

Getting Started With Vanilla GNU Emacs

Peter Prevos

April 28, 2024

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](https://www.gnu.org/software/emacs/)) 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 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.

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 (??). 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 functionality. Keyboard shortcuts are easy to remember as they become part of your muscle memory. Also, regularly moving your hands between the keyboard and the mouse can be annoying and impede your workflow.

When misspelling a word in a word processor, you move your hand from the keyboard to the mouse, click on the offending word and select the desired

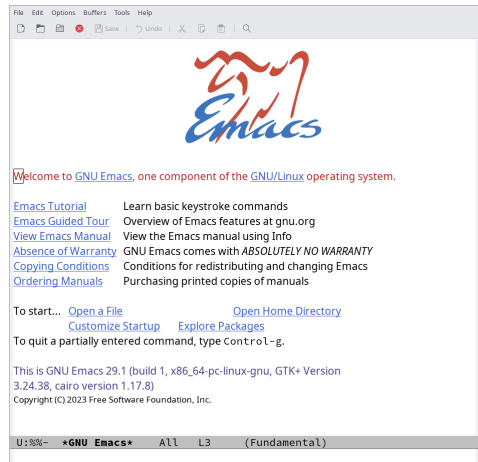


Figure 1: Emacs 29 splash screen.

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 key 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 is 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 shown as `<space>`. Each modifier key has its own letter, as shown in table 1. You can combine modifier keys, occasionally leading to awkward combinations, such as **C-M-S a** (Control, Alt and Shift **a**), which requires the nimble fingers of a sleight-of-hand artist to execute smoothly. The 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 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 will 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 no 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 (?). 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).

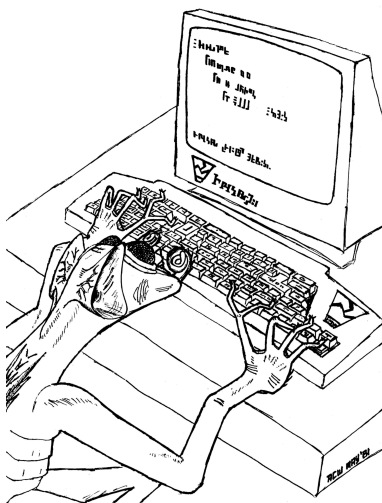


Figure 2: Cover of the 1981 version of the Emacs manual.

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.

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 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.

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. 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 want to 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). 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 your file needs saving. two dashes, means that the content of the file is the same as the buffer.

Table 2: Most commonly used file functions.

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

5 Buffers, Frames and Windows

When you open Emacs, the software runs within a frame (figure 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.

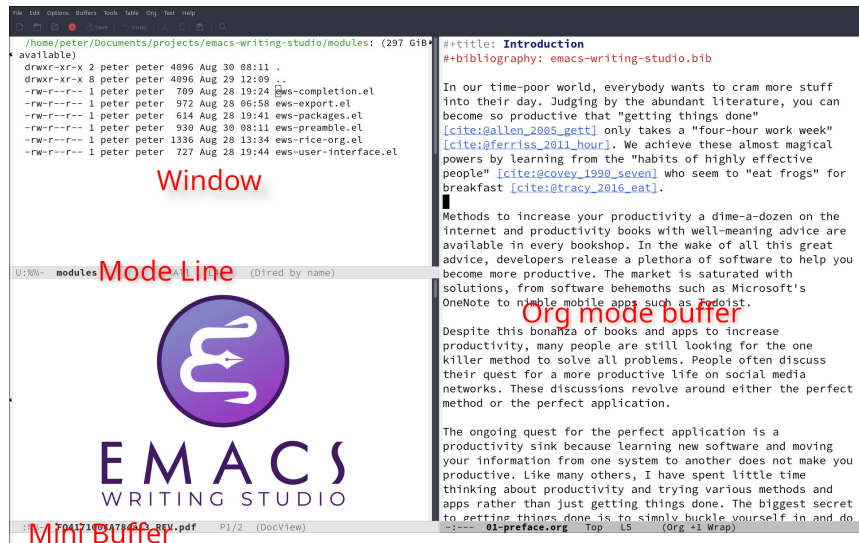


Figure 3: Emacs frame with three windows, a Dired buffer, image buffer and Org mode buffer.

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 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 3: Buffer and window functions.

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

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 its 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.

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`.

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 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 4: Moving around a buffer in Emacs.

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

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.

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 5 to copy and move text from and to the kill ring.

Table 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

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 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 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 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 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.

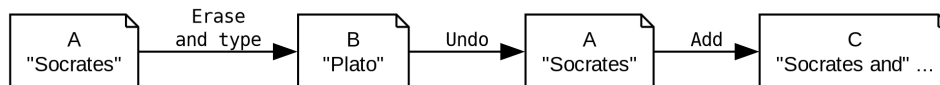


Figure 4: Emacs undo states.

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.

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 Ukrainian 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 pops 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 g input`.

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.

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.

8.1 Emacs Lisp

Commercial software provides graphical menus to define how it operates. For example, in Figure 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.

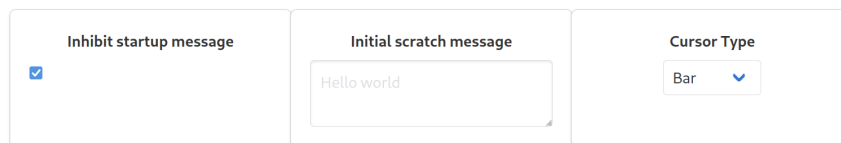


Figure 5: Typical graphical configuration screen.

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 5. Compare the code with the image to reverse engineer the Emacs 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 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)
```

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 shown 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.

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 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, which then looks something like this:

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

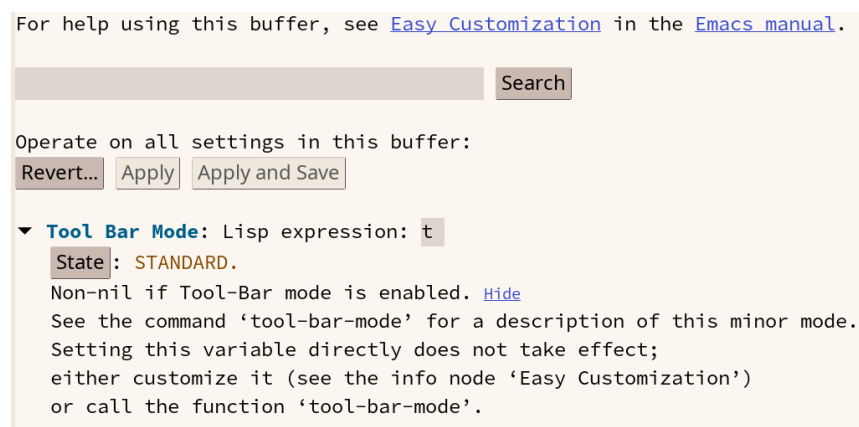


Figure 6: Customisation screen for `tool-bar-mode`.

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.

9 Exiting Emacs

Working with Emacs is so much fun you might never want 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 losing 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.