Emacs Tutorials for Reproducible Research

Scott Jackson

Last updated: July 20, 2013

Contents

1 What, why?

This is a guide for getting started using Emacs as the center of a reproducible research workflow. Emacs is a text editor, but much more than that. My thoughts on why reproducible research is worth the effort are elsewhere. Here's the breakdown of why I think Emacs is so useful for actually *doing* reproducible research.

It's a great editor for simple text. Virtually all research involves writing text. There are many great text editors out there, and many terrible ones. If at all possible, use a great one. Emacs has been around for ages, and it still wins new converts. No piece of software is perfect for every person, so if you never get to liking Emacs, give something else a shot. But Emacs is one of the great ones, and it will greatly benefit you any time you have to edit text. Which is probably pretty often. Also, there is an emphasis on *simple* text. You don't use Emacs to edit Microsoft Word files. Those aren't simple text. Simple text is transparent and is the ultimate in portability. This makes it a cornerstone of reproducible research. Text editors like Emacs are designed to edit simple text, which means they fit perfectly into a reproducible research environment.

It's built on a complete programming environment. At it's core Emacs is not really a text editor. It's a virtual machine that runs a version of Lisp code. Lisp is a very, very powerful programming language. The functionality of Emacs all comes from Lisp code, and you can use Emacs to write Lisp code to add to or modify that functionality. You can be a productive user of Emacs without learning anything about Lisp. If you learn some Lisp, then you gain not only the ability to run Lisp programs from inside Emacs (since it's a Lisp-running machine), but also to alter Emacs to work the way you want it to work. It also means that there is a huge community of people using Emacs and writing code to add more functionality to Emacs, and you can take advantage of all that, as well, again without needing to know a thing about Lisp. In terms of reproducible research, this means that you have both the flexibility of customizing a set-up that enables you to get your work done, but it also enables the sharing and reproduction of that customization.

It's free, open source, and cross-platform. Emacs was initially written by one of the most vocal and iconic figures in the open source software movement, Richard Stallman. It's a cornerstone of the GNU open source software library. This means all the code is open and transparent, and can be freely inspected, modified, and shared by all users. It can also be used on all major operating systems, including Windows, Mac, and Linux. And it works virtually the same on all these platforms. All of these features make Emacs much more amenable to the goals of reproducible research.

It integrates with other important tools for reproducible research. I was initially drawn to Emacs because at the time (before RStudio), it was the one editor I could find that was both a good editor for the R statistical platform and the LaTeX document typesetting system. You can run R from within Emacs. You

can write and compile LaTeX documents from within Emacs. You can run shell commands from within Emacs. You can interact with version control software (like git) from within Emacs. It's like a command center for many of your common research tasks. You can even get Emacs to handle email and web browsing, though personally I don't go that far. But once I experienced the joy of being able to run an analysis in R, combine that with LaTeX, and compile that to a PDF, all without leaving Emacs, I was hooked. Instead of hopping from tool to tool, I could do everything I needed from within Emacs.

Org-mode. Emacs is an incredibly powerful platform, but `org-mode` is, in my opinion, the "killer app," especially when it comes to reproducible research.[1] `Org-mode` is a package of functionality within Emacs, and it's like an ultimate Swiss Army knife for reproducible research. It started as a system for taking notes and managing a to-do list, and it has grown from there into a complete system for creating structured documents, integrating code and text, which can be exported to HTML, LaTeX, Open Document text, and many other formats. You can write a complete academic paper, including statistical analysis, plotting, cross-references, citations, and bibliographies, all inside `org-mode`. And it accomplishes all this with a simple, easy-to-learn and easy-to-read mark-up system. This makes `org-mode` (and by extension, Emacs) arguably the single most powerful, flexible system for carrying out reproducible research, ever.

My goal is to spend the rest of this document helping you get to a point where you can see why I think these things, and helping you make a decision for yourself about whether Emacs fits the bill for you, and if so, how to make it work for you.

2 More about what you're getting yourself into

You might say to yourself, "okay, all that sounds pretty good, but what it is going to cost me? What's the catch?" And you'd be pretty smart to ask that. So let me try to give you a better idea of what the investment is.

The easy one is price. As I mentioned before, Emacs is free and open source. So the only thing it will cost you is time and effort. But those two things are pretty valuable!

I'll be upfront: you will likely *never* learn everything about Emacs. It's too big, too complex, and there's just too much constant development that you will ever be able to (accurately) say, "yeah, I know everything there is to know about Emacs." But once you realize this, it can take a big burden off. What you *need* to learn is very little. Beyond that, you make decisions about what kinds of things you want to be able to do, or to do better, and you learn about how Emacs can (or can't) help you. If you take this approach, you will get less overwhelmed with all the features of Emacs, and you will end up tailoring your Emacs use to your preferred ways of working, and that's really the whole point: to get Emacs to work for you. It can still be useful to try out new things you'd never considered, because there may be a better way to do

[1] Even if it does have a bizarre logo...

things, but Emacs will rarely force you into one way of doing things. Even for many of the "basic" things, if you take the time to learn Lisp, you can change whatever you want about Emacs, but chances are also good that someone else has written some Lisp that does something pretty close to what you want.

So what about this stuff you *need* to learn? The initial learning curve boils down to learning how to do basic things in Emacs, which (probably) work differently from how you do those things in other programs you're used to. This is the part that feels like work, and can feel frustrating. My goal in this set of tutorials is to get you through these parts with a minimum of pain, by taking a task-based approach. The time it takes you to get through this stage is a combination of how long it takes you to learn it initially (which is not long at all), and how long it takes you before it becomes second nature. I can't predict how long that will take, but the more you use Emacs, the quicker it will be, and I can report from my personal experience that it took a lot less time than I expected. By the time you get through the tutorial on editing a simple text file in section 4, you will have gotten through the toughest parts. Up to that point, you may feel like you're having to re-learn how to do basic things that you can easily do in a wide range of other programs. But after that point, you will be learning things that make Emacs more powerful, and more pleasant to work with, than mosty common editors. This will hopefully get you over the initial hump faster than it took me the first time.

Here's how the tutorials are designed. Each tutorial is structured around some task or set of tasks, and in three parts: *walkthrough*, *exercises*, and *reference*. The walkthrough is designed to be a follow-along kind of task, for the first couple of times you do a task. My walkthroughs have lots of narrative description and details. The hope is that they will be easy to read and follow, and they will get you through all the nitty-gritty. The exercises are designed to give you more practice with some things, to try out the tasks with a little less hand-holding. I will also embed cool new things in the exercises, so don't skip over them. Hopefully going through the exercises will go a long way towards helping you retain what you learn. Finally, remembering every detail is impossible, and there may be tasks that you do kind of infrequently, and you may require a refresher of some kind. There are many tasks I do where I still have to refer to a manual or guide somewhere to remind myself of some detail or another. The walkthroughs may not be great for this kind of quick reminder, so I also provide a more succinct reference summary, as something that may be easier to refer back to later, if you just need to jog your memory. The overarching goal is that with each tutorial you complete, you will learn something that is immediately useful. I've ordered the tutorials roughly in order of complexity, but I encourage you to pick-and-choose as you will. If there is some kind of "prereq," such that you should really learn X before you try to take on Y, I will make sure to point that out early on in the tutorial.

3 Installing Emacs

3.1 Overview

Installing Emacs is typically pretty easy, but depending on your operating system (OS), it may be a little different from what you're used to. This tutorial is structured a bit differently than the others. In this tutorial, I give you three walkthroughs: one for Windows, one for Mac, and one for Linux. There are no "exercises," because, well, installation is not something you really need to practice if you can get it right once. There are plenty of other places to get instructions on installation, so my goal here is to present it from my perspective, which may be a little more like your own, compared to the "computer geek" perspective that many Emacs guides are written for.

I will also cover installation of a couple of major packages (Emacs Speaks Statistics, AUCTeX, and org-mode), as well as how to "install" other bits of Lisp code you might find on the web.

3.2 Windows Walkthrough

3.2.1 Preamble

Before going into how to get Emacs on Windows, I want to explain a bit about how Emacs is structured, and about how typical Windows programs are installed. You may know all this already, but this little orientation would have been helpful to me when I first installed Emacs, so it might be helpful to you. First, go into your file system in Windows, by opening "Computer" or "My Computer," and then open the main hard drive (usually the `C:` drive) and go to "Program Files." Find a folder of a program you use (like Microsoft Office or something), and just browse around in there. There are likely a whole bunch of files you typically never have to look at, and if you look around enough, you'll find something that has an `.exe` file extension. If you change your folder view to the "Details" view, so that you have a column called "Type," this kind of file will be called something like "Application." These `.exe` files are also called *executables*, which is where the "exe" comes from. Executables are programs that Windows, well, executes. So if you double-click on one, it will not (usually) open the file up in some kind of editor; it will just run that program. So if you find the executable for Word or Excel, it will start Word or Excel when you double-click it.

But what about all those other files? Those are all ancillary files of some kind or another. Some you are able to open up and modify, but most are just part of the system of software that runs the program. If you went through and started deleting lots of the files in the Microsoft Office folder, you might find that Word or other Office programs would not work correctly afterwards.

What happens when you install a typical Windows program is that you might download something from the internet, or run an installation CD or something, and there is often a "wizard" that takes you through the installation steps, which usually means you click through a bunch of "Next" buttons until it installs. During the installation process, the installer (which is really just a small program bundled up

with the larger program) does a lot of work unpacking the various files and putting them in the right places. This also usually means that it ends up in the Program Files folder, and it usually puts a link in the Start menu and maybe a shortcut on the desktop. So normally, after installation, you interact with one of the many different kinds of shortcuts (on the Start menu, on the desktop, in the Quick Launch toolbar, etc.), instead of browsing into the Program Files folder. But these typical ways of launching a program are all essentially shortcuts to run the main executable (`.exe`) file, which may in turn make use of the various other files that end up in the Program Files folders.

So what's different about Emacs? First, remember that Emacs is open-source. Executable files are called *binary* files, because they are encoded in a binary computer language, which is virtually unreadable to humans. This means that try as you might, you can't "read" an executable to figure out what it does. A human didn't write the binary code, so humans typically can't understand it, either. A human did write the *source* code, but most commercial programs like Word don't make their source code available. Open-source programs like Emacs do. But on the other hand, having the human-readable source code doesn't necessarily mean that the code will *run* on your computer. This is what Windows executables are for. When you take some source code, *compile* it (if necessary) so that it is machine-readable and efficient, and then package it in a way that you can double-click it (or some other interaction like that) and have it run, that's an executable.

Why am I telling you all this? This is all to explain what you need for Emacs to run on Windows. You can download the Emacs source from a lot of different places. But for it to run, you would need to *build* it from source, which essentially boils down to making an executable that will run all that source code. This is possible to do on Windows, but it is not for the faint-hearted, and frankly, it's not something I would ever want to do, given the choice.

Fortunately, we do have a choice! The kind souls that make Emacs available to everyone also release versions of Emacs that contain pre-compiled Windows binaries (i.e., executables). So once you download one of these versions, you can just double-click the executable, and Emacs runs! The main difference is that there is no "wizard" that asks you to click "Next" a bunch of times, and Emacs will not automatically appear in your Start menu or anything. So now with this preamble out of the way, let's do this step by step.

3.2.2 Windows installation in steps

- Decide where you want to install Emacs
  It does not have to go in Program Files. It can go anywhere! You could "install" it on the desktop, at the top of the `C:` drive directory, or in some folder buried deep inside your Documents folder, if you wanted. Personally, my Emacs at home is sitting in my User home folder. To be specific, I have a folder called `C:\Users\Scott` and the Emacs installation is sitting there. This works well for me, but the other users of that computer (my wife and kids) have no need to use Emacs. Otherwise I would need to give them access to my home folder for them to run it. So depending on where you want it to run, you just make a

decision. You can always move it later, if you realize there's a better place for it.

- Download the most recent Windows release
  This is easy. If you go to this address: http://ftp.gnu.org/gnu/emacs/ you will see a bunch of files, most of them with `.tar` in the name. These are all bundled-up source code. Towards the top of the page, you should see a folder icon called "windows." Clicking that should send you here: http://ftp.gnu.org/gnu/emacs/windows/. This new page has a bunch of instructions at the top, and father down, download links to the Windows binaries.

  Which one do you get? You typically should just get the most recent version, because only stable versions are released as Windows binaries; if you want the bleeding edge "development" version of Emacs, you are going to have to learn to obtain that and build it from source on your own. At the time I'm writing this, the most recent stable release is 24.3. But if you see a 24.4 or 25 or whatever, go for that. And you want the file that ends in `.zip`, not `.zip.sig`. So at the time of writing, I see a file called `emacs-24.3-bin-i386.zip`, and that's what I would get. The `bin` tells you it's got binaries in it, and the `i386` tells you that it's built for the i386 computing architecture, which is the common 32-bit Windows architecture.[2] So click the link and download it! The version I see has a 47 MB download.

- Unzip the files
  After your download is complete, you need to unzip the files. You can use any standard unzipping program. Windows usually has at least one built-in. I also like the 7-zip program, which is also free and open source, and a little more versatile than the standard Windows unzipper.

  When you unzip the files, you should end up with a folder called `emacs-<version>` (e.g., `emacs-24.3`, if you downloaded the one I mention above). That's it, Emacs is "installed"! If you didn't unzip the files into the folder where you want Emacs to sit, then just move that whole `emacs-<version>` folder to your target folder.

- Look around
  You should now go into the Emacs folder and browse around a bit. Mostly you should see lots of `.el` and `.elc` files. The `.el` files are Emacs Lisp (also called ELisp) files. You may notice a folder called `elisp` and another called `lisp`. All of the ELisp files are human-readable (that is, if you know Lisp). The `.elc` files are *compiled* ELisp files. For every `.elc` file, there should be a corresponding `.el` source file. The compiled files make Emacs run faster, but they aren't strictly necessary. You may also see some files in C with different

[2] Technically, `i386` refers specifically to Intel processors, but this kind of architecture is essentially the common denominator for any Windows-running PC, and this Emacs version runs perfectly well with my computer, which has a 64-bit AMD processor. However, these binaries may or may not run on ARM processors, which are common in mobile devices like tablets, including the Microsoft Surface RT. Maybe Emacs runs on those, but I haven't tried or researched it much. Running Emacs on a tablet doesn't sound very appealing, anyway.

extensions. Emacs is *mostly* written in Lisp, but it does have some C code for running some very basic operations. You may also see other kinds of documentation and auxiliary files.

- Find the `.exe` and run it
  Now you should locate a folder called `bin` (for *binary*), and inside that, you should see a couple of files that have the little purple Emacs icon. The one called `runemacs` is what you want. Double-click it, and Emacs should start! You should see a "splash screen" that looks something like this:

  Congratulations, you have a working Emacs installation!

- Make a shortcut
  Now you probably don't want to have to go through these folders to run Emacs every time. If you right-click on the `runemacs` executable, and select "Create shortcut", it will make a shortcut to the file. You can then put this wherever you want: in the task bar, on the desktop, in the Quick Launch bar, wherever. If you want it in multiple places, just copy the shortcut at will.

- Uninstalling
  Since "installation" just means unzipping the files and putting them somewhere, "uninstalling" just means deleting those files! That's it.

- Recap and reference
  In the end, the installation process is extremely simple. All you do is:

  1. Download the Windows binary from http://ftp.gnu.org/gnu/emacs/windows/
  2. Unzip the files (moving the resulting folder if needed to wherever you want it)
  3. Run the `runemacs` binary when you want to start Emacs (making shortcuts for convenience)
  4. Delete the Emacs files as desired if you ever want to "uninstall" it.

  Easy!

3.3 Linux Walkthrough

3.3.1 Caveats

I am personally just dipping my toes into Linux. I like it, but I'm a definite noob. This means the advice here appears to work for me, but it may be wrong in some way, and I am probably missing some big points. I will update this section as my Linux knowledge develops, but I'm also happy to get feedback from Linux veterans (or other noobs with different experiences) so that I can improve this section.

3.3.2 Done?

Because Emacs is part of the GNU ecosystem, there is a good chance you already have a version of it on your machine. If you can run `emacs` in the terminal and it starts Emacs, then you can see what version you have installed. Personally, I think there have been enough useful changes in Emacs version 24.x that if you have version 23.x or older, it would be worth getting a newer installation.

3.3.3 `apt-get`

On Ubuntu and related distributions, I think using `apt-get` is the easiest way to get an up-to-date version. In my experience, the graphical software center doesn't always have the most recent version. With `apt-get`, it works pretty much the same as with any other installation. At the time I'm writing this, in Ubuntu or other Devian-derived distributions, the following will get you version 24.2:

```
sudo apt-get install emacs24
```

Now you should be able to start Emacs by running `emacs` in the terminal, or by starting it from a graphical shortcut or whatever other application launcher you like to use. Easy!

Similarly, you can use the typical `apt-get remove` commands to get rid of it if you no longer want it (or if you want to remove it and get a newer version).

3.4 Mac Walkthrough

I don't have a Mac, but it looks like the following site (maintained by David Caldwell) has Mac binaries all ready to go, for the most recent stable release: http://emacsformacosx.com/

You can also go directly to the GNU Emacs site and follow instructions from there. The only two cents I have to add is that not too long ago, a distribution called Aquamacs was popular among the OS X crowd, but currently, the GNU Emacs distribution is the most recommended.

As I get feedback from Mac owners, I'll update this section.

3.5 Installing org-mode

3.5.1 Done?

In the intro, I sang the praises of `org-mode`, which is a special mode in Emacs. The good news is that virtually all recent releases of Emacs come with `org-mode` already bundled. However, you may want a more recent version, because `org-mode` development is pretty constant, and releases faster than new version of Emacs.

Here's my recommendation. If you get a recent version of Emacs (24.x), the version of `org-mode` that comes with it is great and has all the features I'll discuss here. This is what I do. If you are stuck with an older version of Emacs for some reason, you should try to update. The caveat is that the following tutorials should work with version 7.x of `org-mode`, but in particular, some of the exporting may change with the newer 8.x versions. I will catch up on this at some point, but apparently the 8.x release made a big overhaul of the exporting functionality.

4 Editing a single simple file

4.1 Overview

Editing text is the primary function of Emacs. It is a very powerful text editor. However, Emacs does things a little differently, so when you first start, it can feel very alien. The goal of this tutorial is to get you to the point where you are comfortable opening and modifying files with Emacs. I will only cover the very basics in terms of editing, but hopefully, by the end of this, you will start to see how editing text in Emacs can be a lot more pleasant and productive than with other editors.

4.2 Walkthrough

4.2.1 Emacs commands

Start Emacs. If you need to install it, check out my tutorial on that.

You will see a pretty ugly splash screen. Don't worry, in the next tutorial I will show you how to get rid of that. But we're not here to look at a stylized picture of a gnu; we're here to edit text. But before we can do that, I need to introduce a concept: *Emacs commands*. Emacs displays characters as you type it on your keyboard, as you might expect. But aside from basic characters, you do stuff in Emacs by running commands. Commands (usually) use special keys. The two most important keys are the *Control* key and the *Meta* key. This is Emacs terminology. On most keyboards, the Control key is the `Ctrl` key (or `control` on Mac), and the Meta key is the `Alt` key (or `alt/option` on Mac). Some hard-core text-editing nerd... um, *enthusiasts* like to complain about "Emacs pinky," referring to how frequently the left pinky finger is used to hold down the Control key while typing. People who type enough that typing ergonomics are a very serious matter may want to consider re-mapping more convenient keys, like the otherwise useless Caps Lock key, to be an alternative Control key. After trying Emacs for a little while, you should feel free to modify your key mappings to better suit your particular keyboard layout and ergonomic preferences. Exactly how to do this will vary by system. I will stick with the standard Emacs jargon of Control and Meta.

Emacs documentation uses a common notation system for how commands are typed. Here's how it works. Don't try any commands yet, we'll get to that shortly!

- A capital `C` means the Control key, a capital `M` means Meta, and a capital `S` means the Shift key.
- Combinations of simultaneous presses are indicated with a dash. So `C-g` means "hold down the Control key and press g."
- Some combination may involve more than two keys pressed at once. For example, on most American keyboards, `M-%` requires you to press and hold the Meta key and Shift, and then hit 5. At least, on a keyboard where `Shift-5` is a `%` sign. This is not notated as `M-S-5`, precisely because for some keyboards, `M-S-5` is not the same as `M-%`, though it is for typical American keyboards. The point here is that when learning new commands, pay attention to the

dashes, since they tell you which key presses need to be simultaneous, and it may be more than two at a time.

- Some commands use sequences of combinations. For example, `C-x C-f` means "hold down the Control key and press `x`, then hold down the Control key and press `f`." You could just keep holding the Control key the whole time while hitting `x` and then `f`, or you could lift off in the middle. Once you do this a few times, you'll get the hang of it.

- Finally, some commands start with some special keys, but then move to regular keys. For example, `C-x u` means "hold down Control while you hit `x`, then release Control and hit `u` like normal." Similarly, `M-x version RET` means "hold the Meta key while you hit `x`, then type the word `version`, and end by hitting the Return key (which is labeled `Enter` on many keyboards)."

- Don't worry, this is not as complicated as it sounds. Once you do a few of these, you'll get the hang of it very quickly.

4.2.2 First command: opening or "finding" a file

So let's put this into action. We're tired of that splash screen, and want to edit something! The basic way to open a file is the command `C-x C-f`. Again, this means you hold down the Control key (called `Ctrl` on most PC keyboards), and while holding it, you hit `x` and then `f` (you can lift off of the Control key between, it just needs to be pressed while you hit `x`, and then again while you hit `f`). Try it!

4.2.3 Emacs talks to you in the minibuffer

Now look down at the bottom of the Emacs window. There is a little space at the bottom, and it should say "Find file:" and then the beginning of a file location. If you didn't hit the command quite right, don't worry, just hold tight for the next section. This little space at the bottom of the Emacs window is called the *minibuffer*. This is a special little area in Emacs where Emacs will often communicate with you.

4.2.4 Slam on the brakes!

Now hit `C-g` (hold Control and hit g). The minibuffer message will now read "Quit", and Emacs may beep at you. This command may be the most useful command of all when you are first learning Emacs. This command puts a stop to anything that Emacs is doing. This is great when you start to enter a command and get mixed up, or just change your mind. Hit `C-g` anytime you think "wait, hold on, that's not what I want to do!" If you ever get stuck in a complex command and just want to bail out, hit `C-g`. It's your friend.

Now, start the "find file" command again (`C-x C-f`), and hit `C-g` again to quit it. Do that a few times until it starts to feel natural.

4.2.5 Opening a file for real

Alright, enough of that quitting, time to actually open a file. Hit the command again (do you remember it yet?) and instead of quitting, type `firstfile.txt` and hit Return. Don't worry about the rest of the file location that may or may not be displaying in the minibuffer. Just type `firstfile.txt` at the end of whatever's already there and hit Return. If you goof up, don't worry, you've got your old friend `C-g` to bail you out if you type something in the minibuffer that you didn't mean to.

When you execute this command successfully, the Emacs window will go blank, the cursor (the flashing box) will be at the top left of the window, and at the bottom, on the line just above the minibuffer, you should see `firstfile.txt`, or whatever it was you actually typed. Assuming this is not a file that already exists, the minibuffer will also tell you `(New file)`. We expect this now, but if you were trying to open up a file that should already exist and still got the `(New file)` message in the minibuffer, it would be a good hint that you didn't actually find the file.

4.2.6 What did I just do?

If you think about what just happened, it may strike you as slightly odd. We ran a command called "find file", but we gave it the name of a file that doesn't actually exist (yet). So what exactly are we editing now? It turns out that this command will open a file that already exists, but if it's given a file name that does not already exist in this location, it will start a new buffer anyway.

A new what?

When you type in Emacs, you are editing text in a *buffer*, which is like a temporary holding-place. You are not actually changing anything in a file. When you tell Emacs to save or "write" a file, then it changes a file's contents. But when you are just typing in the Emacs window, it's just text that Emacs is holding onto as a temporary buffer until you save it.

When you "find" a new file that doesn't exist yet, like what we just did, Emacs will only bring it into existence if you then save this new buffer to disk.

4.2.7 A little typing

But before we save, let's type some content. You could go with "Hello, World!" if you're feeling traditional, or something else. Type something in the `firstfile.txt` buffer now.

4.2.8 Flagging changes

Now look down to that line above the minibuffer again. It's called the *mode line*. The mode line has a lot of information in it, but we'll get to that gradually. For now, notice that to the left of `firstfile.txt`, instead of a bunch of dashes, there are a couple of asterisks (`*`). These indicate that the buffer has been modified, and the changes have not been saved. In other words, these are like a little flag reminding you "unsaved changes!" Handy.

4.2.9 Saving

Since you don't want to lose the brilliant text you typed into the buffer, let's save it. The basic command for saving any changes to a file is:

`C-x C-s`

When you do this, look down at the minibuffer again, and notice that it confirms that it wrote the file, and it gives you the full directory path. This is a nice reassurance that it did what you wanted it to do.

Now, try `C-x C-s` again, before making any changes, and notice that the minibuffer lets you know that no changes needed to be saved.

At this point, we've covered the most basic Emacs workflow: use `C-x C-f` to open a file (or create a new one), edit the buffer, and use `C-x C-s` to save changes we made in the buffer to a file.

4.2.10 Backups and autosave

Emacs is generally very stable. I'm not sure I've had Emacs crash on me, or at least it happens very infrequently. But I have had *Windows* crash on me (or power go out, etc.) while working in Emacs (not because of Emacs!). The point is that sometimes, even if you are good about saving your work, thigns can go wrong. Emacs does two things to help you not lose your work. When you save new changes in a file, Emacs automatically creates a "backup" file. This is a file in the same folder, with the exact same name as the file you saved, except that the file name ends in a tilde (~) sign. This is a little safety net against unintended changes. If you save changes again, this file is not updated, until you restart Emacs and save more changes. In other words, the backup file is only updated when you first save changes after starting a new session of Emacs.

If you use Emacs a lot, you may start to accumulate many of these backup files. If they are cluttering things up for you, you should feel free to delete them. But I generally leave them alone and just ignore them. There will only ever be one backup file for a given file, so they don't just keep piling up if you're just editing one file.

If you have a buffer open for a while without saving changes, you may occasionally see the message `Auto saving...done` in the minibuffer. This is Emacs giving you a little insurance in case of a crash or unexpected ending to the Emacs session. When Emacs auto-saves a file, it creates a new file, a little like the backup file, except the auto-save file is the original filename surrounded by # signs. So for `firstfile.txt`, the backup file will be called `firstfile.txt~`, and the auto-save file will be called `#firstfile.txt#`. After a crash, if you start Emacs and then open a file that has an auto-save file, Emacs will tell you this in the minibuffer, and remind you how to recover the changes that are in the auto-save.

A later tutorial will walk you through the auto-save recovery process. The thing to note here is that when you successfully save your file, Emacs cleans up (deletes) the auto-save file automatically, so you don't have to worry about any extra housekeeping.

4.2.11 Closing Emacs

So what about when we're done with Emacs? You can close Emacs by clicking the little "x" in the upper right of the window (or however your operating system does it), and if all of your buffers are saved, it will close quietly. If you have unsaved buffers, Emacs will alert you in the minibuffer and give you some choices to make. This is nice, because it actually makes it kind of hard to close down Emacs without saving your work.

But of course, there's also a command to quit Emacs:

```
C-x C-c
```

Quitting this way also gives you the same alerts about unsaved buffers. It's just a way to do it without having to use the mouse. Try both of these techniques for quitting Emacs.

4.2.12 Moving around on a line

Now let's work through some of the basic editing techniques. The first thing is to know how to move around in a buffer. This may sound a little strange if you're used to editing in programs like Word, but once you learn some basic commands for moving around quickly in a buffer, you will really enjoy them. Emacs comes with a tutorial that takes you through a lot of the details of navigation and other things. It may be worth going through at some point. You can get to it either by clicking on the link that's displayed on the initial splash screen (the one that reads: "Emacs Tutorial"), or by using the command:

```
C-h t
```

I'll cover a lot of the same material here, in a slightly different way.

If you don't have a buffer open, go through the steps to start Emacs and open up your `firstfile.txt` (or some other file, if you prefer).[3] Now, a couple of concepts. There is a flashing box that moves as you type. This is called the *cursor*. You can think about it as showing you where the next character that you type will appear. But there's another invisible line called the *point*, and you can imagine it as the left edge of the cursor. This is what Emacs keeps track of internally for keeping track of the position in the buffer.

If you haven't already, type enough in your buffer so that it covers most of the line. Type some actual words with spaces, not just a long sequence of characters.

Now, you can click with the mouse to move the point/cursor, but ideally, the less you have to touch the mouse, the more efficient you will be. You can hit Backspace, which works like you might expect, deleting characters along the way. But many times you don't want to delete everything, just move back to a place where you can change something specific. Here are the basic commands:

- `C-f` : move f orward one character

[3] In brief: (l) start Emacs, (2) use the `C-x C-f` key sequence to start the "find file" command, and (3) type `firstfile.txt` and hit Return.

- `C-b` : move b ackward one character
- `M-f` : move forward one *unit*
- `M-b` : move backward one *unit*
- `C-a` : move to the beginning of the line
- `C-e` : move to the end of the line

Try each of these out. Notice that a "unit" basically means "word." I say "unit" because in different modes (i.e., when you're editing different kinds of text/files) the unit may be slightly different. The beauty of this is that while `C-f` and `C-b` move character by character, you can use the `M-` versions to move more quickly, in *meaningful* chunks, like words.

Now, a confession. I very rarely use `C-f` and `C-b`. If I want to back up a couple of characters, like for a brief typo, I still automatically use Backspace, even if it means I end up retyping several characters that I had typed correctly the first time. If I have to move back farther than that, I'll use the `M-` version, or just reach over to mash the arrow keys a bunch or get the mouse. But I'm also not a good touch-typist, so these choices really don't affect my speed much. But this is kind of the point about Emacs: there are often many ways to do something, and you should feel free to do it the way you find works best for you. There's no shame in skipping `C-f` or any other particular command.

On the other hand, I find `C-a` and `C-e` so useful, especially when working on code like R, that I find myself wishing I had these commands in lots of other contexts. When you're first learning these, a simple mnemonic is that "a" is the first letter of the alphabet, so `C-a` goes to the beginning, and "e" in `C-e` stands for "end" of the line. But if you're like me, you will find these so handy that they will quickly become muscle memory.

And so it goes: some aspects of Emacs will not end up being very helpful, and others you will find absolutely indespensible. My attitude is that Emacs has been constructed over many years by people who edit lots of complex text for a living (programmers), so I should at least try to do thing "the Emacs way" at first. But ultimately, "the Emacs way" is really just "your way," because it's all about customization.

Now that you've suffered through this little side-track, go back and practice moving around the line a little more with these commands.

4.2.13 Moving from line to line

4.2.14 Paging around

4.2.15 From top to bottom

4.2.16 Kill/yank

4.2.17 Regions

4.2.18 Undo

4.2.19 Formatting?

4.3 Exercises

4.3.1 Exercise: open, edit, abort a close, save, quit

4.3.2 Open a file with another program at the same time

4.3.3 Open and edit a pre-existing file

4.3.4 Open a file with drag and drop

4.4 Reference

- open a file
- edit
- kill/yank
- undo
- quit-command
- save
- close

5 Making yourself at home in Emacs

- .emacs
- fonts, size
- customize colors
- start-up file

6 Edit multiple files and buffers

- windows & frames

7 Let Emacs type for you

- dynamic completion

8 Manage files with Emacs

- dired

9 Check spelling

- spell check

10 Comparing files

- ediff

11 Recovering a file with auto-save

For example, let's say you are working on a paper called `mypaper.txt`, and you write a Section 3, but then you lose power or something before you remember to save. When you next start up Emacs and then open `mypaper.txt`, Emacs will give you this message in the minibuffer:

```
mypaper.txt has auto save data; consider M-x recover-this-file
```

If you follow this advice and run that command (which would be to hold Meta and hit `x`, then release Meta and type `recover-this-file` and hit Enter), then Emacs will ask:

```
Recover auto save file c:/Users/yourname/#mypaper.txt#? (yes or no)
```

You reply by typing yes or no (and hitting Enter). If you reply "yes," then Emacs will save the contents of `#mypaper.txt#`

12 Miscellaneous editing tips

- transpose
- Change case
- cua-mode

13 Running system commands in Emacs

- one-off
- shell

14 Running R in Emacs

15 Making a TODO list in org-mode

16 Writing an outline in org-mode

17 Adding links to your org-mode notes

18 Making tables in org-mode

19 Embedding R code in org-mode

20 Using data from org-mode tables in R

21 Weaving python and R in org-mode

22 Complete organizational system in org-mode

23 Managing a large project in org-mode

24 Reproducible literature review in org-mode

25 Writing an academic paper in org-mode