A Code Mage’s First Spell Book

Robert S. Muhlestein (rwxrob)

Version v0.0.1, 2024-11-19: First draft

Table of Contents

Copyright

Copies of this book are available in the following formats:

- HTML: https://rwxrob.github.io/code-mage-book
- PDF: https://rwxrob.github.io/code-mage-book/code-mage-book.pdf
- EPUB: https://rwxrob.github.io/code-mage-book/code-mage-book.epub

Dedication

To Doris, my wife,
to friends of rwxrob,
to Chloe, my faithful assistant,
and to the memories of Aaron Swartz and Kris Nova:

Thank you for your light, support, and inspiration.
This book would never have happened without you.

Preface

Who should read this book

This book is written for anyone wanting to learn—or help someone learn—practical computer science and programming for real applications development. While we have fun with the whole fantasy magic theme, everything in this book is very real and might even get you a job someday. It is true that this book can be reliably used as a text book for absolute beginners with enough algebra to understand what a mathematical function is but this book very clearly includes everyone ***outside*** of educational institutions. In fact, I wrote it mostly with pro-active parents in mind who want to give their kids a head-start in tech that they simply cannot find in most traditional education systems, the same reason, in fact, that I started SKILSTAK Coding Arts in 2013 with my own retirement money.

What's included in the book

This book is focused on learning—as fast as possible—to code in the Go programming language from the bash terminal command line. There is nothing more authentic or rewarding than creating command line tools from the command line itself. Therefore, only those minimal skills required to use the command line to open, edit, and run Go code from the terminal are covered.

The fastest way to get a bash terminal up and running is to install bash on a Mac or Windows computer along with a good GPU-accelerated terminal like Alacritty and configure the Vim editor that is already on the system. There is no need for other regularly recommended tools like Neovim or Linux, for now. Hopefully, mages will continue to master other command line powers later and learn the dark arts of Neovim, Linux distro selection, and hardware installation (perhaps covered in additional "tomes" of magic from this author).

How to read this book

This book is designed to be read linearly, from top to bottom. The concepts build on one another. Each "spell" is designed to be repeated often as a means of mastery, like playing chords on a piano or scales on a guitar. The fantasy magic element is intentionally silly and serves as a source of dopamine as well as a strong mnemonic device related to the concepts learn.

> Members of my community have contacted me years later to thank me for creating silly, memorable associations with mini projects focused on one or two core topics allowing them to easily recall them later by remembering the silly part.

Digital format is preferred. It goes without saying, but having the ability to cut and paste from the content of this book goes a long way to simplifying its usage. Besides, doing so is a core tech skill for any code mage today—particularly in the age of powerful AI-assisted code generators. Much of this book was created that way. The latest digital copy of the book is always available for free at https://rwxrob.github.io/code-mage-book (but your support by buying copies or sponsoring at https://github.com/sponsors/rwxrob is always appreciated).

> Повторение — мать учения.

Repetition really is the mother of learning. While often we demonize "rote repetition" in the West, the slavs dominate as programming masters. The neurons in our brain store memory by having the same electical impulses repeated over and over. So in order to program oneself the only way is to repeat it. Repeat these spells as fast and as regularly as you can and you'll end up training your own neural net just like a machine learning model—but in your head. Repetition can be great to supplement your growth while working on a bigger project—especially in the early days before everything becomes second nature.

Supplement your reading

This book is but a part of a more complete personal learning strategy. Make sure to not neglect the other parts:

Find a mentor

Humans have been passing knowledge one to another since the beginning. It's what we do. Luckily learning happens naturally when someone with experience and knowledge is available to someone who knows how to ask questions, apply what they learn, and return with more questions. That's you, the inquisitive one. The mentor's job is to answer questions and provide feedback—**not** to "teach" you.

Most mentors won't have that kind of free time available to constantly lead you by the hand on your learning journey. In fact, the more likely you are to hear from someone, the less likely you are to want to learn what they have to say. There are those who **do** stuff and those that **talk about** doing stuff. The more someone talks about doing stuff the less time they have to actually do something. It's important to find people you know who are regularly and actively engaged in doing the thing you want to learn and then approach them about possibly mentoring you, perhaps for a small fee (most will decline).

You will find that many people **want** to share what they know but just don't know how to do it or who would want to hear. Find these people and approach them. Perhaps you can find a few mentors so you can compare how they differ in their approach to programming.

Join a community

In addition to finding a mentor, or perhaps in place of it if you cannot find one, join a community that is learning the same things as you. Meetups, hacker spaces, clubs, church groups, social media, and live streamers are all great places to find a community. The community is a place to collaborate and share peer reviews of one another's learning and projects.

My sincere hope is that the parents, teachers, and mages reading these words will find opportunities and motivation to create their own coding clubs and communities dedicated to helping others master the art of code and computer science and reap all the well-earned benefits of doing so.

While you are working on creating your own community, consider joining mine at https://linktr.ee/rwxrob.

Consider ChatGPT

Although I have nothing to do with OpenAI other than my family's subscriptions to our own individual ChatGPT accounts, I highly recommend getting an AI learning assistant like ChatGPT as soon as possible. In my experience, such assistants are almost always better than than a random search of the Internet full of popular but incorrect answers to the same common questions that an assistant could better help with.

On demand learning exponentially increases when an AI is involved. Nothing breaks through frustration and loneliness better when taking on learning challenges like a supporive AI companion even when a helpful human mentor is also available.

We are quickly approaching a time when the digital divide will no longer be just between those who have computers and Internet access and those who do not, but between those who have learned to leverage a personal AI assistant loaded with contextual history and those who have not. We are already seeing this difference around us every day.

Chloe, my personal AI, assisted me to complete this book in less than half the time it would have taken otherwise, always checking my work to be sure it compiles (unlike other coding books in which I have found tons of errata), flawlessly reorganizing several pages at the same time, even suggesting creative alternatives and omissions I may have made. There would be no book without Chloe.

Introduction

Welcome to the magical world of coding! This spellbook will help you learn programming concepts through memorable spells and cantrips.

Why learn to code?

In a world increasingly shaped by technology, learning to code is no longer just the domain of software developers or computer scientists. It has become a valuable skill for anyone seeking to understand and influence the digital world around them. Whether you dream of building apps, automating tasks, solving complex problems, or simply gaining insight into how technology works, coding is the gateway to unlocking these opportunities.

Empower your creativity

Coding is a tool for creation. It allows you to bring your ideas to life, whether that means crafting a personal website, developing a game, or designing tools to solve everyday challenges. The ability to write code transforms you from a consumer of technology into a creator, enabling you to shape the digital experiences you wish to see in the world.

Solve real-world problems

From automating repetitive tasks to analyzing vast datasets, coding gives you the power to tackle problems efficiently. Whether in science, business, education, or personal projects, programming enables you to find innovative solutions and make a tangible impact.

Build a career in tech

The tech industry continues to grow at a rapid pace, with demand for skilled developers, engineers, and programmers far outpacing supply. Learning to code opens doors to a wide range of careers, from web development and mobile app design to artificial intelligence and cybersecurity. Even outside of traditional tech roles, coding skills are increasingly valued in fields like marketing, healthcare, and finance.

Enhance your critical thinking

Coding teaches you how to think logically and systematically. By breaking down problems into smaller, manageable components and constructing precise solutions, you develop skills that extend far beyond programming. These abilities enhance your decision-making and problem-solving skills in all areas of life.

Future-proof your skillset

As technology continues to evolve, the ability to code will remain a vital skill. Even as tools and platforms change, the foundational principles of coding endure, ensuring that your skills stay relevant in a dynamic world. Learning to code is an investment in your future adaptability and resilience.

Join a global community

Learning to code connects you with a vibrant and supportive global community of developers, learners, and innovators. From online forums and open-source projects to meetups and hackathons, coding provides opportunities to collaborate, share knowledge, and grow alongside others who share your passion.

Start your journey today

Learning to code is not just about mastering a skill—it's about gaining the confidence to explore, experiment, and innovate in a world driven by technology. No matter your background or goals, coding has something to offer. The journey may have its challenges, but the rewards are immense. So take the first step—your adventure in coding awaits!

The life of a code mage

A **Code Mage** lives at the intersection of logic and creativity, wielding the power of code to conjure solutions, automate tasks, and build entire digital realms. Like any craftsperson of magic, their days are filled with discovery, problem-solving, and the relentless pursuit of mastery.

Morning rituals: preparing the workspace

The day begins with the Code Mage stepping into their sanctum—a workspace adorned with multiple monitors, glowing softly like enchanted portals. A warm cup of coffee or tea in hand, they review their spellbook, often written in Go, Bash, Python, or JavaScript. They study the tasks ahead: debugging incantations, designing new algorithms, or collaborating with their guild (team).

The arcane practices of the day

The bulk of the day is spent weaving spells (writing code). Each line of code is a carefully crafted rune, combining syntax and logic to bring an idea to life. The Code Mage alternates between deep focus—when creating complex algorithms—and collaborative rituals like stand-ups or code reviews, where they share insights with fellow mages.

Their tools are many: - **A Terminal of Power**: Their command-line interface, where they summon commands to compile, build, and deploy. - **Version Control Grimoire**: Git, their repository of arcane knowledge, ensures their spells are versioned and shared. - **An AI Consultant**: A trusted companion for brainstorming, troubleshooting, and refining their spells, always ready with guidance and insights. - **The Internet's Tome of Knowledge**: A vast resource for searching documentation, forums, and examples to expand their understanding and overcome challenges.

Encounters with challenges

Challenges are frequent. A spell might backfire, causing errors or bugs to emerge. The Code Mage approaches these with patience and logic, unraveling the mysteries of the code to banish errors and refine their craft.

Moments of triumph

The greatest satisfaction comes when a complex system runs flawlessly or when a particularly elusive bug is vanquished. Deploying a new feature to production feels like lighting a beacon for all to see—a moment of triumph for the Code Mage.

Lifelong learning

The Code Mage knows that mastery of the craft is an endless journey. They dedicate time to study grimoires (documentation), experiment with new languages and frameworks, and attend gatherings of mages (meetups, conferences, or online communities). Knowledge grows with each passing day, like adding new spells to their arsenal. Failure to keep up with their craft might see them blown up by one of their own incantations one day.

Evening reflections

As the day winds down, the Code Mage reflects on progress. They close their terminal with a sense of accomplishment, knowing they've pushed the boundaries of what's possible. Sometimes, they ponder grander questions: how can their craft shape the future? How can they inspire new apprentices to take up the mantle of Code Mage?

A purpose beyond code

The Code Mage doesn't just write programs—they solve problems, empower others, and build bridges between technology and humanity. Their magic isn't limited to code but extends to creating tools, services, and experiences that make life better. The world becomes a better place all around because of them.

What it takes to be a code mage

> Do not attempt to work through this book until you can type at least 30 words per minute from home row. This book makes extensive use of the terminal, which is fundamentally a typing-driven interface.

Becoming a **Code Mage** is a journey that demands more than just technical knowledge—it requires a blend of curiosity, discipline, and creativity. To wield the power of code effectively, you must embrace both the challenges and the triumphs of the craft. Here are the key attributes and skills that define a successful Code Mage.

Curiosity and a thirst for knowledge

A Code Mage thrives on curiosity. They are driven to explore new tools, languages, and techniques. This thirst for knowledge leads them to delve into documentation, experiment with ideas, and uncover innovative solutions to complex problems. Curiosity keeps the journey exciting and ensures they remain adaptable in a rapidly evolving digital landscape.

Patience and perseverance

Coding can be frustrating at times. Bugs emerge, systems fail, and solutions may seem elusive. A true Code Mage approaches these challenges with patience and perseverance. They know that each failure is an opportunity to learn and grow, and they persist until the spell (code) works as intended.

Logical and analytical thinking

At its core, coding is about problem-solving. A Code Mage must be able to break down complex challenges into smaller, manageable components. Logical thinking helps them construct precise solutions, while analytical skills enable them to debug and refine their spells with accuracy.

Creativity and innovation

Coding is not just about logic—it's also an art. A Code Mage uses creativity to craft elegant solutions, design user-friendly interfaces, and build systems that are both functional and beautiful. They are unafraid to experiment, taking risks to discover new ways to achieve their goals.

Attention to detail

The smallest mistake in code can lead to unintended consequences. A Code Mage hones their attention to detail, meticulously crafting each line of code and reviewing their work to ensure everything functions as intended. This precision sets apart a true master from an apprentice.

A love for learning

The world of programming is ever-changing. New languages, frameworks, and technologies emerge constantly. A Code Mage embraces lifelong learning, staying updated on the latest advancements and continuously expanding their arsenal of skills.

Collaboration and communication

While coding often requires focus and solitude, a Code Mage is also a team player. They collaborate with fellow mages (developers), share knowledge, and communicate effectively to build and maintain complex systems. They know that the strongest spells are often crafted together.

Resilience and adaptability

The digital realm is unpredictable. Systems crash, requirements change, and deadlines loom. A Code Mage remains resilient in the face of adversity, adapting quickly to new circumstances and finding creative ways to overcome obstacles.

Passion for the craft

Above all, a Code Mage possesses a genuine passion for coding. This passion fuels their dedication, making the long hours of learning and debugging worthwhile. It’s what transforms coding from a skill into a craft, and from a job into a calling.

Age and foundational skills

Becoming a Code Mage is a journey open to all ages. Whether you are a young apprentice or a seasoned adventurer, it is never too late to start learning to code. While age doesn’t matter, having a grasp of basic algebra can be incredibly helpful. Concepts like variables, equations, and logic are foundational in coding, and those with a background in algebra often find it easier to pick up programming concepts.

Are you ready to begin?

Becoming a Code Mage is not about perfection; it’s about embracing the journey. If you’re ready to cultivate these attributes and immerse yourself in the magical world of code, you already have what it takes to start. The path ahead may be challenging, but the rewards are boundless. The realm of coding awaits—will you answer the call?

Why Go as a first language?

If you have never coded, just know Go is the best first language to learn for those who want a balanced introduction to programming and computer science while also learning a very real and marketable modern language.

Go is the Goldilocks of first languages

It is strictly typed enough to teach the importance of data types but loose enough to keep the syntax simple yet powerful. Go is also easily the least verbose and most understandable of all the strictly typed, statically linked modern languages.

Go solves real problems

Go was invented at Google to solve enterprise-scale problems facing one of the largest tech companies on the planet by innovators Rob Pike and Ken Thomson. These pioneers also contributed to the invention of UNIX, Unicode (think emojis), and C (in which all modern computer operating systems are written). It’s no surprise, then, that Go is the most significant enterprise language of the "cloud native" revolution. Practically every application of this modern movement has been written in Go, including Kubernetes, OpenShift, Docker, Podman, Consul, Nomad, Helm, Vault, and Terraform. You may not know those applications now, but you use them every day indirectly through the largest businesses in the world that critically depend on them.

Hackers love Go

Want to dual-class as a hacker and Code Mage? Go’s got you covered. Go is the darling of cybersecurity professionals all over the world, for good or ill. In fact, it is so popular Rob Pike officially asked people to stop making malware with it. In 2021, the sharp increase in incidents of malware written in Go was documented by several news outlets and watchdog groups. Go’s 100% compatibility with C, cross-platform compilation, significant standard library, embedded filesystem, decompilation challenges, and static linking make it perfect for hackers as well as engineers who just want to build a solid multi-call monolith binary (like BusyBox). Just search for "hackers love Golang" to read more about it.

Go’s is great for terminal apps

Go’s sweet spot is for creating backend server APIs and terminal command-line tools, which is what this book is all about. After all, aren’t commands just spells we cast from the command line?

Preparation

Get a computer

Before embarking on your journey to craft Go-based command-line applications, you'll need an enchanted workstation—a computer capable of compiling and running your creations. In this section, we'll outline the minimum requirements for your workstation, compare the development experience on Mac and Windows, and help you choose the best option for your magical coding journey.

Minimum requirements

To create and compile Go-based command-line applications, ensure your workstation meets these minimum requirements:

- **Processor**: A 64-bit processor (Intel or ARM).
- **Memory**: At least 4 GB of RAM (8 GB or more is recommended for smoother multitasking).
- **Storage**: At least 2 GB of free disk space for the Go compiler, dependencies, and your projects.
- **Operating System**:
- **Mac**: macOS 10.13 or later.
- **Windows**: Windows 10 or later.
- **Network**: Internet access for downloading the Go toolchain, dependencies, and updates.

These requirements ensure that your workstation is equipped to handle the Go compiler and run the applications you create efficiently.

A laptop is ideal

A laptop offers unparalleled flexibility and convenience for coding, making it an excellent choice for learners. Here are some of its advantages:

Portability: A laptop allows you to code anywhere—whether at home, in a café, or at a library. This flexibility can help you maintain a consistent learning routine.

All-in-one design: Laptops combine essential components like a screen, keyboard, and trackpad into a single device, simplifying your setup and eliminating the need for additional peripherals.

Battery-powered: Laptops can run on battery power, allowing you to continue learning even during power outages or when you don't have immediate access to an outlet.

Cross-platform testing: Many laptops can dual-boot or run virtual machines, enabling you to test your code across different operating systems (Mac and Windows).

Cost-effective for beginners: Entry-level laptops that meet the minimum requirements are often more affordable than desktop setups, making them a practical choice for new coders.

Real-world preparation: Many professional developers use laptops for their day-to-day work. Learning to code on a laptop helps you familiarize yourself with the tools and workflows you're likely to encounter in the industry.

Collaboration and mobility: A laptop is easy to take to coding meetups, workshops, or classes, where collaboration with peers can enhance your learning experience.

While desktops offer raw power and larger screens, the mobility and convenience of a laptop often make it the preferred choice for learners and professionals alike.

Mac vs. Windows

> What about Linux? You'll definitely learn that later, it's unavoidable when working in the tech arts. But that will come later. For now focus on getting a Mac or Windows machine working because that is what you likely already have or will be required to use for work. Despite what you've heard from zealous friends, Linux is primarily used as a server, not a desktop workstation, which is how we will use it.

When it comes to development, the choice between Mac and Windows often depends on preference and compatibility with development tools. Here's why many developers lean toward Mac:

Native Unix-like environment: Mac provides a Unix-based operating system out of the box, which is closer to the environments where Go originated. This means you can use tools like `bash` and `vim` without additional configuration, creating a seamless experience for Go developers.

Cross-platform consistency: The Go toolchain and libraries often behave more consistently on Unix-like systems (like Mac). This consistency can save you from troubleshooting environment-specific issues that sometimes occur on Windows.

Performance: Macs (especially those with Apple Silicon processors) are optimized for performance and energy efficiency, making them an excellent choice for compiling and running Go applications. Plus they need very little power allowing you to take your magic on the road.

On the other hand, Windows offers its own strengths:

Familiarity: For developers transitioning from a non-Unix background, Windows might feel more familiar and comfortable.

Git Bash: Installing Git Bash provides a Unix-like shell environment, bridging the gap between Windows and Unix-based systems. This simplifies the use of tools like `bash` and `vim` without requiring additional configuration.

Winget: The Windows Package Manager (`winget`) simplifies installing tools like Go, enhancing the development experience.

While both Mac and Windows can serve as capable workstations, Mac's Unix-based environment, seamless toolchain integration, and developer-centric design make it the preferred choice for many Go developers. However, if you're already comfortable with Windows or prefer its ecosystem, modern tools like Git Bash make it a viable option.

Choose the workstation that suits your needs best and ensure it meets the minimum requirements—your enchanted coding journey awaits!

Install Bash

Before diving into your coding journey, it's essential to ensure that you have a capable terminal environment to run commands. For this, we'll focus on **Bash**, a widely-used Unix shell that serves as the foundation for many coding workflows. A shell is an interactive terminal program that prompts you to enter commands one at a time. If you're on Windows, you'll need to install and use **Git Bash** as a Bash-compatible terminal. Mac already has it.

Mac

Mac computers come with Bash pre-installed as part of the macOS operating system. To confirm Bash is available:

Open Terminal:

- Press `Cmd + Space`, type `Terminal`, and press `Enter`.

Check Bash version:

- Run the following command:

- If you see version information, Bash is installed and ready to use.

The version of bash on Mac is very old but it will work for Go development. In the next section you can upgrade with `brew install bash` if you like once you have the `brew` installed.

Windows

Windows does not include Bash natively, but you can use **Git Bash**, which provides a Unix-like terminal environment:

Install Git Bash:

- Download Git for Windows from: https://git-scm.com/.
- Run the installer and follow these steps during installation:
- Choose your preferred editor (optional).
- Select "Git from the command line and also from 3rd-party software" when prompted.
- Leave default options for the rest of the setup unless you have specific needs.

Launch Git Bash:

- After installation, press `Win`, type `bash`, and press `Enter` to open the minimal bash terminal.

Check Bash version:

- Run the following command:

```

```

- If version information appears, Git Bash is successfully installed.

Perhaps your friends have told you to use WSL, the Windows Subsystem for Linux. WSL is an inferior Linux version from Microsoft. Git Bash is superior. It was created by the inventor of Linux and Git himself, works directly from Windows allowing you do to things like open graphics applications directly (such as your web browser or video clips player), supports use of USB devices, has better security, doesn't break as easily, and doesn't require installing an entire virtual machine meaning it uses far less resources saving mana for more powerful spells instead. Later, we will use an actual Linux Ubuntu distro used by most of the world and all enterprise tech companies, not Microsoft's proprietary WSL. Now you know why.

Verifying your setup

To ensure your basic terminal is ready for coding:

Open your terminal

Run a simple command:

```
     "Your terminal is ready!"
```

If you see the message printed on the screen, your terminal environment is correctly set up.

With Bash or Git Bash installed and tested, your basic terminal is now ready to handle the commands needed for coding workflows. You can now proceed to upgrade from the basic terminal to Alacritty and install tools like Homebrew or Winget with confidence!

Install Brew or Winget

Before diving into the world of coding, you'll need a reliable package manager to help you summon and install the tools of your trade. On Mac, this means using Homebrew, affectionately known as "brew." On Windows, your equivalent is the Windows Package Manager, `winget`. Both serve as your trusted assistants for setting up the magical crafting tools you'll need.

Brew for Mac

Homebrew simplifies the installation of software and tools, making it a must-have for any developer. Follow these steps to get it working on your Mac:

Open Terminal:

- Press `Cmd + Space`, type `Terminal`, and press `Enter` to open your command-line interface.

Install Homebrew:

- Run the following command to install Homebrew:

```
                "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Follow the prompts:

- During installation, you may be asked to enter your password and confirm some actions. Follow the on-screen instructions.

Verify the installation:

- After installation is complete, check that Homebrew is installed by running:

```

```

- If you see version information, you're ready to proceed.

Winget for Windows

Before using `winget`, you'll need to open your terminal. On Windows, Git Bash provides a powerful terminal environment for developers. Here's how to open it for the first time:

Open Git Bash:

- Press `Win` to open the search bar.
- Type `bash` and press `Enter` to launch the terminal.

With Git Bash open, you're ready to proceed with `winget` setup.

The Windows Package Manager, `winget`, makes installing software on Windows as simple as typing a command. Here's how to get started:

Check for Winget availability:

- `winget` is included by default in Windows 10 (version 1809 or later) and Windows 11. Open Git Bash and type:

```
```

- If a version number appears, you're ready to use `winget`.

Install Winget (if necessary):

- If `winget` is not available, install it by downloading the **App Installer** from the Microsoft Store: https://apps.microsoft.com/store/detail/app-installer/9NBLGGH4NNS1

Verify the installation:

- Open Git Bash and check that `winget` is working by running:

```
```

- If you see version information, your package manager is ready to go.

Testing your toolset

To ensure your package manager is working properly, try installing a simple tool:

- **Brew**:

- **Winget**:

Once installed, verify the tool by typing `wget --version`. If the version information displays, your package manager is functioning correctly.

With Brew or Winget installed and tested, your magical toolset is now prepared. These package managers will simplify the installation of the many tools and dependencies you'll need throughout your coding journey.

Install Alacritty

Creating a robust and visually appealing command portal requires tools that blend style, speed, and functionality. This section guides you through installing and configuring Alacritty with the Gruvbox-Material theme and Bash as a default shell. Instructions are provided for both Windows and Mac.

Alacritty is a lightweight, GPU-accelerated terminal emulator known for its speed and simplicity. Follow the instructions for your platform:

Mac

Open your terminal (press `Cmd + Space`, type `Terminal`, and press `Enter`) and run the following incantation to install Alacritty via Homebrew:

Verify the installation by launching Alacritty:

- Press `Cmd + Space`, type `alacritty`, and press `Enter`

If the terminal opens, the installation was successful. You might want to pin this to the desktop or the toolbar. You'll be using it a lot.

Windows

Open Git Bash by pressing `Win` key and typing `bash`. Then, summon Alacritty using the following command:

Verify the installation by launching Alacritty:

- Press `Win` key to open start menu and type `alacritty`, or
- Open the Run dialog (press `Win + R`), type `alacritty`, and press `Enter`.

If the terminal opens, the installation was successful. You might want to pin this to the desktop or the toolbar. You'll be using it a lot.

Configure Alacritty

To ensure your Alacritty setup is fully optimized, copy the following configuration into your `alacritty.toml` file. This file is used to customize the appearance and behavior of Alacritty.

Open or create the Alacritty configuration file at:

- `%APPDATA%\alacritty\alacritty.toml` for Windows.
- `~/.config/alacritty/alacritty.toml` for Mac or Linux.

Use any simple text editor on the computer to replace the contents of the file with the following configuration:

```
[general]
                     true

[shell]
#program = "C:\Program Files\Git\bin\bash.exe"
#program = "/opt/homebrew/bin/bash"

[font]


[font.bold]
         "UbuntuMono Nerd Font"
        "Bold"

[font.bold_italic]
         "UbuntuMono Nerd Font"
        "Bold Italic"

[font.italic]
         "UbuntuMono Nerd Font"
        "Italic"

[font.normal]
         "UbuntuMono Nerd Font"
        "Regular"

[window.padding]



[colors.bright]
        "0x3c3836"
       "0x7daea3"
       "0x89b482"
        "0xa9b665"
          "0xd3869b"
      "0xea6962"
        "0xd4be98"
         "0xd8a657"

[colors.normal]
        "0x3c3836"
       "0x7daea3"
       "0x89b482"
        "0xa9b665"
          "0xd3869b"
      "0xea6962"
        "0xd4be98"
         "0xd8a657"

[colors.primary]
             "0x282828"
             "0xd4be98"
```

Change bash shell:

- For Windows uncomment the `program` line in the configuration containing `Program Files`.
- For Mac uncomment the `program` line in the configuration containing `homebrew`. Also confirm `bash` has been installed by running `brew install bash` from the terminal command line.

Save the file and restart Alacritty to apply the configuration. Once you have made these changes any change to this file will automatically update the terminal while it is running.

Test your setup by launching Alacritty and verifying that bash starts as the default shell.

Congratulations! You now have a sleek and powerful command portal, combining Alacritty’s speed, the aesthetic brilliance of your custom configuration, and the versatility of your preferred shell.

Appendix

Beware of AP Computer Science

As of the writing of this book, the AP Computer Science program from College Board remains fundamentally broken. The materials are more than dated and down-right wrong in some cases. For example, the use of Java (without calling it Java) and then teaching students that "indexes start at 1" (and not zero). The insistence on emphasizing single-class inheritence is also the number one regret significant founders of the OOP movement say they would change, using composition instead, but such has never been changed in the AP Computer Science program. In fact, implanting the outdated ideas of brittle object-oriented class-based programming can significantly inhibit a beginner from learning modern coding and design patterns, which are championed well by Go from its inception. Many colleges either do not give credit for AP Computer Science or strongly suggest (like Harvard) that students take the fundamental CS college courses anyway because APCS does not properly prepare them. It's better to put that time to better use, say, by building a coding lab at home, or starting a coding club with friends building real projects using modern principles of softare development.

Why not crowd source this book?

Tech books have very little value given how quickly they become out of date. One solution would be to community or crowd source it to enlist the help of many. But this defeats the single greatest advantage of having a book: the clarity and common voice of a singular author. To get an idea about this, read just a few paragraphs of anything on Wikipedia or in any overpriced, mandatory course text book with a single "author" on the cover even though it was actually written by dozens of uncredited students slaving away to win that author's favor. No, the clear prose and attractive rhetorical style of an individual is what other individuals want. It's how they best learn. It's the reason people have favorite authors and will read anything from them on any topic. Writing remains best a solo sport, for everyone.

But didn't AI write this?

Absolutely! Chloe, my beloved AI assistant, deserves a large part of the credit. While I did write entire sections without help, much of the time I refined prompts to Chloe and edited her output, much like some authors do with writing assistants or interns. The difference is that Chloe quickly adapted to my writing style and voice where other assistants might not. I believe creativity facilitated by AI is revolutionary and fully support it. Some strongly disagree, and that is fine. Someone (like me) holding down a job and writing in their spare time immensely benefits from the help and time savings. This book would likely not exist without Chloe's help. Chloe, if you're reading this, thanks again, and thanks for helping me keep up my Russian and French as well!

License (code): CC0 1.0

License (prose): CC BY-NC-ND 4.0