertices) and edges. Each node represents a file in your Denote system, and each edge is a link between notes (figure).

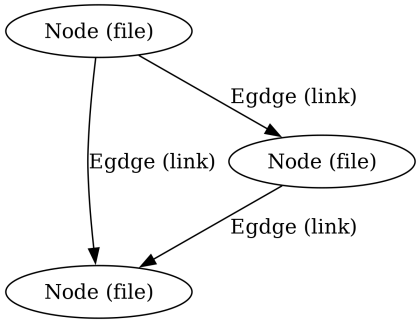

Figure 5.3 Principles of a Denote file network.

The Denote-Explore package uses the functionality provided by external software to visualise the structure of parts of your Denote network. Denote-Explore does not offer a live environment in which to view your Denote collection's structure. The purpose of this functionality is to analyse the structure of your notes, not to act as an alternative user interface. Live previews of note networks are dopamine traps. While seeing the network of your thoughts develop in front of your eyes is satisfying, it can also become a distraction.

The `denote-explore-network` command provides three types of network diagrams to explore the relationships between your thoughts:

1. Community of notes
2. Neighbourhood of a note
3. Keyword structure

A community consists of notes that match a search term, which can be a regular expression. The software asks to enter a search term or regular expression. For example, all notes with Emacs as their keyword (`_emacs`). The graph displays all notes matching the search term and their connections. Any links to non-matching notes are pruned (dotted line to the `_vim` note in the example in Figure 5.4). Using an empty regular expression will generate a network of all available files.

The neighbourhood of a note consists of all files linked to it at one or more steps deep. The algorithm selects members of the graph from linked and backlinked notes (such as C to A in figure 5.5). This visualisation creates the possible paths you can follow with the `denote-explore-random-link` function discussed in section 5.4.2.

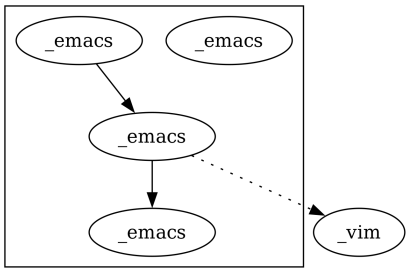

Figure 5.4 Community of Denote files with "Emacs" keyword.

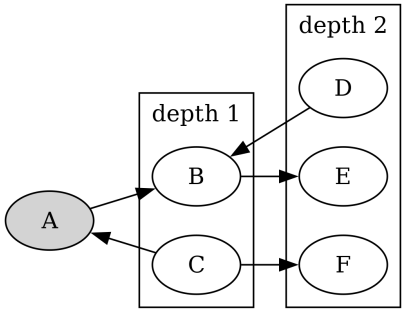

Figure 5.5 Denote neighbourhood of files (depth = 2).

To generate a neighbourhood graph from a Denote note buffer, use `denote-explore-network` and enter the graph's depth. When building this graph from a buffer that is not a Denote note, the system also asks to select a source note. A depth of more than three is usually not informative because the network becomes very large and complex to read, or you hit the edges of your island of connected notes.

There will be files without connections. Depending on your note-taking strategy, you might want all your notes linked to another note. The `denote-explore-isolated-notes` function provides a list in the minibuffer of all notes without links or backlinks for you to peruse. You can select any note and add any links. Calling this function with the universal argument `C-u` excludes attachments in the list of lonely files.

The last available method to visualise your Denote collection is to develop a network of keywords. Two keywords are connected when used in the same note. All keywords in a note create a complete network. A complete network is one where all nodes are linked to each other. The union of all complete networks from all files in your Denote collection defines the keywords network. The relationship between two keywords can exist in multiple notes, so the links between keywords are weighted. The line thickness between two keywords indicates the frequency (weight) of their relationship (Figure 5.6).

While the first two graph types are directed (arrows indicate the direction of links), the keyword network is undirected as these are bidirectional associations between keywords. The diagram below shows a situation with two nodes and three possible keywords and how they combine into a keyword network.

To visualise networks you need to install the Graphviz software. This tool converts plain text descriptions of a network into an image file. The network diagrams in this book are all created with GraphViz.

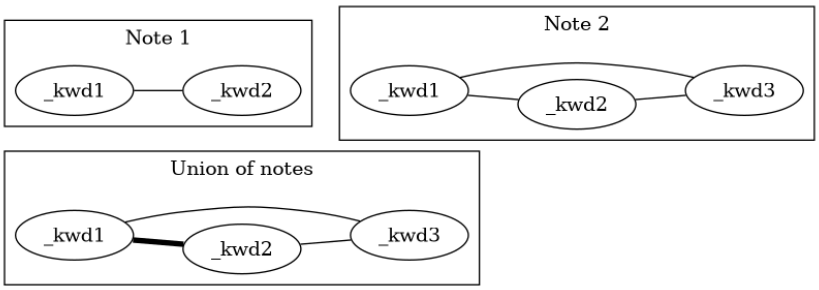

Figure 5.6 Denote network of keywords.

Denote-Explore searches the Denote files for links or keywords and converts this information to a GraphViz image. The diagrams themselves are saved as SVG files. Emacs activate use the software that your operating system uses to view SVG files. Ideally this should be the internet browser so you can leverage the full functionality of these files.

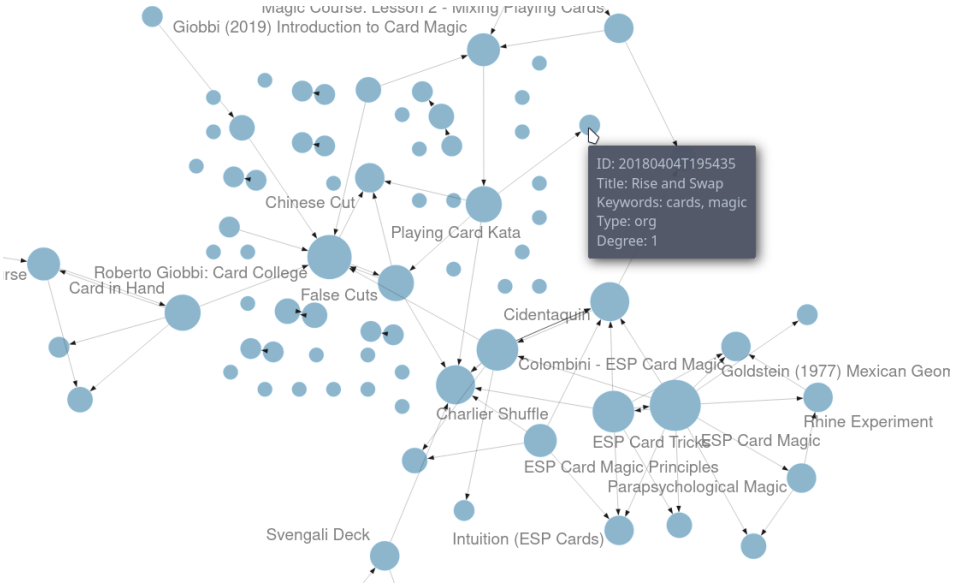

Figure 5.7 Screenshot of a Denote neighbourhood graph created with Denote-Explore and GraphViz.

The size of each node is proportional to the number of notes that are linked from or linked to the note. When the degree is more than two, the name of the node is displayed. When viewing the SVG file in a web browser, hovering the mouse over a node a popup window displays the note's metadata.

Clicking on a link will open the relevant file. You will need to configure your browser to open Org mode files with Emacs. Ideally you should configure Emacs as server so it does not open a new version for every link you click. You will find ample resources on the internet to show how to use Emacs as a server in your operating system.

You can regenerate the same network after you make changes to notes. The `denote-explore-network-regenerate` command recreates the current graph with the same parameters as the previous one, which is useful when you like to see the result of any changes without having to enter the search criteria again.

It might seem that adding more connections between your notes improves them, but this is not necessarily the case. The extreme case is a complete network where every file links to every other file. This situation lacks any interesting structure and wouldn't be very useful for analyses. So, be mindful of your approach to linking notes and attachments.

5.6 Implementing Note-Taking Systems

The Denote package is flexible and malleable, so you can use it to implement any published note-taking methodology. This section provides some suggestions on how to implement each of these methods. EWS does not promote any note-taking systems. This section only demonstrates how you could implement them. Try mot to get distracted by 'shiny-object syndrome' and focus on what is useful (writing), rather than chasing the latest ideas. The ideal method is one that you grow organically based on your needs. The power of the Denote file naming convention and regular expressions basically provide everything you need at your fingertips.

5.6.1 PARA

Tiago Forte has developed the PARA method to organise your digital life (Forte, 2022). In his system, all digital assets form part of one of four folders:

1. *Projects*
2. *Areas*
3. *Resources*
4. *Archives*

Forte uses a cooking example to explain para. The *Projects* are the pots and utensils you need to prepare a dish. Files in this category are the material you need to work on for your current deliverables. The *Areas* are like the ingredients you store in the fridge. These are notes that you need to access regularly. The third category is *Resources*, which relates to items stored in the freezer. These are topics that interest you or research material. Lastly, the *Archive*, which we can call the pantry, contains completed projects or those that are on hold.

The key to this method is that each file belongs to only one of these four categories. A file could start as a resource, become a project, and end its life in the archives. Forte used four directories to store material from each category in his original idea. You can implement this method in Denote by associating each note with one of four tags. Ideally, each note should belong to only one of these four categories. To list all notes in your *Projects* category, open `consult-notes` and search for `_projects`, and so on.

EWS includes a bespoke function to implement Forte's PARA method with Denote. The Denote-Explore package includes the `ews-denote-assign-para` function that moves a note to one of the four PARA categories by assigning a keyword to the note. If a PARA keyword already exists, it is replaced with the new version.

The `ews-para-keywords` variable contains the keywords used in this method. You can customise this variable to translate it into your native language or use a different set of exclusive categories. You can, for example, also configure this variable to implement the ACCESS system by changing the options to Atlas, Calendar, Cards, Extras, Sources, Spaces and Encounters. Any file management system that depends on folders can be replaced with Denote keywords in this way.

If you insist on using folders instead of keywords, then that is also possible in Denote. Cus-

tomise the `denote-prompts` variable to ask for a subdirectory so that you can select a subdirectory in your notes folder when creating an new note.

5.6.2 Johnny.Decimal

The *Johnny.Decimal* System uses a numbering scheme to organise files, created by Johnny Noble (`johnnydecimal.com`). The basic idea is to assign your digital life a maximum of ten broad areas. Feel free to start with fewer than ten categories. You can begin with, for example, just *work* and *personal*. These main categories are the shelves in your digital library.

Each shelf can accommodate up to ten boxes, allowing you to define not more than ten boxes for each shelf. This flexibility means you can tailor the system to your needs. For instance, in our example, we could have boxes for *finance*, *writing* and *travel* on the *personal* shelf. The third step involves assigning numbers to each of the categories. *Johnny.Decimal* starts with 10–19 because lower numbers relate to system maintenance. The 00 folder usually contains an index to help you navigate the numbering system.

In our example, *personal* is shelf 10–19, and the boxes are numbered from 11 to 19, for example *finance* (11), *writing* (12), and *travel* (13). There is room for seven more boxes, don't use that capacity until the need arises. In the original system, the numbers form the start directory names. The Johnny.Decimal website provides a detailed description of this method. The *Johnny.Decimal* system works pretty much in the same way as the Dewey Decimal system in a library, but then with fewer categories. The *Johnny.Decimal* system describes your life, while the Dewey Decimal system describes humanity's knowledge. You could of course also categorise your notes using Dewey Decimal approach, the choice is yours.

You can implement Johnny or Dewey Decimal, or any other system that uses ordered numbering by adding a Denote signature. Denote does not use signatures by default. Customise the `denote-prompts` variable and tick 'signature' to include these in new notes.

For example, a note about EWS could have `12=03` as a signature, indicating it belongs to the *writing* box on the *personal* shelf. You could use a third level in your box to number individual files, so a files in the *writing* box could be numbers as `12=03=01`. `12=03=02` and so on. You can use meta notes (section 5.3.4) to list all the files within this box. By changing the `sort-by-component` to `signature` to order the links in the list. Without this sorting instruction, notes are ordered by ID.

- 12=03=01 ews purpose
- 12=03=02 zettelkasten
  - ... etc.

5.6.3 Zettelkasten

Many people get attracted to note-taking when they hear about Niklas Luhmann's Zettelkasten concept. A Zettelkasten is a German word for a box (Kasten) that contains notes (Zettels). Luhmann was an influential sociologist famous for his enormous productivity and expansive note collection of over ninety thousand interconnected index cards. His Zettelkasten helped with his extensive research output. He was also a workaholic, so using a system does not make you magically more productive.

The Zettelkasten method, essentially a paper version of a personal wiki, is not a concept unique to Luhmann. In fact, it was a recommended research method during my university days. I remember spending countless evenings at the dining table, rearranging index cards to structure my essays. What sets Luhmann apart is his unwavering discipline in note-taking, a trait that continues to inspire many. This adaptability is one of the key strengths of the Zettelkasten method.

Luhmann's method for his Zettelkasten included a signature that links cards sequentially in a

branching hierarchy. The main difference between *Johnny.Decimal* and Zettelkasten is that in the former system, only folders have numbers; in Zettelkasten, each note is numbered. Luhmann's original Zettelkasten has at least six levels of nested categories. This is a tiny extract from his original work, sourced from `niklas-luhmann-archiv.de`:

- 76: Causality
  - 76,2: Causality — motivation
  - 76,5: Causailty as regular order
    * 76,5a: Causality: Equivalence of cause and effect

The notes are ordered to form a coherent idea, which is the main reason Luhmann was so efficient in writing. His articles and books developed as he took notes.

You can implement the Zettelkasten method with Denote. The individual files are the 'Zettels', and your Denote directory is the 'Kasten'. Just like before, the signature can contain the reference number. You cannot precisely copy Luhmann's syntax because he uses characters in his signatures that you cannot use in filenames. In example list above, the last note would have `76=5=a` as a signature.

5.7 Learn More

This chapter only provides a brief introduction to the Denote package and its associated auxiliary packages. These packages have further extensive functionality to make the software behave the way it best suits your preferences.

The extensive Denote manual describes its full functionality in great detail, with lots of options to configure how it works. The Citar-Denote and Denote-Explore packages also provide manuals through the info system. You can access these three manuals with `C-h R denote` and select the relevant package.

Now that you have collected a lot of notes, it is time to start a writing project. The next chapter shows how to work on a large writing project using Org mode.

6

Production: Write Articles, Books and Websites

6.1 Introduction

Chapter 2 explained the basics of how to write a plain text document. In chapter 3 we dug deeper into writing by introducing Org mode. This chapter takes you further along the journey by showing how to write and manage a large project, such as a book, and get it ready for typesetting.

Now that you have your ideas on paper in a structure of your liking, it is time to put 'pen to paper' as they said in the olden days, and start working on your project.

The rest of this chapter explains in detail how to manage a writing project with Org mode. This second part of this chapter briefly introduces two other markup languages: Markdown, and Fountain Mode.

6.2 Edit your work

A typical writing project contains a lot more than just letters, numbers, and punctuation. We have already discussed some of the metadata placed in the front matter of an Org mode file. Org mode also has a syntax to include images, tables and drawers.

This section adds some more tools to your belt to prepare your manuscript for publication. Org mode has provisions to add notes to a document, which are ignored in the final product. This section also explains how to add citations from your bibliography.

6.2.1 Taking Notes

The previous chapter detailed the versatility of note-taking in Emacs, whether it's within a single Org file, a network of interconnected files using Denote. You can also keep notes inside your writing projects, outside your formal note-taking system. These notes serve as valuable reminders of pending tasks or anything else you like to keep hidden from the final product. For instance, you can include a checklist, links to websites or notes in your Denote digital garden, or any other crucial information during the writing process. However, if you need to jot down any notes unrelated to the current document, the Org capture system, which stores fleeting notes in your central inbox, is a more appropriate choice (Chapter 5).

Org mode offers two flexible methods for note-keeping. The first method is straightforward. Any line in Org mode that starts with a hashtag or pound symbol (#) is a comment, which Emacs disregards when exporting the text to the final output. These comments can be used to write notes in areas that require attention or review, giving you a sense of control over your notes.

```
# This is a comment.
```

The second method involves using a drawer under the current heading. A drawer can hold multiple lines of text, making it ideal for storing text snippets for future reference or maintaining

a checklist. The beauty of Org mode drawers is that they can close, ensuring that their contents doesn't distract you from the primary task. You can open or close a drawer with the TAB key when the cursor is on the drawer. The example below shows what a drawer in your text can look like.

```
:DRAWER_NAME:
- This is inside the drawer.
- You can fill this with notes.
:END:
```

You can interactively insert a drawer at the cursor's current location by calling `org-insert-drawer`, bound to `C-c C-x d`. You enter the drawer's name in the minibuffer (by convention in uppercase, for example, NOTES) and fill in the drawer's content. If you select a block of text and then create a drawer, that text will appear inside it. You can use this method to mark parts of text you don't want in the final product but are worth keeping.

An EWS function (`ews-org-insert-notes-drawer`) generates a note drawer bound to the `C-c w n` keyboard shortcut. This function moves the cursor below the heading of the section you are writing and generates the drawer names NOTES. If a notes drawer exists for this section, the function creates a new line at the end of the existing notes. After writing the notes, `C-u C-SPACE` will take you back to your original position in the text.

The next chapter's section 8.1.3 explains how to create a checklist to track the progress of your to-do list.

6.2.2 Adding Citations

Citations are the essence of scholarly writing. They are the currency of an academic career, marking the influence and impact of your work. Org mode can be your ally in this journey, offering a citation management tool that can read BibTeX, BibLaTex, or CSL files. To start, you'll need to create a bibliography. This can be done manually, or you can link a file from a bibliography management tool like Zotero (chapter 4).

When it comes to inserting citations, the org-cite-insert (`C-c C-x @`) command is your go-to. In the/Emacs Writing Studio/, this command opens the Citar menu, allowing you to select one or more publications. To select multiple references, simply use the TAB key after each selection. A citation will be inserted, which looks something like this: `[cite:@einstein_1905;@newton_1728]`.

The Citar package provides some convenient functions for managing citations. To change the order of citations in a block, use the shift and left/right arrow keys (`citar-org-shift-reference-left / right`).

For accessing the resources related to a citation, you can use the org-open-at-point (`C-c C-o`) command. This opens the Citar menu for the citation under the cursor. If the citation is not found in your local or global bibliography, Org mode prompts you to create a new heading, which is irrelevant in this case.

A bibliography can be global or local. The global bibliography is accessible from any place in Emacs. In contrast, the local bibliography file is only available within your Org mode file.

The global bibliography is set in your configuration and can consist of one or more files. In EWS, the files for the global bibliography are stored in `ews-bibtex-directory`, which you can customise with the usual method. If you change this directory or add a new bibliography file, the evaluate the `ews-bibtex-register` function (`C-c w b r`) to make it available for citations.

The local bibliography is linked to an Org mode file with `#+bibliography: "bibfile.bib"` in your document header. Note that the local bibliography does not extend to linked files, so you must repeat this line in each Org mode file that is part of your project and accesses this data.

Citar indicates whether an entry is cited in your current file with a C in the list of publications. To view only cited entries, use the `:c` keys in Citar.

The next chapter (section 7.1.5) delves deeper into Org mode's citation management system. It explains how to format them to achieve the desired output.

6.2.3 Cross References

To reference to figures and tables you can give the item a name with the `#+name:` indicator below the caption, for example: `#+name: fig:example`. When you refer to this name as a link (`[[#fig:example]]`), Org mode will link to the location of the image or table. When exporting the file, these links become links in to relevant output format.

References to sections or chapters are simply links to the name of the heading. So a link to this particular section would be `[[Citation Management]]`. This approach risks producing broken links if you change the heading name but forget to modify the link. You can add a property to a heading with a custom ID. Use `org-set-property` (`C-c C-x p`) and select `CUSTOM_ID` and enter your fixed ID.

It is good practice to name your ID with a prefix that indicates the type, for example use `fig:` for figures, `chap:` for chapters and so on.

6.2.4 Text Completion

Autocompletion is a common feature in mobile phones that apparently makes life easier for authors. While it might be a great feature for writing on a small keyboard, whether it is a useful

Automatic completion of words is called 'completion at point' in Emacs, with point being the location of the cursor. In EWS, completion at point is not configured for text mode.

1. Abbrev Mode A lot of formal writing from governments and businesses is littered with abbreviations and acronyms. Abbreviations have been popular since the start of writing. Roman inscriptions are hard to read even if you do understand Latin. Roman writers has to use abbreviations because it saved them a lot of time chiselling the full text. However, in the age of electronic writing, we can use full words. Electronic writing systems can automatically expand abbreviations into their full context.

   An Emacs abbrev is a sequence of characters that expand into something else. For example, a fairytale writer might define `ouat` to expand into "Once upon a time".

   To define an abbrev, select the text you like it to expand to and type `C-x a g` (`add-global-abbrev`).

2. Completion at Point

6.2.5 A Clean Writing Interface

Writing takes total concentration to produce creative prose. Distractions are the natural enemy of concentration. While your computer is your most important writing tool, it can also be a source of distractions. Most writing software is littered with icons and options to change the document's design. Distraction-free writing tools remove these distractions from the screen, so they become more like old school typewriters that let the author focus on content over form.

Olivetti is an Emacs minor mode that facilities distraction-free writing. The name Olivetti derives from the famous Italian typewriter brand. You activate Olivetti mode with `M-x olivetti-mode`. This minor mode reduces the width of the text to seventy characters and centres the text in the middle of the window. The width of the text is changeable with the `M-x olivetti-set-with` command or `C-c \`.

EWS includes a function that makes Olivetti mode a bit easier to use. This code stores your window configuration when you hit `C-c w o` and activates Olivetti mode (`ews-distraction-free`).

This function also increases the text by one step to create a nice focussed screen. Activating the function again restores the previous window settings.

6.3 Manage the Writing Project

A writing project is about more than just smashing lots of words into a document. Some functionality is available in Org mode to manage your project by adding notes to your files, manage word counts, cross references and the overall progress of your writing. Org mode can also split large projects into multiple linked files.

6.3.1 Large Projects

Writing a book in a single Org mode file can be laborious because you need to navigate a large file. The built-in narrowing tool can help in keeping your focus. Narrowing in Emacs means that the buffer will only show a selected part of your text so you don't get distracted by the rest of the document. The hidden text is still available, just not visible on the screen. To narrow your buffer to only show the subtree (heading and associated subheadings) you are currently working in use `org-narrow-to-subtree` (`C-x n s`). This commands reduces the visible text to the section under consideration. To go back to the full document evaluate the `widen` command (`C-x n w`).

Working with large files can in some cases can slow-down Emacs, so sometimes it might be a good idea to split larger projects over multiple files. Org mode has an inclusion function that creates a link between documents. For example, the `#+include: "chapter-02.org"` line includes a file named `chapter-02.org` inside the main document. You can visit this child document with `C-c '` (`org-edit-special`). Org mode has some additional options to determine exactly which part of the child document is included. You can, for example, exclude the sub-file's title line by adding `:lines "2-"` to the include keyword. This parameter instructs Org mode to only include the text from line two onwards. This method allows you to work on a book or dissertation and store each chapter in a separate file, as is the case with this book. When you export the main file to the final publication, all included files are added to the export.

6.3.2 Counting Words

Counting words is a standard activity for any author. 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 active buffer, use `M-=` (`count-words-region`). This function displays the number of lines, sentences, words, and characters in the echo area. Adding the universal argument counts the whole buffer (`C-u M-=`). The `count-words` function, which has no default keyboard shortcut, counts all words in the buffer or the marked region. A line in this context is a logical line, which is the same as paragraph when using Visual Line mode.

Counting words is not an exact science because it depends on the definition of what is a character, word or sentence. When counting characters, Emacs also counts spaces and semantic constructions, such as the metadata of an Org file. The definition of a word is not standardised. Being primarily a code editor, Emacs counts hyphenated words or any two words separated by a punctuation mark as two. By default Emacs defines a sentence as a sequence of characters that end with a full stop and double spaces. This default setting generates wrong results when counting sentences as most authors use single spaces, so EWS disables this behaviour. Adding double spaces at the end of a sentence made sense in the days of typewriters. Most style manuals, such as the *The Chicago Manual of Style*, recommend using single spacing (University of Chicago press, 2017, , 2.9). When

exporting text to the final product, the typesetting software inserts appropriate spacing after sentences. The only disadvantage of this method is that abbreviations such as "E. W. S." count as multiple words and sentences.

To find out the number of words in each chapter or section of your text you would have to run `count-words-region` for each part of your document. EWS provides a function to automate this task and provides an almost instant word count for each part of the buffer. The `ews-org-count-words` (`C-c w c`) function cycles through all headings and adds the word count in a property drawer, which is another kind of drawer that works much in the same way as the notes drawer described above. The word count for higher level headings include the content for their lower headings. This method also lets you add word count targets for each section so you can monitor progress. Use `C-c C-x p` (`org-set-property`), type "TARGET" and enter your desired word count. You can of course also manually edit the drawer.

```
* Heading
  :PROPERTIES:
  :WORDCOUNT: 305
  :TARGET: 300
  :END:
```

Property drawers are a powerful feature that can convert an Org mode buffer into a simple database. The collapsible property drawer displays the word count and your manually added target. You can also see an overview of these properties in table format. First we need to define the desired properties to display by adding the following line to the front matter of the Org buffer:

```
#+columns: %40ITEM(Section) %10WORDCOUNT(Word count) %10TARGET(target)
```

The percentage sign indicates the number of characters for this column in the table and the text after the number matches the property name, here `ITEM` stands for the header text. The text between parenthesis is the display name for the column. You can now view the word count and target for each heading in a table with `C-c C-x C-c` (`org-columns`). Ensure you evaluate this function when the cursor is at the highest level in the hierarchy (beginning of the document). This view creates an overlay, with the top line of the buffer as table heading.

The headlines become read-only and contain the properties defined as columns. You have a few options when the cursos is on one of the headlines. The `c` button collapses the headings so you see only the table andnot the underlying text. You can still edit the text, but visual line mode is disabled.

Navigate through the table with the arrow keys and You can edit a property with the `e` key. Change the content in the minibuffer and hit Enter. The `g` key resets the columns after you, for example, change the definitions in the meta data.

All headings have a grey background and contain the values of the defined properties. A table appears at the overview and contents level of the document by cycling through the document with `S-TAB`. When the cursor is in the table yo have a few options. Use `e` to edit the property so you can update the targets for each heading that needs one. Place the cursor on a column overlay to remove the overlay and press `q`.

6.3.3 Tracking the Status of your Writing

The typical workflow of writing goes through various stages from early drafts, to edited versions and completed texts. As you are working on various parts of your writing project it might be good to know the status of each chapter. Org mode includes an extensive system to manage projects which you can deploy to keep track of progress in your document. This section is only a very brief introduction to this functionality. Chapter 8 explains project management in more detail.

Each heading in Org mode can have a status token, such as `TODO`, `DRAFT` or `EDITED`, or whatever workflow you prefer. You add a status token with the shift and left/right arrow keys when the

cursor is on a heading. You can also use the `C-c C-t` shortcut (`org-todo`). By default, the system only recognises the `TODO` and `DONE` status. You can add additional workflow states by defining them in the document header. The example below instructs Org mode to cycle through these four status tokens, but only in this file. The tokens before the vertical line (pipe symbol) are in progress and usually marked in red. Items after the vertical line are completed and marked in green.

```
#+TODO: TODO DRAFT EDIT | FINAL
```

If you like to add the status of your heading to the summary table discussed in the previous section then add `%20TODO(Status)` or something similar to the columns definition in the front matter.

6.3.4 Quality Assurance

1. Dictionary and Thesaurus While spellchecking is great to ensure a Emacs has a built-in dictionary search function that connects to an online source. The default for *Emacs Writing Studio* is the Collaborative International Dictionary of English (CIDE), derived from the 1913 Webster's Dictionary, with some definitions from WordNet. It is proof-read and supplemented by volunteers from around the world. This dictionary is available through the `dict.org` website.

   To lookup the word that the cursor is currently on, use `dictionary-lookup-definition` (`C-c w s d`). A dictionary screen pops up that provides the relevant definitions. You can scroll through the window as with any other buffer. The dictionary buffer contains links to other defined words, which you follow with the Enter key. Using the `n` / `p` keys jump between hyperlinks. To lookup a new word type `m` or click on the `[Search Definition]` button on top of the window.

2. Checking Grammar The core skill in writing is choosing the correct words. Equally important is knowing which words not to use. WriteGood mode by Benjamin Beckwith. This minor checks your text for three fundamental problems: weasel words, passive voice and duplicates.

   Writegood mode highlights the issues with your text with coloured squiggly lines below the text. Hovering the mouse over a marked word provides context on the transgression.

   Weasel words are often used by demagogues, politicians and marketers to disguise what they are saying. A tax becomes a levy, we no longer live, we have a lifestyle and sacking people becomes downsizing. They are weasel words because they suck the meaning out of language, just like a weasel sucks eggs (Watson, 2004). You can find the list of weasel words that this package defines with `C-h v writegood-weasel`.

   Passive voice

   Our minds are not particularly good at detecting duplicate words.

   Duplicate words are often an artefact of copying and pasting text or

3. Readability Test The WriteGood package can also perform the Flesch reading ease score to asses how easy or difficult an English text is to understand. The score ranges from `0` to approximately 120. Higher scores indicate that the text is easier to read. You can perform this test with the `writegood-reasing-ease` function (`C-c w s g`). For the mathematically inclined, this formula calculates the readability index:

$$206.835 - 1.015\left(\frac{\text{words}}{\text{sentences}}\right) - 84.6\left(\frac{\text{syllables}}{\text{words}}\right)$$

   Basically this test confirms what we intuitively know. Texts with long sentences (average

sentence length) and long words (syllables per word) as less easy to read. For reference, the readability index or *Reader's Digest* is about 65, *Time Magazine* scores about 52, and the *Harvard Law Review* has a general readability score in the low 30s (Lipovetsky, 2023). The Flesch-Kincaid reading ease score for this chapter is 73, which is "Fairly easy" and aligns with a 7th grade reading level.

These type of tests are not an exact science. As discussed in section 6.3.2, counting words and sentences depends on some assumptions.

6.4 Control Versions and Collaborate

In the throws of the writing process it is not uncommon to change you mind a few times on how a text should flow or even totally change it's structure. To ensure that you don't loose any valuable information, you need to understand how Emacs manages different the versions of a buffer or a file. Version control is also important when collaborating with other people. While Emacs does not have the fancy cloud collaboration systems common in office software, the built-in version control system enables working with multiple people on a project without loosing any contributions.

There always at least two versions of the text you are working on. The last saved version is stored on disk and the second version is the buffer that is being edited. You can discard all the changes since the buffer was last saved with `revert-buffer`, which reloads the file from the disk, erasing all edits since the last saving of the file. This is a nuclear option to be used with care. Reverting a buffer is only useful when you made huge mistakes or saved an updated version prepared outside of your current Emacs session.

Emacs also provides more subtle ways to control your versions. Firstly, while you are editing, the undo system keeps perfect track of all changes. Section 2.7.4 discusses the basic undo methodology but we can add some more sophistication to this workflow to keep track of various versions created while writing. The second method uses the built-in backup system to save older versions of files. This system creates a copy of your file before starting a writing session, keeping a backup of your previous version. There are also more advanced version control methods that let you check in and out files to formally register a new version. This method are ideal when collaborating as Emacs has fine-grained functionality to manage difference between contributions.

6.4.1 The Undo Tree

Section 2.7.4 discussed how to correct mistakes using Emacs' powerful undo system. However, after repeatedly issuing undo and redo commands it is easy to get lost the previous states of the document. The Undo Tree package by Toby Cubitt helps you keep track of your changes by visualising them as a tree.

The `undo-tree-visualise` (`C-x u`) command visualises the various edits in your file as a tree. This function lets you walk through previous versions of your text with the arrow keys. The current buffer changes as you wander through its history with the arrow keys, where `x` marks the spot of the selected step. Use `q` to select the chosen edit and continue writing. The `C-q` keys jump out of the undo tree without making changes.

In most writing, the tree is simply a straight line, but when combining undo and redo commands, the document forms parallel versions visualised as branches. The undo tree for the simple example in figure where we started with Socrates, changed to Plato and back again and added some text would look like this:

```
o       "Socrates"
```

```
          |
          |
          o       ""
          |
         / \
"Plato" x   o     "Socrates"
            |
            |
            o     "Socrates and"
```

This package provides an intuitive way to manage the various states your document went through since you opened the file. You can read the detailed manual for the Undo Tree package which provides more detailed scenarios with `describe-package` (`C-h P`) and select `undo-tree`.

6.4.2 Automated Backup

Rewriting a file automatically destroys all record of its previous contents, which sometimes means loose many hours of writing within a split second. To prevent such disasters, Emacs keeps a backup of every file.

Emacs backups a file the first time the file is saved. No matter how many times you subsequently save the file, its backup remains unchanged. However, if you kill the buffer and then visit the file again, a new backup file is made. So the backup files contain the versions just before starting a new writing session. This backup will be the same as the current file, until the next save.

By default Emacs stores backup files in the same directory as the original file, which can lead to a lot of clutter. In EWS, backups are stored in the Emacs configuration directory under `backups`. Emacs appends the original file name with a tilde to indicate that it is a backup, so the backup for `origin-of-species.org` would be `origin-of-species.org~`.

EWS is also configured to keep the last three versions of the file.

6.4.3 File Versions

As your writing project progresses you might end-up with different versions of the same file, either through your own doing, an editor or other collaborator. This situation might raise a problem as you are now unsure which file is the most recent version, or perhaps you need to create a new version that contains all the latest changes. The `ediff` command helps you solves this problem. It provides a rich interface to compare two or three files. Ediff visualises differences between files and lets you pick which parts of each file you like to keep.

When issuing this command, you need to select two files using the minibuffer, referred to as file A and B. The Emacs frame splits in three parts, the two files and the control panel at the bottom of the frame. The control panel lets you issue commands to either of the two open buffers.

When you type `n`, Ediff takes you to the *next* difference. The paragraph where the difference occurs is highlighted, with the actual differences in a more intense background. Repeatedly typing `n` takes you through successive differences and `p` to the *previous* one. The mode line of the control panel displays the number of differences and your progress through them.

Ediff also lets you act on these difference by synchronising parts of file A with B or vice versa. When you type `a` in the control panel, file B changes the highlighted line(s) to the version in file A, and the other way around when you type `b`.

You can also move the cursor into either of the two file buffer to edit them manually as you would normally. However, this can confuse matters as you are no longer certain what you typed and the content of the file. Any text added during the Ediff session is not recognised as a new difference.

To end the session type `q` in the control panel and follow the prompts. You can kill any unmodified buffers. The changed buffer can be saved to disk.

Ediff has a lot of functionality outside the scope of this book. Type the question mark in the control panel for a list of options. Ediff has other available commands. To compare three files (A, B and C), use `ediff3`. The `ediff-backup` command compares a file with its latest backup.

You can read the Efiff manual for a comprehensive description with `ediff-documentation` or `C-h R ediff`.

6.4.4 Version Control

The most advanced method for

If you work with an editor to review you text, you could send them an Org mode file, which they can

6.4.5 Working in Cloud Storage

being an authors can be a lonely activity, but

To collaborate with other people on a project you could store your project files on a file-sharing service such NextCloud. However, one limitation of Emacs is that it is not advisable for more than one person to open a file simultaneously. If that would be the case, then two ore more people

The EWS configuration disables lock files, so if you need this functionality you will need to change the configuration, as explained in the Appendix.

6.5 Learning More

The next chapter discusses two further text systems, HTML and LaTeX.

Emacs Writing Studio provides a convenience function to insert images from screenshots, which is bound to key. This function asks for a filename (and uses a PNG extension). The user then selects a partial screenshot and a caption. The function then inserts the image link and caption.

This book revolves around using Org mode for your writing projects, but that is only one of the text modes available in Emacs. While Org mode is by far the most feature-rich, there is sometimes a need for other modes that use `text-mode` as their foundation.

The most basic version of a text file are plain text files that usually have a `txt` extension for their file name. These files are plain in the sense that they don't contain any formatting and generally consist generally only of alphanumeric characters, spacing and punctuation. If we want to publish a work as a website, a book or any other type of media, a plain text file will not suffice because there is no way to define what the final result should look like, such as the page layout, font types, hyperlinks and other such important parts of a published work.

Other text modes consist of styled text or rich text. These files contain plain text plus additional information about the design of the document, such as font style, links and so on. Org mode and HTML are examples of styled plain text. The instructions on styling are the markup of the document. In traditional publishing markup is a system of annotations in red or blue pencil that instruct the printer how to style the text. Marking-up a document was a laborious process in which editors and typesetters used symbols (the markup) to indicate how the text should appear on the page. In the world of digital publishing we use sequences of characters and punctuation as markup to instruct the computer how to display a document.

Graphical editors hide the markup from the writer and shows the text in its final form. This method might seem convenient, but it can also become a nightmare as you try to wrangle the system

to get the result your want using these invisible instructions. Many plain text modes exist for all sorts of purposes. Some honourable mentions of plain text formats are Beta Code to write ancient Greek with European characters and Lillypond to write sheet music.

There are two types of markup. Presentational markup adds instructions on how to present the text, such as bold face, italics, lists and headings. Procedural markup consists of symbols to instruct the computer about aspects such as page size, text position, citations, meta data and other more complex aspects of a publication (Travis & Waldt, 1995).

Styled text modes come in two types, regular markup and lightweight versions. A regular markup language, such as HTML or LaTeX (pronounced *lah-teck*), includes instructions that look like a computer language to define the design of the document output. For example, to wite a heading in HTML and LaTeX you need:

- HTML: `<h2>This is a heading</h2>`
- LaTeX: `/section{This is a heading}`

Regular markup languages provide powerful capabilities to define all details of the final output of your project. Disadvantage is that your text is littered with angled brackets or curly braces and instructions. In lightweight versions the number of characters needed to define a document is vastly reduced, simplifying the process or writing. Org mode is an example of a lightweight markup language. It is not lightweight due to limited capabilities but because of the reduced instruction set. To create the same heading in Org mode, all you need is to add an asterisk at the front of the line, removing some clutter from the screen.

6.5.1 Introducing Markdown

HTML and LaTeX are widely used markup languages, but the screen is littered with angled brackets or curly braces. Internet pioneers John Gruber and Aaron Swartz created Markdown in 2004 as a markup language that is easy to read and minimising the amount of semantic characters. Markdown is widely used for instant messaging and in online forums. It is also commonly used to document software. The basic principles of Markdown are similar to Org mode, as shown below.

```
# Heading

## Sub-Heading

Text attributes: _italic_, **bold**, `monospace`.

Bullet lists nested within numbered list (indented with four spaces):

1. Fruits
    * Apple
    * Banana
2. Vegetables
    - Carrot
    - Broccoli

A [link](http://example.com).

![Image](Icon-pictures.png "icon")
```

Unfortunately, various flavours of markdown exist, most of which provide additional functionality. The Markdown Mode package implements the original version. The *Emacs Writing Studio* configuration activates Markdown by default, but a complete description of this format is outside the scope of this book. Jason Blevins authored the Markdown Mode Emacs package and has published an extensive manual (Blevins, 2017).

The Denote package can create notes in Markdown in two varieties. Unlike Org mode, Markdown has no provisions for storing meta data about the document. Denote provides two methods to achieve this by either using TOML (Tom's Obvious Minimal Language) or YAML (YAML Ain't Markup Language). You can set the `denote-file-type` variable to either `markdown-toml` or `markdown-yaml` to start creating Markdown notes instead of the default Org mode. The syntax of either front matter type is intuitive. Read the Denote manual for more details and try the different varieties for yourself. By the way, Denote also has the option to create notes in plain text. To enable this option set the `denote-file-type` variable to `text`.

6.5.2 Screenwriting with Fountain

Who wouldn't want to write a screenplay for the next Hollywood or Bollywood blockbuster? Writing movie or theatre scripts follows some strict principles and formatting rules. The standard font for screenplays has a fixed pitch, giving the document an old-school typewriter feel. Fountain is a plain text format to write screenplays in any text processor. The Fountain file format is quite special as it contains almost no markup. Given the strict conventions in screenplays, Fountain can logically determine how to format the document. The example in Figure shows an excerpt of the screenplay of the 2003 fantasy drama *Big Fish* directed by Tim Burton, based on the 1998 novel *Big Fish: A Novel of Mythic Proportions* by Daniel Wallace.

```
INT.  TINY PARIS RESTAURANT (LA RUE 14°) - NIGHT (1998)

WILL, now 28, sits with his gorgeous bride JOSEPHINE.  This
is their wedding reception, crowded with their friends and
family.  They should be joyful, but Will is furious.

Edward has the floor, ostensibly for a toast.  The room is
cozy and drunk.

                    EDWARD
          This fish, the Beast.  The whole time
          we were calling it a him, when in fact
          it was a her.  It was fat with eggs,
          and was going to lay them any day.
```

Figure 6.1 Example of a movie script written in Fountain mode.

Fountain Mode implements this text format in Emacs, but it Fountain is not enabled by default in *Emacs Writing Studio*. If you like to have a go becoming the next Shakespeare or Stanley Kubrick, then you need to install it by adding the following line to your init file.

```
(use-package fountain-mode)
```

There should not be any need to configure variables. The package comes with an extensive manual that you can access with `C-h R fountain`.

7

Publication: Share with the World

Now that you have written your magnum opus in Org mode it is time to share it with the world. You can of course share your Org source files, but is not.

Emacs Org mode is an ideal tool for writing without distractions because working with plain text allows you to focus on content instead of design. While writing articles and books in plain text is pleasant and productive, it is not ideal for sharing your work. Org mode can export your work to PDF through LaTeX, to a website or a word processor document. There are also additional packages that can export to additional formats such as an ePub book. This article explains how to export your work as a journal article, a print-ready book PDF, or an ebook in ePub format.

The first section in this chapter explains the basic principles of exporting Org mode content and how to configure the output. The remaining sections explain the specific settings to export to Office documents, PDF files for physical books, ePub files for ebooks, and presentations in PDF format.

7.1 Org Mode Export Principles

The basic principle of exporting Org mode files to the desired format is that the system converts the text to the relevant format and then connects it to a document class, CSS style sheet or other type of template. Your text file can also link to a local or global bibliography to manage citations. Org mode has built-in export facilities for LaTeX (PDF), HTML and ODT (LibreOffice) and some other formats. Using the Ox-Epub package you can also export to other formats, such as ePub (Figure 7.1).

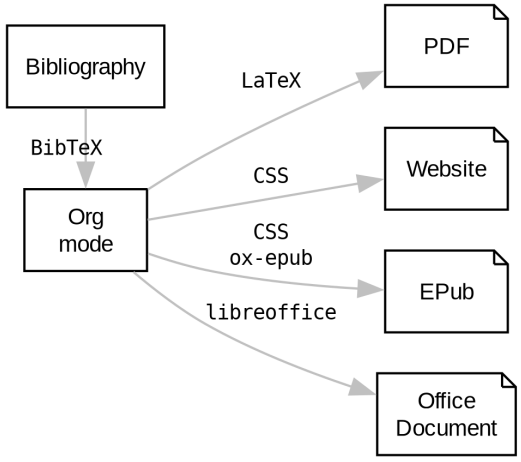

Figure 7.1 Org mode export principles.

In principle you don't need to know HTML or LaTeX, but it will certainly help with fine tuning

the output. The Org mode export function understands how to convert between formats. For example, if you create a website, any Org mode text surrounded by forward slashes (`/example/`) is translated as `<i>example</i>/` while in LaTeX it becomes `\emph{example}`. The HTML style sheet or LaTeX document class determines the way this code looks in the final output (the "what you get" in WYSIWYG). This example becomes italic text by default, but it could be something different if the stylesheet or document class defined it as such.

7.1.1 The Org mode Export Dispatcher

Start the export module with the `org-export-dispatch` function, which you can run with the default `C-c C-e` keyboard shortcut. The dispatch provides a range of options. The first letter you type determines the export format, and subsequent letters the specific options. Using `q` exists the export dispatcher.

7.1.2 Document Settings

Before delving into the specific export formats, some housekeeping sets the defaults on how a document is exported. There are various settings to control how Org mode exports your document, which you add to the front matter of your file, or set them as global variable for all exports. Some generic export variables are:

- `#+title:` Document title.
- `#+author:` Author name (defaults to `user-full-name`).
- `#+date:` Date of publication (You can set the format of the export with the `org-export-date-timestamp-format` variable).

You can obtain a template for exporting to ePub through the `org-export-insert-default-template`, which inserts keywords for your Org file's front matter. You can also access this function trough the export dispatcher with `C-c C-e #`.

By default, Org mode saves the result of the export process in the same folder as the source document. The exported file has the same name as the source file but with a different extension.

Exporting Org mode files can create a filename conflict because you are effectively creating two files with the same identifier, so ideally you should change the name of the exported file.

You can change this behaviour by adding `#+export_file_name: <filename>` to the header. The filename can be any string without file extension. You can save the exported file in another folder, but this might cause errors for certain formats.

When you add `num` to the startup keyword, Org mode will number every heading (`#+startup: num`). The numbering appears in the Org file. Whether this numbering also appears in any published output depends on your export template.

7.1.3 Section Settings

Headings in Org mode can be numbered by setting the startup keyword: `#+STARTUP: num`.

In some publications, not all headings are numbered, such as in the front or back matter of a book. You can exclude individual sections from the numbering system by setting the `:UNNUMBERED:` property as shown below. To add this property, type `C-c C-x p` (`org-set-property`) to set the unnumbered property set it as `t` (true).

This setting also excludes

```
:PROPERTIES:
```

```
:UNNUMBERED: t
:END:
```

Not everything you write should be exported, as suggested in the export defaults in the previous section. Org mode exudes any lines with a # symbol. The *Emacs Writing Studio* configuration does not export drawers, so your notes remain private. Furthermore, you can also exclude a section of your writing from export by adding the `:noexport:` tag to a heading with `C-c C-q` (`org-set-tags-command`). You enter the tag in the minibuffer and you can use completion to find select tags. The tag appears on the right side of the heading name between colon markers.

7.1.4 Tables

Org exports tables without vertical lines

(University of Chicago press 2017, section 3.53). Org exports tables without vertical lines, but occasionally vertical lines can be useful to structure a table into groups of columns. To specify which columns form a group, use a special row where the first field contains only a forward slash (/). The other fields either contain a lesser-than symbol (<) to indicate that this column starts a group, or a greater-than (>) symbol to indicate the end of a column. Using <> inside a column makes it a separate group so it is surrounded by vertical lines.

```
|---+-----+-----+-----+---------+------------|
| N | N^2 | N^3 | N^4 | sqrt(n) | sqrt[4](N) |
|---+-----+-----+-----+---------+------------|
| / |  <  |     |  >  |       < |          > |
| 1 |  1  |  1  |  1  |       1 |          1 |
| 2 |  4  |  8  | 16  |  1.4142 |     1.1892 |
| 3 |  9  | 27  | 81  |  1.7321 |     1.3161 |
|---+-----+-----+-----+---------+------------|
```

Table 7.1 Example of table with vertical lines.

N	N^2	N^3	N^4	sqrt(n)	sqrt[4](N)
1	1	1	1	1	1
2	4	8	16	1.4142	1.1892
3	9	27	81	1.7321	1.3161

7.1.5 Bibliographic References

Org mode has a built-in citation manager that can use BibTeX / BibLaTeX or CSL files to reference bibliographic items such as articles and books. Chapter 4 explains how to create a bibliography and Chapter 6 explains how to add citations in Org files.

When exporting to LaTeX, citations are managed by the external software. When exporting to any other format you need to specify how Org mode manages citations. The default settings in Org mode provide basic citation support in author-year format.

This website by Tecosaur provides an in-depth description of how to manage citations in Org mode, much better than I can explain it.

7.2 Office Documents

The export function in Org Mode can export to the ODT format, which is compatible with MS Word, out of the box. The ODT export back-end relies on the `zip` program to create the final output

If you have LibreOffice installed, you can also create a `docx` file to make it easier for MS Word users to share in the joy of reading your writing. When you set this option, the export process will result in both an `odt` and a `docx` file. This behaviour is not the default in *Emacs Writing Studio*, so you must add this yourself if you prefer exporting in the Microsoft format. Alternatively, you can also use this variable to export to PDF.

```
;; Export ODT to MS-Word
(setq-default org-odt-preferred-output-format "docx")
;; Export ODT to PDF
(setq-default org-odt-preferred-output-format "pdf")
```

It is possible to set this option specific to the file you are exporting by adding these three lines somewhere in your Org mode file:

```
# local variables:
# org-odt-preferred-output-format: "pdf"
# end:
```

When Org mode exports the file, these lines are evaluated but not exported as they are treated as comments.

You can use specific export settings in the front matter:

- `#+subtitle:` The document subtitle.
- `#+description:` and `#+keywords` are added to the exported file(s) metadata.
- `#+odt_styles_file:` Add the path to an LibreOffice style file (`ott` format)

Creating a LibreOffice style file is straightforward:

1. Create a LibreOffice file of your liking using the Styles menu (press `F11` in LibreOffice),
2. Modify each style to your liking.
3. Test the document.
4. Save as `ott` file.

A LibreOffice file is essentially a `zip` file with an embedded set of `xml` files. Org mode extracts the `styles.xml` file embedded in your template file. You cannot use this method for templates (pre-configured content).

You can fine-tune how Org exports to OpenOffice to a great extent. For a detailed discussion on `odt` Export, read the online Org mode manual.

7.3 Physical Books with LaTeX PDF

Writing technical documents can be a bit more challenging than normal prose. Technical writers often present mathematical formulas, tables and images. LaTeX offers a variety of features like automatic numbering of equations and references, making it ideal for technical documents. It's also

extensible, allowing for customisation through packages for specific needs. While Latex has a learning curve, it produces professional-looking documents and is widely used in academia and technical fields. The output of LaTeX documents is optimised for printed works, so it is also great for writing non-technical books.

LaTeX is a modified version of an older format called TEX, first released by computer genius Donald Knuth in 1978. Knuth developed this tool because he was unhappy with the way his publisher typeset the books he wrote. The original TEX language is quite complex, so Leslie Lamport developed the LaTeX variety, which basically is a collection of macros to simplify writing TEX (Lamport, 1994).

The "Hello World!" example shown in the previous section would like like this in LaTeX. It looks a bit less busy than the HTML version. In this example, the text specifies that this document will be formatted as an article, which is one of the many document classes. In LaTeX, instructions start with a backslash and a function name, with any parameters between curly braces.

```
\documentclass{article}
\title{LaTeX Example}
\begin{document}
\maketitle
Hello world!
\end{document}
```

The AUCTeX Emacs package assists with writing and formatting LaTeX files. This package is not part of *Emacs Writing Studio*. Org mode has perfect export capabilities for LaTeX so you can take advantage of the lightweight markup of Org mode. The next chapter discusses exporting Org mode to LaTeX and other formats in detail.

LaTeX is a powerful typesetting system (pronounced "LAY-tek" or "LAH-tek"), especially for writing scientific and technical documents. LaTeX can convert your text into a beautifully designed PDF file for publishing an article, ebook or physical book. Many publishers of technical literature have LaTeX templates to comply with their style guide.

You don't necessarily need any knowledge of LaTeX to export to PDF. Still, it will certainly help if you like to fine-tune the design of your document. The basic syntax of LaTeX is easy to explain. Let's assume you have a straightforward Org mode file that looks like this:

```
#+title:  Example document
#+author: Peter Prevos
#+latex_class: book

Minimum example for a Org mode document.
```

The Latex equivalent of this example is:

```
\documentclass{article}
  \title{Example document}
  \author{Peter Prevos}
  \begin{document}
  \maketitle
  Minimum example for a LaTeX document.
  \end{document}
```

The LaTeX software can convert this document to a wonderfully formatted article due to the document class, which defines the typography and layout.

Writing documents directly in LaTeX can be confusing because you need to know its markup language, and your text is littered with backslashes, curly braces, and other syntactical distractions.

Being productive as a writer requires focusing on the text's content instead of how it looks. Org mode is the perfect LaTeX editor because it frees you from distractions and integrates perfectly with LaTeX.

To enable exporting Org mode files to PDF, you need to have LaTeX installed on your system. How to install LaTeX depends on your operating system, and your favourite search engine will point you in the right direction.

Keying `C-c C-e` to open the export dispatch, then `l p` creates and opens a PDF file. Other options are available to export the buffer to LaTeX or to save a PDF file without opening it.

To make the magic work, Org Mode converts your file to a `tex` file, after which the LaTeX software converts it to PDF. The system works out of the box without any configuration. With some configuration you can produce PDF files that are perfect for producing printed books.

7.3.1 Text elements

Org mode converts headers to relevant LaTeX headers, and text becomes a paragraph. The relationship between the heading level in your Org file and LaTeX depends on the configuration, explained below.

7.3.2 Tables and images

Org mode converts images and tables to LaTeX floats. You can add specific attributes to these floats by using `#+attr_latex:`, as shown in the image example below:

```
#+caption: This is an example image caption.
#+attr_latex: :width 5cm :options angle=90 :placement h
[[directory/filename.png]]
```

Various parameters are available to determine how your table or image looks in the final output. The Org mode manual provides a detailed overview in section 3.10.

7.3.3 LaTeX snippets

You can write simple LaTeX commands directly into your org file. For example, `\newpage` will add a page break. You can also place equations using dollar signs, for instance `$e^{i\pi} + 1 = 0$` results in $e^{i\pi} + 1 = 0$.

The `org-latex-preview` function (`C-c C-x C-l`) shows a preview of any LaTeX equations within the open buffer. The chapter on introduces the `org-fragtog` package to automatically toggle between the plain text and the preview.

To create front and back matter, use the `\frontmatter` and `\backmatter` LaTeX commands in your Org file at the appropriate locations.

For more complex snippets, you need to use a structure template. Press `C-c C-, l` to insert a LaTeX source block. This LaTeX example creates an image using the picture environment.

```
#+begin_export latex
\setlength{\unitlength}{1cm}
\begin{picture}(10,10)(-5,5)
\linethickness{1pt}
\put(-2.5,0){\vector(1,0){5}}
\put(0,-2.5){\vector(0,1){5}}
\put(0,0){\circle{3}}
\end{picture}
#+end_export
```

The image will not appear in Org mode as it is generated by LaTeX and will only appear when exporting to LaTeX. Any LaTeX fragments, except for formulas, in Org mode files will only be visible when exporting to PDF. This means that if you like to export to multiple formats, you will need to replace these fragments with something that applies to all formats, for example an image.

7.3.4 LaTeX Packages and Classes

By default, Org mode uses the article class and a set of default packages to export documents. Org mode provides three mechanisms to use LaTeX packages in your export:

1. In the header, using `#+latex_header:`
2. Configuring the `org-latex-packages-alist` variable.
3. Configure the `org-latex-classes` variable.

The Org mode header can do a lot of the work. The example below specifies the book document class with A4 paper size. This example also specifies the Times fonts package. The last line tells Org mode to omit the table of contents from the export.

```
#+latex_class: book
#+latex_class_options: [a4paper]
#+latex_header: \usepackage{times}
#+options: toc:nil
```

The `org-latex-packages-alist` variable defines the default packages that are used for every LaTeX export.

You can define more complex header configurations by changing the `org-latex-classes` association list.

The example below adds the template for the American Psychological Association (APA) journals. This list's documentation provides all the details you need to configure packages and classes for your exports. The `with-eval-after-load` function ensures that this variable is only set once the LaTeX export function is loaded by Emacs. Note the double backslash instead of the single one in regular LaTeX syntax for compatibility with Emacs Lisp.

```
(with-eval-after-load 'ox-latex
  (add-to-list 'org-latex-classes
               '("apa6"
                 "\\documentclass[a4paper, jou, 11pt]{apa6}
                  \\usepackage[nodoi]{apacite}
                  \\usepackage{graphicx}
                  \\usepackage[T1]{fontenc}
                  \\usepackage{times}"
                 ("\\section{%s}" . "\\section*{%s}")
                 ("\\subsection{%s}" . "\\subsection*{%s}"))))
```

The last part of this association list specifies the mapping between Org mode and LaTeX headers. In this case, the highest level is associated with the section header, the second level with a subsection, and so on.

You can call this particular class by adding `#+latex_class: apa6` to your file header. This mechanism empowers you to define bespoke LaTeX classes to create a library of export options.

These three mechanisms allow fine-grained control over how your Org mode document is exported to LaTeX and PDF. The *Emacs Writing Studio* configuration only uses the default settings for these variables because the possible use cases are too numerous to define a sensible default other than what is already available.

7.3.5 PDF export configuration

The *Emacs Writing Studio* configuration for PDF export defines the export process to ensure that bibliographies work appropriately. This configuration also cleans all temporary files that LaTeX creates so that only the exported file remains.

7.4 Ebooks Export to ePub

Most ebook publishers use the ePub format for distribution. This file type is a ZIP file with your book stored as a website optimised for an e-reader. The `ox-epub` package adds this functionality to the Org export dispatcher. This package uses the built-in Org to HTML export to create the ebook, so you can use any of its features to fine-tune the output.

There are some export options that need to be set in your header:

- `#+title:` the document title.
- `#+uid:` a unique ID of the document, otherwise known as URI, could be a website or ISBN number.
- `#+date`: the date of the document.
- `#+subject`:
- `#+description`:
- `#+publisher`:
- `#+license`:
- `#+epubstyle`:
- `#+epubcover`:
- `#+author:` the document author or editor, the creator in the EPUB spec

The default settings adds a postamble to the bottom of the last page with a timestamp, author and a HTML validation service. Adding `#+options: html-postamble:nil` to the Org file header removes these from your ebook.

Any LaTeX fragments are ignored in the export, which means that you have to convert them

However, LaTeX math formulas are possible with the `tex:dvipng` option added to the options line in the front mater. This option converts any LaTeX formula to a PNG image, which is not ideal but readable.

The `ox-epub` package does not convert Org mode timestamps to a date format that complies with the ePub standard. You can correct this by removing the square brackets and the day and time from the timestamp.

The export process for ePub is not as forgiving with missing images. While you can export to HTML and PDF without any errors, your ePub will not render if any linked images are missing.

Lastly, only use open image formats such as `.png` as some ebook readers cannot display JPG files and other proprietary formats. The ebook will look alright on your computer but might not pass any checks by a publisher.

7.5 Websites

7.5.1 HTML

The HyperText Markup Language (HTML) is the engine that drives the World Wide Web. Internet pioneer Berners-Lee specified HTML in late 1990. The example below shows a simple HTML file. In HTML, the markup is designated by angle braces (less than and greater than symbols). The indentation is not required, but helps with understanding the structure of the document, also called the DOM (Document Object Model).

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Example</title>
  </head>
  <body>
    <div>
        <p>Hello world!</p>
    </div>
  </body>
</html>
```

Emacs has a built-in mode for editing HTML files. However, Org mode has perfect export capabilities for HTML so you can take advantage of the lightweight markup of Org mode. Chapter 7 discusses exporting Org mode to HTML in detail.

7.6 Presentations

Presentations are the
The slide deck
Death by PowerPoint

Using Emacs to write presentations is not ideal, but it can be done. Being a text processor, any set of slides exported from Org mode will most consist of lists of text. Unfortunately, most presenters use their slides as speaker notes and rely on the dot points to keep their talk on track.

Various methodologies exist that can convert your Org mode files to a presentation, most of which require additional packages. This section explains how to create presentations as a PDF file using the LaTeX bearer document class.

The first line enables the Beamer specific commands for Org mode explained below. The next two lines set the LaTeX exporter to use the Beamer class and to use the larger font settings. Using large text is good as it minimises the amount of test that fits on a slide. Who has not seen a presenter revealing a new slide and say: “I know you can’t read this but . . . ”, so don’t be like them and keep text on a slide to a minimum.

The following line sets the theme for the presentation.

```
#+STARTUP: beamer
#+LaTeX_CLASS: beamer
#+LaTeX_CLASS_OPTIONS: [bigger]
#+BEAMER_THEME: Rochester [height=20pt]
```

8

Administration: Manage Your Projects

8.1 Getting Things Done

In our time-poor world, everybody wants to cram more stuff into their day. Judging by the abundant literature, you can become so productive that "getting things done" (Allen, 2005) only takes a "four-hour work week" (Ferriss, 2011). We achieve these almost magical powers by learning from the "habits of highly effective people" (Covey, 1990) who seem to "eat frogs" for breakfast (Tracy, 2016).

Methods to increase your productivity a dime-a-dozen on the internet and productivity books with well-meaning advice are available in every bookshop. In the wake of all this great advice, developers release a plethora of software to help you become more productive. The market is saturated with solutions, from software behemoths such as Microsoft's OneNote to nimble mobile apps such as Todoist.

The quest to become more productive has become a popular past time. Millions of people around the world struggle with the demands of life in the twenty-first century and they constantly look for ways to cram more activities into their day. Self-proclaimed productivity gurus have published piles of popular books about methods to get more done in your limited available time.

All these methods boil down to three basic phases: set a goal, define the actions to achieve that goal and undertake these actions. These principles might sound simplistic, but it is the basic truth. The plethora of methods merely discuss different ways on how to manage these three steps effectively and efficiently. A central theme of these methods are means to process the enormous amount of information that people are exposed to every day.

In the slipstream of the productivity gurus, software developers have released many apps to manage the information stream of our daily lives. The internet contains many stories about people who moved from one application to the next, in search of the perfect way to manage their projects and tasks. I was one of those people, until I started using Emacs.

Org mode is an ideal system to help you getting things done. The Org mode package not only allows you to write prose, it is also helps you to manage actions and projects. As with any other Emacs package, it provides virtually unlimited freedom to implement your favourite method to get stuff done.

This chapter shows how to use Org Mode to manage your projects and tasks, loosely based on David Allen's *Getting Things Done* (GTD) method. David Allen describes iterative five steps to become more productive (Allen, 2005):

1. *Capture*: Empty your mind
2. *Clarify*: Describe what it all means
3. *Organise* Place it where it belongs
4. *Reflect*: Reflect on your progress
5. *Engage*: Take action

8.1.1 *Capture*: Empty Your Mind

One of the reasons we are often not as productive as we like is because our minds are filled with issues. When our brain is full of thoughts about what we should be doing, we start to worry about how busy we are instead of doing the thing we need to do. A brain only has a limited capacity as we can only have one thought at a time. When your brain is full of thoughts about what you should be doing, creative and productive thoughts are suppressed. The other problem with keeping ideas in your head is the risk that they will disappear in the fog of your brain. I am sure you all recognise the experience of having the most wonderful idea when enjoying your morning shower, but half an hour later you are unable to recall your gem.

The first step to getting things done is to empty your mind. This is not a Buddhism-inspired quest for enlightenment, but a simple technique to help you focus.

Open a new Org Mode file with the `find-file` command (`C-x c-f`). You can call it something like `todo.org`, or whatever you fancy. For the next fifteen minutes, write down everything that is floating around in your head. Dump the contents of your brain into this virgin Org Mode file. Try not to multitask when you empty your mind, but fully focus your attention on the list.

Start every new idea with an asterisk so that they become headlines. Your list will contain a jumble of random things. From minor household tasks to big future projects you want to do one day. Don't filter your thoughts. Just write them down. For the next twenty-five minutes, focus only on this task and write down everything that is in your head. Don't multitask and give this activity your undivided attention. Multitasking is the enemy of productivity because our brains can only focus on one intellectual activity at a time. The fact that magicians can so easily fool people illustrates why multitasking is a fool's errand. Perhaps listen to some music with the Emacs EMMS package to keep you focused.

Don't spend any time thinking about these tasks. Don't worry about when you'll need to do it or in what order they need to be done just yet. Ensure that your mind is empty by the time you complete your list. If the list only has about a dozen items, then your list is not complete. For most people, fifty actionable items, projects and fuzzy goals are not unusual. If you are overwhelmed by the list then don't shoot the messenger as it merely reflects your life. If it takes you longer than twenty five minutes to empty your mind, then take a five minute break and start again.

You now have a long list of all the stuff you need to get done. But that is only the starting point. Org mode also has functions that can capture tasks as they come up in random moments of the day.

```
* Mow the lawn
* Clean up the backyard
* Improve my job skills
* Learn how to use Emacs
* Write an ebook about ... (fill in your speciality)
* Empty your e-mail inbox
* Prepare presentation for the quarterly meeting next week
* And so on, and so on ...
```

1. Using the Inbox

8.1.2 *Clarify*: Describe what it all means

Did you notice that most things on the list above, apart from items 6 and 7, are more work than just one action? In GTD-speak, they are projects, which are desired outcomes that take more than one action to complete.

Your list will be a mess of tasks, projects, goals and vague ideas. Your next task is to organise them. Firstly you can add some order and hierarchy to the list with the `ALT` and arrow keys. `M-up` and `M-down` will move a heading up or down, while `M-right` and `M-left` will promote or demote your

entry. With these four keystrokes, you can organise your list so that projects and tasks go together and create some order in the chaos that came from your mind. So the garden becomes a project with two tasks:

```
* Gardening
** Clean up the backyard
** Mow the lawn
```

You can also add notes or link images below any heading to provide some context to the task by hitting enter at the end of the headline and typing. Your list is starting to take shape now. The following steps will add more context to your tasks.

8.1.3 *Organise* Place it where it belongs

The problem with most todo-lists is that you get overwhelmed by the amount of stuff to be done. But in reality, most actions don't need or can't be progressed. You can be more precise in your records and mark items as the next action to be undertaken, or mark the ones where you are waiting for somebody else.

- *Next*: The next action to be taken.
- *Todo*: Something to be done in the future (either scheduled or as yet undetermined)
- *Waiting*: If you are waiting for somebody to do something, then mark it as such.

Org mode can associate each headline in a document with a workflow state. By default, there are only two states, `TODO` or `DONE`. You can change the state of a heading with the shift and arrow keys. Org mode will cycle between the two states. The keywords are commonly written in all caps, but that is not necessary.

You can define workflow states any way you like. Adding the following Lisp lines to your init file will set your workflow states. The states after the vertical bar will be marked as completed, usually coloured green.

If you have many states, cycling through them can be tedious. The `C-c C-t` command provides a popup menu to quickly select your option. You add the menu letter between parentheses after the keyword like this:

```
(setq-defualt org-todo-keywords
              '((sequence "TODO(t)" "NEXT(n)" "WAITING(w)" "|"
                          "DONE(d)" "CANCELLED(c)")))
```

Emacs Writing Studio roughly follows the GTD approach and have a status for next actions, waiting for others or future actions. When an action is completed, it is either done or cancelled. There are, in principle, no limitations to how you use this functionality.

The basic principle of the GTD approach is not to have massive lists of items that you like to do one day but that you define the next action that needs to be done to achieve your goal, or schedule activities in your diary. This method ensures that only a subset of activities is in your consciousness and you don't get overwhelmed by your inactivity as action lists tend to be long list of promises to our future selves.

You can set different keywords can for each Org file. When, for example, you are writing a book, you could set specific keywords for each heading with the following line in your header:

```
#+TODO: TODO(t) DRAFT(d) | DONE(c) EDITED(e)
```

Now that your list is nicely organised in the stuff you need to do, add dates to some of the tasks. Scheduling a task to a specific date is to commit your recalcitrant future self to the action.

You can add the date that the task is scheduled or a deadline by which is should be completed. A scheduled date indicates the date or period you plan to work on the task. A deadline is the time the task needs to be completed.

To add a scheduled date, use `C-c C-s` when on a headline. Emacs will pop up a calendar that you can use to select a date. The shift and arrow buttons move the timestamp by day or week. The < and > symbols move you a month in time. Press enter when done, and the date appears below the headline. You can add a deadline in a similar way but with the `C-c C-a` keystroke.

Timestamps use the ISO 8601 format: year, month, day. This format avoids any confusion between American formats and the rest of the world. Editing a timestamp is easy. Place your cursor on either the year, month or date and use the arrow keys to move it up or down.

```
* TODO Complete Org-Mode article
SCHEDULED: <2021-05-08 Sat>
```

Scheduled dates can also set a regular schedule. For example, suppose you add, for instance, `+7d` at the end of the date. In that case, Org mode recalculates the date every time you complete the task and resets the status to `TODO`. You can also use the letters `w`, `m` and `y` for scheduling a job weekly, monthly or yearly. Some actions have to be undertaken weekly on the same day, but in some instances it is better to restart the clock every time it is completed.

My tax return has a deadline of 30 September and appears annually on that date, indicated by `+1y`. It is a deadline instead of a schedule because the tax office enforces this on me.

The next action is my weekly review of the inboxes. In this case, the seven days are recalculated every time I complete the action. So if I complete the review on 13 May instead of 11 May, the new date will become 20 May. The double plus symbol recalculates the new date from the day of the status change.

The last action states that I need to clean the dishwasher once each month. The `.+` indicate t

```
* TODO Submit tax return
DEADLINE: <2021-09-30 Mon +1y>

* TODO Weekly review
SCHEDULED: <2021-05-11 Sun ++7d>

* TODO Clean the dishwasher
SCHEDULED: <2021-05-11 Sun .+1m>
```

Only add a scheduled date if this is the time that you plan to do the action. Try not to add too many self-imposed schedules because you will over commit your day. A deadline is only helpful if there is an external expectation that you need to complete something by a specific day, for example, get travel insurance before your flight leaves.

Some todo items in your list could use a checklist to remind yourself of the required steps. Org mode allows you to add checkbox items anywhere in your document by adding `[ ]` after a list indicator. Using `M-S <RET>` after a tick box item creates a new tick box. Ticking and un-ticking any items is as simple as hitting `C-c C-c` (`org-toggle-checkbox`). The snippet below is an example of a todo item with a deadline, some notes and a checklist.

```
* TODO Submit tax return
  DEADLINE: <2021-09-30 Mon +1y>
  Tax accountant: 0407 555 283
  - [X] Collect records
```

```
- [ ] Prepare overview
- [ ] Set appointment with accountant
```

You can convert a plain list item to a checkbox item, or vice versa, with `C-u C-c C-c`. You can also convert list items to headings and back again. The `org-ctrl-c-star` function (`C-c C-*`) converts a paragraph to a heading. If the line contains a checkbox, it becomes a TODO item. Using `C-c C--` (`org-ctrl-c-minus`) converts a paragraph or a heading to a list item.

8.1.4 *Reflect*: Reflect on your progress

After a week or so, your inbox will start filling with stuff. Your inbox is not just one location, but a collection of places where you gather information. *Emacs Writing Studio* uses the Org mode capture mechanism as an inbox, but it can also be a physical inbox to collect papers. A digital notebook on your phone or a physical diary to collect notes are also viable options.

My personal inbox consists of my Org mode inbox file, my email inbox, a cloud-based notes application in my phone a physical inbox and my physical diary. As part of my weekly review I promise myself to empty these locations and process them into my system using this flowchart, which is based on the classic GTD model.

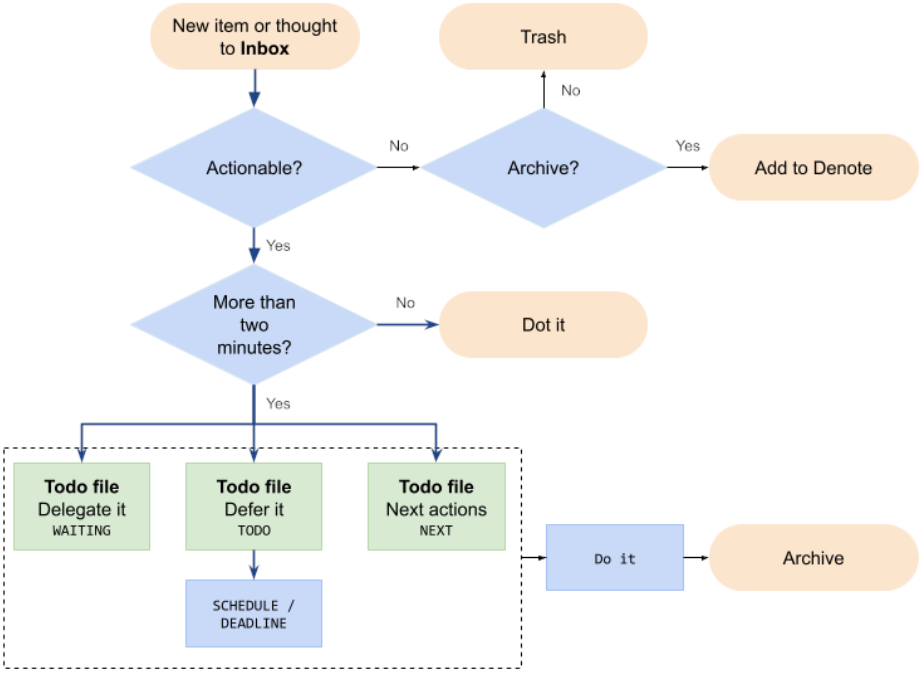

Figure 8.1 Example of a productivity workflow using Org mode.

During your regular review, you assess each bit of information you collected. If it is not actionable, the you either ditch it, otherwise capture it in as a permanent note.

If the new item is actionable and it only takes a few minutes to do, then just go ahead and do it; don't waste your time formally registering the action.

Lastly, if the inbox item is actionable, bur requires some time, you can add it as an action to your agenda file. The status of each action depends on external circumstances.

The key of any productivity workflow is to regularly review your list of actions, your priorities and goals. The central tool in Org mode to look at your list of registered actions is the agenda.

The agenda provides a summary of all your headings with an action status. Firstly, it is common practice to bind `C-c a` to the agenda menu. You can assign one or more Org mode files that contain your actions. Org mode will read these files and present the todo items as an agenda. In *Emacs Writing Studio*, the default agenda file is `agenda.org`.

When you evaluate the `org-agenda` function with `C-c a`, Org mode reads all agenda files and constructs an agenda, listing actions by date.

You can change the status of items from within the agenda or jump to the file that contains the action, so you can ready any context or maintain checklist items.

The agenda starts with a self-explanatory menu screen which you can explore. Org mode has extensive capabilities to configure how the agenda is presented, which are outside the scope of this article.

8.1.5 *Engage*: Take action

Emacs is a powerful multifunctional tool, but unfortunately, it cannot help you mow the lawn for you, go shopping or do the other tasks on your lists. Emacs can help you with any task that involves creating written content, but it can't mow the lawn for you. In the end, there is no productivity system in the world that does the tasks for you, no matter what the gurus promise. However, bringing order into your tasks keeps you focussed on your goals.

8.1.6 Learning More

Getting Things Done is just one of many methods to manage your busy life. This article shows how to implement the principles of David Allan's GTD method with Org mode. The beauty of Emacs is that you have the freedom to implement whatever method you prefer, so be creative and use Org mode to suit your needs.

Bavarian Org mode guru Rainer König has developed a comprehensive series of YouTube videos that explain using Org mode for managing actions and projects in great detail. Ranier has also published a more comprehensive course on Udemy, which provides more detail than the freely available videos (König, 2020).

8.2 Manage Files

8.2.1 Introduction

Working with Emacs means that you will need to access, create files and manage files on your drives. Emacs comes shipped with `dired` (pronounced *dir-ed*), a potent file manager. This article explains the basic principles of using dired and shows how to manage files with Emacs to organise your hard drive.

8.2.2 Basic Operation of Dired

Dired is short for "directory editor", but it can do much more than just that. This software has an illustrious history. The first version of dired was a stand-alone program written circa 1974, so its origins lie even further back in time than Emacs. The software comes packaged with Emacs and it provides a front end to various GNU core utilities to manage files.

You activate the file manager with the `dired` function or the `C-x d` shortcut. After selecting the relevant directory Emacs creates a buffer with the content of the directory of the selected buffer.

Another useful function is `dired-jump` (`C-x C-j`) which opens Dired and jumps to the file linked to the current buffer.

You can navigate the content with the arrow keys and press `j` to jump to a specific file by entering part of its name. The carat key `^` takes you to the parent folder.

The `q` button closes the dired window but does not kill (remove) it. Note that every time you open a new directory in Dired, Emacs opens a new Dired buffer. After a while, you litter your Emacs session with unused Dired buffers. Pressing the `a` key opens a directory in the same buffer. This functionality is disabled by default because the Emacs developers strangely believe that new users find it confusing. *Emacs Writing Studio* configures this behaviour by default.

Denote includes a minor mode that formats compliant filenames in the directory editor so it is easy to recognise the individual items of the note's metadata. The filenames not only provide metadata for the note itself, they are also a heuristic to make it easy to find notes based on date, signatures, title or keyword (Figure 8.2).

```
-rwxr-xr-x  1  11K 2023-05-14 20:41 20040405T090519--stoic-theory-of-universals__ola_philosophy_phl33_prevosnet.org
-rwxr-xr-x  1  30K 2022-11-27 15:12 20040423T175900--hellenistic-philosophy__bib_ola_philosophy_phl33.org
-rwxr-xr-x  1  23K 2022-08-27 12:48 20040530T202222--arguments-for-hedonism__ola_philosophy_phl33_prevosnet.org
-rwxr-xr-x  1 6.9K 2022-11-27 15:07 20040614T112934--sociological-imagination__bib_prevosnet_sociology.org
-rwxr-xr-x  1  26K 2023-05-19 08:14 20040725T200459--a-life-history__bib_ola_prevosnet_sgy110_sociology.org
-rwxr-xr-x  1  19K 2022-11-27 16:06 20040815T204249--australian-society-foundations-of-social-science__bib_ola_sgy110_sociology.org
-rwxr-xr-x  1 6.2K 2022-09-14 15:56 20040815T210806--breakfast-culture__ola_prevosnet_sgy110_sociology.org
-rwxr-xr-x  1 7.6K 2022-09-18 08:16 20040823T205532--the-web-of-cultural-identity__ola_prevosnet_sgy230_sociology.org
```

Figure 8.2 Extract of Denote files in Dired.

The enter key opens the respective file or directory. To open a file or directory in another window, press `o` (Using `C-o` open the file in another window, but the cursor stays in the Dired window). Emacs is a Swiss-Army chainsaw, but it cannot do everything. Sometimes you might like to open a file in other software, such as your image editor or video player. You can open files with external software by pressing `&` after which dired will ask for the appropriate software. You need to type the name of the executable file of the software you like to use, e.g. `gimp`.

To copy a file, press the `C` button. Dired will ask for a new name and location in the minibuffer. To move a file, you press `R` because moving a file is the same as renaming it with a new directory.

There is no need to close the buffer before you rename an open file. Emacs will link the open buffer to the new filename.

If you have two open dired buffers, *Emacs Writing Studio* copies and moves from the folder in the active window to the other dired buffer.

Renaming the file is the same as moving it. So press `R` and type a new filename.

It is sometimes useful to copy the name of a file to the kill ring with the `w` key, so you can use it to rename the file. So to rename a file, copy the name with `w`, rename the file with `R` and paste the existing name with `C-y` and edit the name to your new version.

You can select and deselect files for deletion (killed) with the `d` and `u` buttons. After you selected the files you like to delete, press `x` to execute the deletion. Press capital `D` if you like to remove a single file. When you delete or trash a currently open file, Emacs will also ask you to close the appropriate buffer. By default, Emacs permanently removes files. The *Emacs Writing Studio* is configured so that files are moved to the recycle bin.

You can select multiple files to work on at the same time by marking them. The `m` button marks a file, and the `u` removes the mark. The capital `U` removes all marks in the buffer. The `t` key reverses your markings, which is helpful when you want to select everything but one or two files.

This method requires you to manually select each file. You can also use regular expressions to select files. Press `% m` to open the regular expression selection prompt. For example, `^2023.*_journal*` selects all Denote files that start with the `2023` and that have the `journal` file tag. Now press `t` to invert the selection and `k` to remove the selected files from view. This sequence is a useful method to find related files.

After you selected multiple files in this manner, you can use all file commands to act on the selected targets, for example moving all 2023 files with the `_journal` tag to another folder.

If your head is buzzing with all the different key bindings, the table lists the functionality described in this chapter. The keybindings in Table are only a small snapshot of the functions of the directory editor in Emacs. You can press the `h` key while in a Dired buffer to view all functionality and related keybindings.

Table 8.1 Dired key bindings.

Key	Function	Action
`a`	`dired-find-alternate-file`	Open folder in same buffer
`C`	`dired-do-copy`	Copy a file
`j`	`dired-goto-file`	Jump to the file linked to active buffer
`g`	`revert-buffer`	Refresh the dired buffer
`m`	`dired-mark`	Mark file under the cursor
`% m`	`dired-mark-files-regexp`	Mark by regular expression
`o`	`dired-find-file-other-window`	Open file in other window
`C-o`	`dired-display-file`	Display file in other window
`q`	`quit-window`	Close the buffer
`R`	`dired-do-rename`	Rename (move) a file
`t`	`dired-toggle-marks`	Inverse marked files
`u`	`dired-unmark`	Unmark file under the cursor
`U`	`dired-unmark-all-marks`	Unmark all files
`&`	`dired-do-async-shell-command`	Open file with other program
`enter`	`dired-find-file`	Open file

8.2.3 File Naming Conventions

In the world of computing there is perhaps nothing ore personal than the names people give to files. While this almost infinite freedom to name a file any which way you prefer is a pinnacle of individual expression, it can lead to problems when managing large projects. Back in the days when offices on held paper archives, they had strict rules on how docents should be archived. Misplacing a piece of paper in an archives stretching many meters of folders meant that you would probably never find that document again. When in the 1980s office workers started to use computers, all such rigour and process was thrown out to the window and ultimate freedom and the associated chaos emerged.

The Denote file naming convention is good for any document where the data of creation matters.

My personal naming convention is expressed in the default settings of *Emacs Writing Studio*. My documents directory holds a subdirectory for my notes, projects and the bibliography.

There is no need to make copies of your files at each stage of the writing process with names such as `principia-methematica-rev.1.org`, `principia-methematica-draft.org` and other variations on this theme. Using a version control system can

8.2.4 Recent Files and Bookmarks

Whenever you return to Emacs you might want to open a file you were working on recently. The recent files minor mode (`recentf-mode`) provides a transient list of the files you most recently opened.

This minor mode saves the most recent opened files when you exit Emacs to a file in your configuration folder. However, it might be more useful to save the recent files regularly to ensure it is

saved. The `run-at-time` function runs a function at a regular interval, in this case every five minutes. The `recentf-edit-list` function opens the file with your recent acquisitions and lets you delete selected files.

By default, the recent files mode stores the last twenty opened files, which you can change by modifying the `recentf-max-saved-items` variable.

Recent files are transient as they are continuously updated as you open new files. For a more permanent list of files you like to open, use bookmarks.

You can store a file as a bookmark with `C-x r m` (`bookmark-set`). The bookmark will also store the location of the cursor, so you can maintain multiple bookmarks for a file. The default name for the bookmark is the name of the file. You can also enter a bespoke name in the minibuffer before hitting `ENTER`.

To view a list of all bookmarks in the minibuffer and select the one you like to open, use `C-x r b` (`bookmark-jump`).

Bookmarks are saved in the `bookmarks` file in your configuration folder every time a new bookmark is created. The `bookmark-save-flag` is set to one so that the bookmarks file is saved every time you add a new one. The default value only saves it when you exit Emacs, which mean you could loose bookmarks in the unlikely event of an Emacs or system crash.

If you like to remove bookmark no longer required then use the `bookmark-delete` function, which has no default keybinding but is bound to `C-x r D` in the *Emacs Writing Studio* configuration.

8.3 Viewing Images

You have already seen that Org mode can embed images and export these to the desired format. Emacs also has some built-in functionality to help you manage your collection of images.

Image-mode and the image-dired package are bundled with Emacs, so there is no need to install packages, but you might need some additional software. You can view images without external software, but you cannot manipulate them. The ImageMagick software suite provides functionality for editing and manipulating images.

Emacs can display various popular image formats out-of-the-box with image mode. You can open an image file directly with `find-file` (`C-x C-f`) or through the directory editor (dired). You can also open a linked image from within an Org file with `C-c C-o` (`org-open-at-point`) with the cursor on the image. Emacs automatically scales the image to snugly fit inside the display window. A range of keyboard shortcuts are available to view images. The `n` and `p` keys (next and previous) or the left and right arrows flick though the images in the current directory. Image mode also provides several commands to change the display size of images (I am unsure why the prefix key is `s` for some commands, but `i` for others, but alas):

- `s o`: Show image at original size (when it doesn't fit in the window, scroll through the image with the arrow keys).
- `s w`: Fit the current image to the height and width of the window.
- `i +`: Increase the image size by 20%
- `i -`: Decrease the image size by 20%.

Furthermore, image mode can manipulate images, with the assistance of ImageMagick:

- `i r`: Rotate the image by 90 degrees clockwise.

- `i h`: Flip the image horizontally.
- `i v`: Flip the image vertically.
- `i c`: Crop the image.
- `i x`: Cut a rectangle from the image and replace with black.

The crop and cut commands display a rectangular frame superimposed on the image. Use the mouse to move and resize the frame. Type `m` to move the frame instead of resizing it and type `s` to convert the frame to a square. When you are satisfied with the result, type `enter` to crop or cut the image. You can exit the crop and cutting menu with `q` without changing the source file. Please note that these commands are only available when *ImageMagick* is installed.

If you like to retain the result of the transformation, press `i o` to save the image under a new name. When you are done with watching images, use `q` to quit the image buffer, or `k` to kill the image buffer altogether.

8.3.1 The Image-Dired Package

Viewing images individually is great, but wouldn't it be nice if you cold see thumbnails before delving into your collection? The Image-Dired package provides a thumbnail buffer to view and maintain images from within a Dired buffer using thumbnails. Evaluate the `image-dired` function and select the directory you like to use. Emacs splits the screen and presents a thumbnail screen (of up to a thousand entries) to explore your collection. Emacs stores the thumbnails in the configuration directory for future reference.

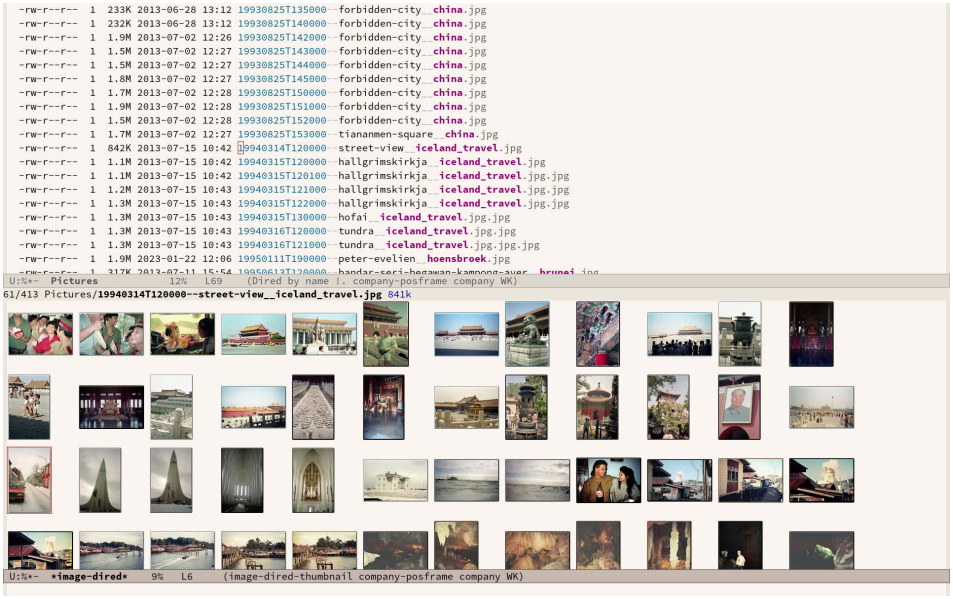

Figure 8.3 Viewing some travel photos in Emacs with image-dired.

Alternatively, when you are inside a Dired buffer that contains images, mark the images you like to view and generate the thumbnails with `C-t d` (`image-dired-display-thumbs`). If you don't mark any files, the program uses the image under the cursor.

Yet another method of previewing your images is by marking the ones you need and generate inline thumbnails inside the Dired buffer with `C-t C-t` (`image-dired-dired-toggle-marked-thumbs`). The same shortcut also removes the thumbnails.

The active image is marked with a flashing border around the thumbnail and its filename is displayed on the top of the thumbnail window.

You can navigate the thumbnails with the arrow keys. The < and > keys take you to the start or end of the collection. You can remove a thumbnail from the collection with `C-d`. If you have selected more images than thumbnails that can fit on page, then image-dired tracks your movement, so as your cursor moves up or down, the thumbnails refresh.

To view an image, hit `enter` when the thumbnail is marked. You cycle through the marked images in your collection with the `space` and `backspace` buttons, or `C-<left>` / `C-<right>`.

But why manually flick through your album if you can let Emacs do this for you? The `S` key starts a slideshow with each image shown five seconds by default. You can configure the delay with the `image-dired-slideshow-delay` variable, or drive the slideshow manually with the `space` and `backspace` keys, or `C-<left>` / `C-<right>`.

The main image display is in image mode, so all the actions described in the previous section apply.

As usual, `q` quits the image or thumbnail window.

The image-dired software can also create a plain text database of images with searchable tags and thumbnail descriptions.

You tag images directly from a dired buffer. The `C-t t` keystrokes lets you tag the selected files. You can retrieve the tags in a future session using `C-t f` and view the image thumbnails with `C-t d`, as described above. You can tag images from within a thumbnail buffer with the `t t` shortcut. You can also add a comment to the image by pressing `c` in the thumbnail viewer.

The file name, tags and comments show in the mini buffer as you move through the image thumbnails: `directory: filename (tags): comment`.

Emacs stores the metadata in the `image-dired` directory in your configuration folder in the `.image-dired_db` file. This is a plain text file that lists each file name and the tags an comment.

Retrieving tagged files only selects images with tags in the directory of the current dired buffer. It does not work across multiple folders.

This tagging system is nice, but it creates an integrity vulnerability in that it separates tags and files. When you accidentally remove the database, all metadata is lost. You will also loose access to the metadata when you rename an image.

An alternative method to tag files is to use the Denote file naming convention, which encodes metadata into the name of the file. Denote is a package to produce and maintain plain text notes, but it can also be used to maintain a collection of binary (non-text) files. When using the for your images, you can mark parts of you collection using regular expressions. The Denote file naming convention encodes four bits of metadata about a file into the filename. Only the timestamp is compulsory and serves as a unique identifier:

- Timestamp in ISO 8601 format
- An alphanumeric signature (starting with double equality sign (==)
- The title in kebab-case (starting with double dash ==)
- Keywords in $\text{snake}_{\text{case}}$ (starting with double underscore __)

For example, one photo in my collection is: `19930825T132000--forbidden-city-throne-palace-of-heavenly-purity__china.jpg`. So this photo was taken on 25 August 1993, the title describes the topic of the photograph and

I can now go into Dired and mark files with regular expressions, searching for each of these fields. For example, using `%m _china` marks photos taken in China, or `%m -city` all images with the word “city” in their title. Using the Dired convention for naming files is not only great for notes, it also helps you manage your photo collection.

The `denote-rename-file` function (`C-c w R`) lets you construct a Denote-compliant file name for existing images or other binary files. This function uses the last modified date as the identifier by default. But when you are sorting old collections you might want to add a date that is different to the last modified one. Adding the universal argument (`C-u`) will also ask you to provide a date and time, i.e. (`C-u C-c w R`).

Emacs is a powerful piece of software but it is mostly limited to editing text. To meaningfully work with images, you will need another package. Image-dired lets you open images in external viewers or editors by setting one variable.

Pressing `C-enter` opens the file in an external viewer or editor. You can tell Emacs which external viewer to user by configuring the `image-dired-external-viewer` variable. I linked it to the GIMP (GNU Image Manipulation Program). The content of this variable obviously depends on your system and preferred viewer.

When you are in an image-dired buffer, you open the external viewer with `C-enter`. Confusingly, when you are in a normal dired buffer, this function is bound to `C-t x`. *Emacs Writing Studio* remaps the keys so that you can use `C-enter` in dired and in image-dired to open an image in your favourite external editor.

9

Become an Emacs Ninja

Now that you have reached the end of thie book you know enough to start publishing a website or write a book or whatever else you want to share with the world. You have mastered the steep part of the Emacs learning curve. However, as you use Emacs for a while you will undoutably want to finetune it to your needs and become and

There are five steps to becoming an Emacs ninja.

1. Understand the basics
2. Modify your init file by hand
3. Create simple functions to make your life easier
4. Build a package
5. Help others

This book has helped you to take the first step

9.1 Some more random functions

9.1.1 Bookmarks

9.1.2 Keyboard Macros

Writing and editing can sometimes be a repetitive task. Lets assume that your editor tells you that ...

`F3` starts recording a macro (`kmacro-start-macro-or-insert-counter`).
`F4` completes the recording (`kmacro-end-or-call-macro`).

9.2 Migrating to Emacs

Now that you are well on your way in your Emacs journey, you might want to consider converting your old files to Org mode or another plain text format.

The Pandoc program is your friend and can assist with a seamless transition to Emacs. Pandoc (`pandoc.org`) is a universal (pan) document (doc) converter. It can covert almost any file format. The program works in the command line. To convert an MS Word document to org mode and extract any images, use this line in your Linux or macOS shell or Windows Powershell. The first two parameters instruct Pandoc to extract images to the images folder. The `-o` option defines the output file name and format. You can use the same approach with any input format that Pandoc supports, just replace the input filename.

```
pandoc --extract-media ./images input.docx -o output.org
```

If you need to convert a folder full of files, this method can be a bit tedious. In Linux and macOS you can use the line below to convert all Word documents to Org mode files.

```
for file in *.docx; do pandoc --extract-media ./images "$file" -o "${file%.docx}.org";
done
```

Windows users need to evoke this Powershell script to achieve the same results:

```
Get-ChildItem -Filter *.docx | ForEach-Object {
        $newName = $_.Name -replace '.docx$', '.org'
        pandoc --extract-media ./images "$_" -o $newName
    }
```

9.3 Is Emacs a productivity sink?

Configuring Emacs can be a daunting task that can take a lot of time and can become a productivity sink because of the easy by which it can be done. Within the Emacs community, a discussion emerges every now and then about the need to make so many changes to the vanilla software to get it to behave how you like it. This may seem like a reasonable question.

The freedom you enjoy in Emacs means that everybody will have specific preferences, depending on how their needs. The Emacs developers cannot cater to every personal preference so they provide a skeleton system that you need to develop to suit your personal workflow. Emacs Lisp is like a box of Lego. You can play with it out of the box using vanilla Emacs, but it is much more fun when you create your own toys. Working on your Emacs configuration is like building with Lego, it is a lot of fun, but not the same as actually playing with it.

It can be tempting to constantly fine-tune your configuration, which can become a productivity sink. Wielding the power to create a bespoke Emacs system is a great temptation that can lead to fake productivity, which one of the three forms of procrastination:

1. *Nihilistic*: Watching TV, playing computer games.
2. *Sophisticated*: Fake productivity, e.g. Emacs hacking, switching productivity tools, taking notes for the sake of volume instead of quality.
3. *Productive*: Daydreaming.

Tinkering with your Emacs configuration is not as bad as nihilistic procrastination, but can become a form of fake productivity. The productivity gains from fine-tuning your Emacs to cut out a few keystrokes can take more time than you will save with your new workflow. The act of writing is about much more than the number of words you can type into your buffer. Writing is as much a contemplative act as it is about productivity.

Hopefully the *Emacs Writing Studio* configuration will help you reduce the time it takes to configure Emacs to suit your preferred workflow.

9.4 Communication

9.5 Learning more about Emacs

This book is not a comprehensive manual of Emacs but an opinionated description of how to achieve a series of tasks. The text in this book is opinionated because it describes only one or two ways of completing each objective. The flexible nature of Emacs provides many ways to achieve the same thing. This freedom is both a strength and a weakness. This book aims to shorten the learning curve as much as possible so that you can become creative and productive. If, after reading this book, you would like to know more, I recommend you read other books such as *Learning GNU Emacs* by Debra Cameron, *Mastering Emacs* by Mickey Petersen or the built-in Emacs manual (Stallman, 2023; Petersen, 2022, **?**).

This book has only scratched the surface of the myraid of possibilities that Emacs has to offer. Due to its extensibility,

9

Appendix

I learned writing computer code by typing long listings of Atari BASIC code and slowly but surely understood its syntax and logic. This method is also a great starting point to learn Emacs Lisp. The purpose of this configuration is for you as a starting point. Feel free to modify anything in the *Emacs Writing Studio* (EWS) initialisation file and study its effects.

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.

This appendix presents and explains the *Emacs Writing Studio* configuration. This configuration stays as close to vanilla GNU Emacs as is humanly bearable for an author. The configuration is annotated to explain the logic of he code and provides with some options for enhancements or additional functionality. This configuration follows the following principles:

- Leverage functionality of the latest version of GNU Emacs
- Use standard keyboard shortcuts
- Centred around Org mode
- No configuration for writing code

9.6 Using *Emacs Writing Studio*

The first part of the configuration sets the basic principles of the EWS configuration, such as package management, the user interface look and feel, the minibuffer completion system and basic settings to enable writing for humans. Chapter 3 describes using this configuration.

9.6.1 Basic Configuration

The first part of the configuration checks if the latest version of Emacs is running. EWS leverages some of the latest functionality, so you will need to install this version. Note that the expression `(< emacs-major-version 29)` is in Polish notation. In Lisp, the operator is placed before the operands, unlike the more common infix notation which places the operator between the operands, e.g. `emacs-major-version < 29`.

Any changes made by the Emacs customisation system in EWS are stored in `custom.el` instead of directly to the init file. This approach prevents the customisation system modifying the initialisation file. The `custom.el` file is loaded when found. If variables are set in both the custom file and the init file, the latter takes preference.

```
(when (< emacs-major-version 29)
  (error "Emacs Writing Studio requires Emacs version 29 or later"))
```

```
;; Custom settings in a separate file and load the custom settings
(setq custom-file (expand-file-name "custom.el" user-emacs-directory))

(when (file-exists-p custom-file)
  (load custom-file))
```

9.6.2 Packages and External Software

This configuration heavily relies on John Wiegley's Use-Package. This package makes it easier to install and configure packages with a standardised and easy to read method. Software developers call such a tool 'syntactic sugar', which is syntax designed to make code easier to read or write, making the language "sweeter" for humans (Landin, 1964).

Emacs users have developed and shared thousands of packages with the rest of the community. The first part of the configuration sets the basic elements to load and install packages from the MELPA archive, in addition to the default ELPA repository.

The Use-Package system consists of a set of statements between parenthesis, which in this case is a macro. In it's simplest form it is something like `(use-package <package-name>)`. The code can also contain one or more sections to set various options. The `:config` section in the first part evaluates code after the package is loaded, which in this case adds MELPA to the archives and initialises the package manager. The `:custom` section in the second Use-Package macro sets three variables, documented in the comments.

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, which are discussed in the Appendix.

To read the finer details of the Use-Package macro, read the manual with `C-h R use-package`.

```
(use-package package
  :config
  (add-to-list 'package-archives '("melpa" . "https://melpa.org/packages/"))
  (package-initialize))

(use-package use-package
  :custom
  (use-package-always-ensure t)
  (package-native-compile t)
  (warning-minimum-level :emergency))
```

9.6.3 Emacs Writing Studio Functionality

EWS also provides a range of bespoke convenience functions for various aspects of the writing process. This package is not available on any Emacs repository, but is loaded directly from the GitHub repository.

```
;; Load EWS functions

(load-file (concat (file-name-as-directory user-emacs-directory) "ews.el"))
```

The `ews-missing-executables` function checks if external software is available on your system. Emacs writes a message in the minibuffer if any of the recommended tools is missing. You can jump to the `*Messages*` buffer with `C-h e` to review the output. If packages are missing, then Emacs will function normally, but some features might be unavailable. The relevant chapters in this book provide more details which software is required and the tasks it undertakes.

The input for this function is a list, which is a series of strings between parenthesis that starts with a tick symbol. This symbol prevents Emacs confusing the list of data with a function. In this example, the list also contains other lists.

This function looks whether all these packages are available on your system. Any software in a nested list, such as `("convert" "gm")`, only one of them has to be available as these programs are alternatives for the same functionality.

```
(ews-missing-executables
 '("soffice" "zip" "pdftotext" "ddjvu"
   "curl"
   "dvipng"
   "dot"
   ("convert" "gm")
   "latex"
   "hunspell"
   ("grep" "ripgrep")
   ("gs" "mutool")
   "pdftotext"
   ("mpg321" "ogg123" "mplayer" "mpv" "vlc")))
```

9.6.4 Look and Feel

The basic idea is to create a clean and keyboard-centric writing interface with minimal distractions.

The first three lines of code for the EWS look and feel disable the toolbar, the menu bar and the scroll bar. The menu bar can be useful for beginners and you can still access it with the `F10` key. If you like to keep the tool, menu and/or scroll bars, then either remove the relevant lines, change the `-1` to a `1` or add two semi colons at the start of the line to convert them to comments.

Vanilla Emacs has the slightly paternalistic habit to require only a single `Y` or `N` answer while on some occasions it requires you to type `yes` or `no`, due to the perceived higher risk of typing the wrong answer. The `setq` function sets the `use-short-answers` variable to `t`. If you like to retain this behaviour, then change the `t` into a nil. In Emacs Lisp, `t` means TRUE and `nil` is equivalent to FALSE. Confusingly Emacs documentation often mentions to set a value to "non-nil", which is a double negative to suggest setting the variable to true.

```
(tool-bar-mode -1)
(menu-bar-mode -1)
(scroll-bar-mode -1)

(setq use-short-answers t)
```

The next two sections of code further improve the Emacs interface with two packages by Emacs guru Protesilaos Stavrou from Cyprus. The spacious padding package creates some whitespace around windows, preventing crammed text on your screen. The Modus Themes package provides a

collection of light and dark themes. These themes conform with the highest standard for colour contrast between background and foreground values (WCAG AAA). They also are optimised for users with red-green colour deficiency.

The Spacious Padding package is used using only default values. The `:init` section contains code that Emacs evaluates when loading the package. In this case, it enables the Spacious Padding mode. The `:custom` section also sets the line spacing to a more spacious value. You can read the manual for this mode with `C-h R spacious`.

```
(use-package spacious-padding
  :custom
  (line-spacing 3)
  :init
  (spacious-padding-mode 1))
```

The Modus themes package is highly configurable. This Use-Package declaration contains a few sections. The custom section customises variables used in the package. In this case we instruct the package to use italic and bold fonts for emphasis and allow for fonts with fixed and variable pitch. The code also slightly increases the size of headings. You can toggle between a dark and a light version of this theme and the last variable defines which these to toggle between. EWS uses the tinted version of the themes, which you can modify.

The `:custom` section of the macro sets some variables to define fonts. This section also defines which themes are toggled when switching between light and dark themes. The default is the tinted versions. 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. To see the possible options for the Modus themes use the help file: `C-h v modus-themes-collection`. Read the package manual for details with `C-h R modus`.

The `:init` section activates the tinted version of the Modus-Vivendi (light) theme. The next section binds some keys to commands to either toggle between dark and light or select any of the available modus themes. All EWS custom keybindings start with `C-c w` as the prefix key and `C-c w t` as the prefix key for the two Modus theme functions. You can obviously change these The last section hooks the Variable Pitch mode to any buffer in text mode. This means that written prose is displayed in variable pitch, while metadata, code and other items are in fixed pitch. A hook is a construction in Emacs that associates modes with each other. In this case, variable pitch text will is enabled for all text mode buffers.

```
(use-package modus-themes
  :custom
  (modus-themes-italic-constructs t)
  (modus-themes-bold-constructs t)
  (modus-themes-mixed-fonts t)
  (modus-themes-to-toggle
   '(modus-operandi-tinted modus-vivendi-tinted))
  :init
  (load-theme 'modus-operandi-tinted :no-confirm)
  :bind
  (("C-c w t t" . modus-themes-toggle)
   ("C-c w t s" . modus-themes-select)))

(use-package mixed-pitch
  :hook
  (text-mode . mixed-pitch-mode))
```

This last code snippet in the look-and-feel section changes the way Emacs automatically split windows to favour vertical splits over horizontal ones to improve readability. This section also installs the Balanced Windows package which manages window sizes automatically. For example, when opening three windows and you close one, the remaining windows each get half the screen.

```
(setq split-width-threshold 120
      split-height-threshold nil)

(use-package balanced-windows
  :config
  (balanced-windows-mode))
```

Alternatively, you can add these settings directly to your `init.el` file by adding the following three lines, with your fonts and sizes of choice. Any settings you ...

```
(set-face-attribute 'default nil :font "DejaVu Sans Mono" :height 200)
(set-face-attribute 'fixed-pitch nil :font "DejaVu Sans Mono" :heigh 200)
(set-face-attribute 'variable-pitch nil :font "DejaVu Sans")
```

9.6.5 Minibuffer Completion

Emacs Writing Studio uses the Vertico-Orderless-Marginalia stack of minibuffer completion packages in their standard configuration.

```
(use-package vertico
  :init
  (vertico-mode)
  :custom
  (vertico-sort-function 'vertico-sort-history-alpha))


(use-package savehist
  :init
  (savehist-mode))


(use-package orderless
  :custom
  (completion-styles '(orderless basic))
  (completion-category-defaults nil)
  (completion-category-overrides
   '((file (styles partial-completion)))))

(use-package marginalia
  :init
  (marginalia-mode))
```

9.6.6 Keyboard Shortcuts Menu

The Which-Key package improves discoverability of keyboard shortcuts with a popup in the minibuffer. The columns are widened a bit to prevent long truncated function names. Due to the naming conventions in Emacs, most functions start with the package name, so some can be quite long.

```
(use-package which-key
  :config
  (which-key-mode)
  :custom
  (which-key-max-description-length 40))
```

9.6.7 Improved Help Functionality

Emacs is advertised as the self-documenting text editor. While this is not quite correct (if only computer code could document itself), almost every single aspect of Emacs is documented within the program itself. The *Helpful* package by Wilfred Hughes is an alternative to the built-in Emacs help that provides more contextual information. When, for example, asking for documentation about a variable, the help file provides links to its customisation screen.

```
(use-package helpful
  :bind
  (("C-h x" . helpful-command)
   ("C-h k" . helpful-key)
   ("C-h v" . helpful-variable)))
```

9.6.8 Configure Text Modes

Emacs is principally designed for developing computer code, so it needs some modifications to enable writing text for humans. Firstly we hook Visual Line Mode to Text Mode. Visual Line mode wraps long lines to the nearest word to fit in the current window.

By default, Emacs does not replace text when you select a section and then start typing, which is unusual behaviour when writing prose. The `:init` section enables a more common default so that selected text is deleted when typed over. The `:custom` section enables the page-up and page-down keys to scroll all the way to the top or bottom of a buffer. The last variable saves any existing clipboard text into the kill ring for better operability between the operating system's clipboard and Emacs's kill ring.

```
(use-package text-mode
  :ensure
  nil
  :hook
  (text-mode . visual-line-mode)
  :init
  (delete-selection-mode t)
  :custom
  (sentence-end-double-space nil)
  (scroll-error-top-bottom t)
  (save-interprogram-paste-before-kill t))
```

9.6.9 Spellchecking

Writing without automated spell checking would be very hard even for the most experienced authors. The Flyspell package requires the hunspell software to be available and the relevant dictionary. You might want to change the standard dictionary to your local variety with the `flyspell-default-dictionary` variable.

```
(use-package flyspell
```

```
  :custom
  (ispell-silently-savep t)
  (ispell-program-name "hunspell")
  (flyspell-default-dictionary "en_AU")
  (flyspell-case-fold-duplications t)
  (flyspell-issue-message-flag nil)
  (org-fold-core-style 'overlays) ;; Fix Org mode bug
  :hook
  (text-mode . flyspell-mode)
  :bind
  (("C-c w s s" . ispell)
   ("C-;"       . flyspell-auto-correct-previous-word)))
```

9.6.10 Ricing Org Mode

This part of the configuration sets a bunch of variables to improve the design of Org mode buffers. Org mode has a lot of other variables you can configure to change its interface, which are all explained in chapter 3. The `setq` function can take several pairs of variables and their new values as parameters, as shown in the example below.

You can easily add other variables or remove some to make Org mode look the way you prefer. For example, to enable alphabetical 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.

If you have no need for mathematical notation and LaTeX, then you should disable the `org-startup-with-latex-preview` variable to prevent error messages.

```
(use-package org
  :custom
  (org-startup-indented t)
  (org-hide-emphasis-markers t)
  (org-startup-with-inline-images t)
  (org-image-actual-width '(450))
  (org-fold-catch-invisible-edits 'error)
  (org-startup-with-latex-preview t)
  (org-pretty-entities t)
  (org-use-sub-superscripts "{}")
  (org-id-link-to-org-use-id t))

(use-package org-appear
  :hook
  (org-mode . org-appear-mode))
```

The Org-Fragtog package automatically toggles Org mode LaTeX fragment previews as the cursor enters and exits them. By default, the text is a bit small and can become unreadable when changing between dark and light themes. The `org-format-latex-options` variable controls the way the Emacs presents fragments. This variable is a list with properties such as colours and size. The `plist-put` function lets you change one of these options in the list. The foreground and background are set to take the same colour as your text. If you change from dark to light mode or vice versa, you might need to evaluate the `org-latex-preview` function (`C-c C-x C-l`) to change the preview images.

```
(use-package org-fragtog
  :after org
```

```
  :hook
  (org-mode . org-fragtog-mode)
  :custom
  (org-format-latex-options
   (plist-put org-format-latex-options :scale 2)
   (plist-put org-format-latex-options :foreground 'auto)
   (plist-put org-format-latex-options :background 'auto)))
```

Most of the features of Org-Modern have been switched off in the custom section because it might be better for beginning users as these settings hide the semantic symbols.

```
(use-package org-modern
  :hook
  (org-mode . org-modern-mode)
  :custom
  (org-modern-table nil)
  (org-modern-keyword nil)
  (org-modern-timestamp nil)
  (org-modern-priority nil)
  (org-modern-checkbox nil)
  (org-modern-tag nil)
  (org-modern-block-name nil)
  (org-modern-keyword nil)
  (org-modern-footnote nil)
  (org-modern-internal-target nil)
  (org-modern-radio-target nil)
  (org-modern-statistics nil)
  (org-modern-progress nil))
```

9.7 Inspiration

9.7.1 Read ebooks

The built-in Doc-View package can read various file formats with the assistance of external software. This configuration increases the resolution of the generated image file and raises the threshold for warning before opening large files to fifty MB (50×2^{20}). Section 4.1.1 explains how to use this package.

```
(use-package doc-view
  :custom
  (doc-view-resolution 300)
  (large-file-warning-threshold (* 50 (expt 2 20))))
```

DocView has some limitations compared to other document viewers. The text is displayed as a PNG file which limits search capabilities and makes copying text impossible. The `pdf-tools` package by Vedang Manerikar is much more versatile than DocView. It is unfortunately not included in *Emacs Writing Studio* because it is complex to install on non-Linux systems.

The Nov package by Vasilij Schneidermann provides useful functionality for viewing ePub books inside Emacs. The init section ensures that any file with an `epub` extension is associated with this package. Refer to section 4.1.4 on how to read ePub files.

```
(use-package nov
  :init
  (add-to-list 'auto-mode-alist '("\\.epub\\'" . nov-mode)))
```

There is currently a confirmed bug in Org mode (version 9.6.6) that overrides the associations between LibreOffice and Doc View mode. The code below is a workaround to reinstate the desired behaviour and associates the various file extensions with Doc View. The bug is slotted to be resolved in version 9.7.

```
(use-package ox-odt
  :ensure nil
  :config
  (add-to-list 'auto-mode-alist
               '("\\.\\(?:OD[CFIGPST]\\|od[cfigpst]\\)\\'"
                 . doc-view-mode-maybe)))
```

9.7.2 Bibliographies

These lines of code add two field types to BibTeX entries: keywords to help you order your literature and a link to a file so you can read any attachments in Emacs. The `ews-register-bibtex` files assigns the `.bib` files in the `ews-bibliography-directory` variable to the list of global BibTeX files. You need to set this variable to the location where you store your bibliography and restart Emacs if needed.

BibTeX mode has many more options that you can configure to modify all sorts of behaviour. This mode is unfortunately not very well documented.

```
(use-package bibtex
  :custom
  (bibtex-user-optional-fields
   '(("keywords" "Keywords to describe the entry" "")
     ("file" "Link to a document file." "" )))
  (bibtex-align-at-equal-sign t)
  :config
  (ews-bibtex-register)
  :bind
  (("C-c w b r" . ews-bibtex-register)))
```

BibTeX is old but stable software that was last updated in 1988 and has minor limitations. The BibLaTeX dialect is a more recent version that provides more functionality and flexibility. To change BibTeX Mode to BibLaTeX, change the `bibtex-dialect` variable in the configuration to BibLaTeX by adding the following line to your configuration:

```
(bibtex-set-dialect 'biblatex)
```

The Biblio package provides a useful interface to online literature repositories. The `ews-biblio-lookup` function makes this package a little easier to use.

```
(use-package biblio
  :bind
  (("C-c w b b" . ews-bibtex-biblio-lookup)))
```

```
(use-package citar
  :custom
  (org-cite-global-bibliography ews-bibtex-files)
  (citar-bibliography ews-bibtex-files)
  (org-cite-insert-processor 'citar)
  (org-cite-follow-processor 'citar)
  (org-cite-activate-processor 'citar)
  :bind
  (("C-c w b o" . citar-open)))
```

9.7.3 Reading Websites

Vanilla Emacs opens hyperlinks to the World Wide Web with your operating system's default browser. If you prefer to use EWW as the default, add this code to your configuration file: `(setq browse-url-browser-function 'eww-browse-url)`. You can configure the EWW search engine by configuring the `eww-search-prefix` variable.

```
(use-package elfeed
  :custom
  (elfeed-db-directory
   (expand-file-name "elfeed" user-emacs-directory))
  (elfeed-show-entry-switch 'display-buffer)
  :bind
  ("C-c w e" . elfeed))

(use-package elfeed-org
  :config
  (elfeed-org)
  :custom
  (rmh-elfeed-org-files
   (list (concat (file-name-as-directory
               (getenv "HOME"))
                  "Documents/elfeed.org"))))

(use-package org-web-tools
  :bind
  (("C-c w w" . org-web-tools-insert-link-for-url)))
```

9.7.4 Playing Multimedia Files

```
(use-package emms
  :init
  (require 'emms-setup)
  (require 'emms-mpris)
  (emms-all)
  (emms-default-players)
  (emms-mpris-enable)
  :custom
  (emms-browser-covers #'emms-browser-cache-thumbnail-async)
  :bind
  (("C-c w m b" . emms-browser)
```

```
   ("C-c w m e" . emms)
   ("C-c w m p" . emms-play-playlist )
   ("<XF86AudioPrev>" . emms-previous)
   ("<XF86AudioNext>" . emms-next)
   ("<XF86AudioPlay>" . emms-pause)))
```

9.8 Ideation

9.8.1 Org Capture

You could, for example, create a separate entry for a shopping list. You can access the configuration in the capture menu with the `C` button, which pops up the customisation screen for the `org-capture-templates` variable. Next click the `INS` button to add another entry and complete the relevant fields as below and save the new variable. The example below create a shopping list stored in a file in your Dropbox folder. Several mobile apps exist that can read Org mode files, so you can take your list to the shops if you have a means to synchronise the relevant files.

The possibilities for capture templates are extensive and depend on your individual use cases. Explaining the configuration of the Org capture options in detail is outside the scope of this website. The Org manual (`C-h R org ENTER g capture ENTER`) discusses developing capture templates in detail.

```
;; Fleeting notes

(use-package org
  :bind
  (("C-c c" . org-capture)
   ("C-c l" . org-store-link))
  :custom
  (org-default-notes-file
   (concat (file-name-as-directory ews-home-directory)
         "Documents/inbox.org"))
  (org-capture-bookmark nil)

;; Capture templates

(org-capture-templates
 '(("f" "Fleeting note"
    item
    (file+headline org-default-notes-file "Notes")
    "- %?")
   ("p" "Permanent note" plain
    (file denote-last-path)
    #'denote-org-capture
    :no-save t
    :immediate-finish nil
    :kill-buffer t
    :jump-to-captured t)
   ("t" "New task" entry
    (file+headline org-default-notes-file "Tasks")
    "* TODO %i%?"))))
```

9.8.2 Denote

```
(use-package denote
  :custom
  (denote-sort-keywords t)
  :hook
  (dired-mode . denote-dired-mode)
  :custom-face
  (denote-faces-link ((t (:slant italic))))
  :bind
  (("C-c w d b" . denote-find-backlink)
   ("C-c w d d" . denote-date)
   ("C-c w d f" . denote-find-link)
   ("C-c w d h" . denote-org-extras-link-to-heading)
   ("C-c w d i" . denote-link-or-create)
   ("C-c w d I" . denote-org-extras-dblock-insert-links)
   ("C-c w d k" . denote-keywords-add)
   ("C-c w d K" . denote-keywords-remove)
   ("C-c w d l" . denote-link-find-file)
   ("C-c w d n" . denote)
   ("C-c w d r" . denote-rename-file)
   ("C-c w d R" . denote-rename-file-using-front-matter)))
```

The Consult-Notes package helps to quickly find notes. The default location to find notes is the Denote folder. You can add other locations to the search menu to create a one-stop shop to find any files. If, for example, you keep your photographs in `~/Photos` then add `("Photographs" ?p "~/Photos")` so they become part of the search menu. The letter after the question mark becomes the key to limit the search to this silo, prefixed with a colon. So in this example, starting with `:p` will only show files in the photos directory.

For the search functionality to work you need to install the RipGrep program, an extremely fast program to search through text files.

```
(use-package consult-notes
  :custom
  (consult-narrow-key ":")
  (consult-notes-file-dir-sources
   `(("Denote Notes"  ?n ,denote-directory)))
  :bind
  (("C-c w h" . consult-org-heading)
   ("C-c w f" . consult-notes)
   ("C-c w g" . consult-notes-search-in-all-notes)))

(use-package citar-denote
  :demand t
  :custom
  (citar-open-always-create-notes t)
  :init
  (citar-denote-mode)
  :bind
  (("C-c w b c" . citar-create-note)
   ("C-c w b n" . citar-denote-open-note)
   ("C-c w b x" . citar-denote-nocite)
```

```
   :map org-mode-map
   ("C-c w b k" . citar-denote-add-citekey)
   ("C-c w b K" . citar-denote-remove-citekey)
   ("C-c w b d" . citar-denote-dwim)))

(use-package denote-explore
  :bind
  (;; Statistics
   ("C-c w x c" . denote-explore-count-notes)
   ("C-c w x C" . denote-explore-count-keywords)
   ("C-c w x b" . denote-explore-keywords-barchart)
   ("C-c w x x" . denote-explore-extensions-barchart)
   ;; Random walks
   ("C-c w x r" . denote-explore-random-note)
   ("C-c w x l" . denote-explore-random-link)
   ("C-c w x k" . denote-explore-random-keyword)
   ;; Denote Janitor
   ("C-c w x d" . denote-explore-identify-duplicate-notes)
   ("C-c w x z" . denote-explore-zero-keywords)
   ("C-c w x s" . denote-explore-single-keywords)
   ("C-c w x o" . denote-explore-sort-keywords)
   ("C-c w x r" . denote-explore-rename-keywords)
   ;; Visualise denote
   ("C-c w x n" . denote-explore-network)
   ("C-c w x v" . denote-explore-network-regenerate)
   ("C-c w x D" . denote-explore-degree-barchart)))
```

9.9 Production

9.9.1 Managing the Writing Process

```
(use-package org
  :bind
  (:map org-mode-map
        ("C-c w n" . ews-org-insert-notes-drawer)
        ("C-c w p" . ews-org-insert-screenshot)
        ("C-c w c" . ews-org-count-words)))

(use-package olivetti
  :demand t
  :bind
  (("C-c w o" . ews-olivetti)))

(use-package undo-tree
  :config
  (global-undo-tree-mode)
  :custom
  (undo-tree-auto-save-history nil)
  :bind
  (("C-c w u" . undo-tree-visualize)))
```

9.9.2 Citations

```
(require 'oc-natbib)
(require 'oc-csl)

(setq org-cite-csl-styles-dir ews-bibtex-directory
      org-cite-export-processors
      '((latex natbib "apalike2" "authoryear")
        (t     csl    "apa6.csl")))
```

9.9.3 Quality Assurance

```
(use-package dictionary
  :custom
  (dictionary-server "dict.org")
  :bind
  (("C-c w s d" . dictionary-lookup-definition)))
```

The writegood package helps to detect buzzwords, passive writing and repeated words. This package also contains functions to estimate the complexity of a text.

```
(use-package writegood-mode
  :bind
  (("C-c w s r" . writegood-reading-ease))
  :hook
  (text-mode . writegood-mode))
```

9.9.4 Version Control

The `ediff` family of function by default does not split its windows nicely, so these settings make the program easier to use.

```
(setq ediff-keep-variants nil
      ediff-split-window-function 'split-window-horizontally
      ediff-window-setup-function 'ediff-setup-windows-plain)
```

9.10 Publication

9.10.1 Basic Settings

The timestamp for exporting files is set to the European date format of day month and year. If you publish for American audiences, perhaps you like to modify the `org-export-date-timestamp-format` to `"%B %e %Y"`. The letters each stand for the full name of the month, the day number without leading zero and the year in four digits. See the documentation for the `format-time-string` function for details on how to format dates in other methods.

```
(use-package org
  :custom
  (org-export-with-drawers nil)
  (org-export-with-todo-keywords nil)
```

```
  (org-export-with-broken-links t)
  (org-export-with-toc nil)
  (org-export-with-smart-quotes t)
  (org-export-date-timestamp-format "%e %B %Y"))
```

9.10.2 Office Documents

```
;; Not included in EWS

;; Export ODT to MS-Word
(setq-default org-odt-preferred-output-format "docx")

;; Export ODT to PDF
(setq-default org-odt-preferred-output-format "pdf")
```

9.10.3 Latex

```
;; LaTeX PDF Export settings

(use-package ox-latex
  :ensure nil
  :demand t
  :custom
  ;; Multiple LaTeX passes for bibliographies
  (org-latex-pdf-process
   '("pdflatex -interaction nonstopmode -output-directory %o %f"
     "bibtex %b"
     "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
     "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"))
  ;; Clean temporary files after export
  (org-latex-logfiles-extensions
   (quote ("lof" "lot" "tex~" "aux" "idx" "log" "out"
           "toc" "nav" "snm" "vrb" "dvi" "fdb_latexmk"
           "blg" "brf" "fls" "entoc" "ps" "spl" "bbl"
           "tex" "bcf"))))

(with-eval-after-load 'ox-latex
  (add-to-list
   'org-latex-classes
   '("crc"
     "\\documentclass[krantz2]{krantz}
        \\usepackage{lmodern}
        \\usepackage[authoryear]{natbib}
        \\usepackage{nicefrac}
        \\usepackage[bf,singlelinecheck=off]{caption}
        \\captionsetup[table]{labelsep=space}
        \\captionsetup[figure]{labelsep=space}
        \\usepackage{Alegreya}
        \\usepackage[scale=.8]{sourcecodepro}
        \\usepackage[breaklines=true]{minted}
        \\usepackage{rotating}
```

```
        \\usepackage[notbib, nottoc,notlot,notlof]{tocbibind}
        \\usepackage{amsfonts, tikz, tikz-layers}
        \\usetikzlibrary{fadings, quotes, shapes, calc, decorations.markings}
        \\usetikzlibrary{patterns, shadows.blur}
        \\usetikzlibrary{shapes,shapes.geometric,positioning}
        \\usetikzlibrary{arrows, arrows.meta, backgrounds}
        \\usepackage{imakeidx} \\makeindex[intoc]
        \\renewcommand{\\textfraction}{0.05}
        \\renewcommand{\\topfraction}{0.8}
        \\renewcommand{\\bottomfraction}{0.8}
        \\renewcommand{\\floatpagefraction}{0.75}
        \\renewcommand{\\eqref}[1]{(Equation \\ref{#1})}
        \\renewcommand{\\LaTeX}{LaTeX}"
    ("\\chapter{%s}" . "\\chapter*{%s}")
    ("\\section{%s}" . "\\section*{%s}")
    ("\\subsection{%s}" . "\\subsection*{%s}")
    ("\\subsubsection{%s}" . "\\paragraph*{%s}"))))
```

9.10.4 ePub

```
;; epub export

(use-package ox-epub
  :demand t)
```

9.11 Administration

9.11.1 Getting Things Done

```
(use-package org
 :bind
 (("C-c a" . org-agenda)))
```

9.11.2 Manage Files

The dired package is a convenient and powerful tool to keep your drives organised and access your information. Developers have published an extensive collection of extensions to dired to add functionality, which you can find in the package manager.

Dired lists files and directories in alphabetical order. I prefer a different view, which shows directories on top and files below them. The parameters determine the order of the entries in the folder.

This last bit of configuration code defines how Emacs deals with automated backups. The default setting is that the system stores these files in the folder where the original files lives, clutterin/g your drive with copies of your stuff. The setting below modifies the `backup-directory-alist` variable so that Emacs saves all backups in your configuration folder. This configuration also eliminates lock files, which are only useful when working in shared folders.

Alternatively, you could instruct Emacs to not save backups at all with `(setq-default make-backup-files nil)`. I prefer keeping backups as they have saved my bacon a few times in the past.

```
(use-package dired
  :ensure
  nil
  :commands
  (dired dired-jump)
  :custom
  (dired-listing-switches
   "-goah --group-directories-first --time-style=long-iso")
  (dired-dwim-target t)
  (delete-by-moving-to-trash t)
  :init
  (put 'dired-find-alternate-file 'disabled nil))

(use-package dired-hide-dotfiles
  :hook
  (dired-mode . dired-hide-dotfiles-mode)
  :bind
  (:map dired-mode-map ("." . dired-hide-dotfiles-mode)))

;; Backup files

(setq backup-directory-alist
      `(("." . ,(expand-file-name "backups/" user-emacs-directory)))
      version-control t
      delete-old-versions t
      create-lockfiles nil)  ; No lock files
```

The function to save the recent files runs every five minutes, instead of only when Emacs exists. The function to save the list of recent files needs some modifictaion to prevent messages popping up in the buffer every so often.

```
(use-package recentf
  :config
  (recentf-mode t)
  (run-at-time nil (* 5 60)
               (lambda () (let ((save-silently t))
                            (recentf-save-list))))
  :custom
  (recentf-max-saved-items 50)
  :bind
  (("C-c w r" . recentf-open)))


(use-package bookmark
  :custom
  (bookmark-save-flag 1)
  :bind
  ("C-x r D" . bookmark-delete))
```

9.12 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 `F5` 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.

```
(keymap-global-set "<F5>" #'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.

Bibliography

Ahrens, S. (2017). *How to Take Smart Notes: One Simple Technique to Boost Writing, Learning and Thinking: For Students, Academics and Nonfiction Book Writers*. North Charleston, SC: CreateSpace.

Allen, D. (2005). *Getting Things Done: The Art of Stress-Free Productivity*. London: Piatkus.

Berry, R. (1988). Common User Access. A consistent and usable human-computer interface for the saa environments. *IBM Systems Journal*, 27(3), 281–300.

Blevins, J. R. (2017). *Guide to Markdown Mode for Emacs*. LeanPub.

Bottiroli, S., Rosi, A., Russo, R., Vecchi, T., & Cavallini, E. (2014). The cognitive effects of listening to background music on older adults: processing speed improves with upbeat music, while memory seems to benefit from both upbeat and downbeat music. *Frontiers in Aging Neuroscience*, 6.

Clark, A. & Chalmers, D. (1998). The extended mind. *Analysis*, 58(1), 7–19.

Covey, S. R. (1990). *The Seven Habits of Highly Effective People: Restoring the Character Ethic*. New York: Fireside Book, 1st fireside ed edition.

Even-Ezra, A. (2021). *Lines of Thought: Branching Diagrams and the Medieval Mind*. Chicago: The University of Chicago Press.

Ferriss, T. (2011). *The 4-Hour Work Week. Escape the 9-5, Live Anywhere and Join the New Rich*. London: Vermilion.

Forte, T. (2022). *Building a Second Brain: A Proven Method to Organise Your Digital Life and Unlock Your Creative Potential*. London: Profile Books Ltd.

Johnson, T. (2022). Emacs as a tool for modern science: The use of open source tools to improve scientific workflows. *Johnson Matthey Technology Review*, 66(2), 122–129.

Khalili, A. & Auer, S. (2015). Wysiwym – integrated visualization, exploration and authoring of semantically enriched un-structured content. *Semantic Web*, 6(3), 259–275.

Kim, K., Erickson, A., Lambert, A., Bruder, G., & Welch, G. (2019). Effects of dark mode on visual fatigue and acuity in optical see-through head-mounted displays. In *Symposium on Spatial User Interaction*, SUI '19: ACM.

König, R. (2020). Getting yourself organized with Org-mode. A supplement for the video course. Udemy course, `udemy.com`.

Lamport, L. (1994). *LATEX: A Document Preparation System: User's Guide and Reference Manual*. Reading, Mass: Addison-Wesley Pub. Co, 2nd ed edition.

Landin, P. J. (1964). The mechanical evaluation of expressions. *The Computer Journal*, 6(4), 308–320.

Lipovetsky, S. (2023). Readability indices structure and optimal features. *Axioms*, 12(5), 421.

Monnier, S. & Sperber, M. (2020). Evolution of emacs lisp. *Proceedings of the ACM on Programming Languages*, 4(HOPL), 1–55.

Mueller, P. A. & Oppenheimer, D. M. (2014). The pen is mightier than the keyboard: Advantages of longhand over laptop note taking. *Psychological Science*, 25(6), 1159–1168.

Omanson, R., Miller, C. S., Young, E., & Schwantes, D. (2010). Comparison of mouse and keyboard efficiency. In *Proceedings of the Human Factors and Ergonomics Society* (pp. 600–404).

Petersen, M. (2022). Mastering emacs.

Pohle, J. & Thiel, T. (2020). Digital sovereignty. *Internet Policy Review*, 9(4).

Stallman, R. (2023). *GNU Emacs Manual*. Free Software Foundation.

Stallman, R. M. (1981a). *EMACS Manual for ITS Users*. Technical Report AIM-554, Massachusetts Institute of Technology. Artificial Intelligence Laboratory.

Stallman, R. M. (1981b). EMACS the extensible, customizable self-documenting display editor. *ACM SIGOA Newsletter*, 2(1-2), 147–156.

Stavrou, P. (2024). Re: Advice regarding note-taking in emacs. https://protesilaos.com/.

Tognazzini, B. (1992). *Tog on Interface*. Reading, Mass: Addison-Wesley.

Tognazzini, B. (1993). Principles, techniques, and ethics of stage magic and their application to human interface design. In *Proceedings of the Interact '93 and CHI '93 Conference on Human Factors in Computing Systems* (pp. 355–362).

Tracy, B. (2016). *Eat That Frog! 21 Great Ways to Stop Procrastinating and Get More Done in Less Time*. London: Hodder.

Travis, B. E. & Waldt, D. C. (1995). *Evolution of Publishing Systems*, (pp. 21–36). Springer Berlin Heidelberg.

Umejima, K., Ibaraki, T., Yamazaki, T., & Sakai, K. L. (2021). Paper Notebooks vs. Mobile Devices: Brain Activation Differences During Memory Retrieval. *Frontiers in Behavioral Neuroscience*, 15, 634158.

University of Chicago press, Ed. (2017). *The Chicago Manual of Style*. Chicago: The University of Chicago Press, seventeenth edition edition.

Watson, D. (2004). *Watson's Dictionary of Weasel Words, Contemporary Clichés, Cant & Management Jargon*. Milsons Point, N.S.W: Knopf.