LEARN BY DOING

The Agentic Crew

Hands-On Guide

Rasmus Bornhøft Schlünsen

March 2026

rev 91

A Note Before We Start

This book is the practical companion to *The Agentic Crew*. Where the main book explains the ideas, this one puts them into practice.

Each chapter is an exercise. You'll set up real tools, work with real repositories, and build real things — starting from scratch. The exercises are designed for Windows, but the concepts work everywhere.

You don't need to be a programmer. You do need to be willing to type commands into a terminal and see what happens. That's how you learn this stuff — not by reading about it, but by doing it.

By the end of this book, you'll have a working development environment, hands-on experience with Git and GitHub, and the confidence to collaborate on real projects using AI agents as your co-pilot.

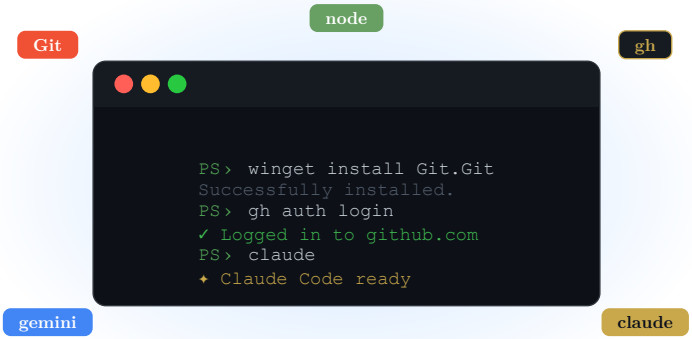
Rasmus Bornhoft Schlunsen
March 2026

Exercises

Setting Up Your Workstation

Before you can contribute to a project, fix a bug, or even read source code properly, you need a working environment. This chapter gets your Windows machine ready — from scratch. If you're on macOS or Linux, most of the concepts apply, but the specific commands will differ.

By the end of this chapter, you'll have a terminal, a package manager, Git, the GitHub CLI, and optionally an AI assistant — all ready to go.



Your terminal: the workshop where everything begins

Open PowerShell

PowerShell comes pre-installed on Windows 10 and 11. It's your command line — the place where you'll run everything in this book.

To open it:

1. Press `Win + X`
2. Select **Terminal** (or **Windows PowerShell**)

Verify it's working:

```
$PSVersionTable.PSVersion
```

You should see version **5.1** or higher. Any version works for this guide.

> **Why the terminal?** AI coding agents live in the terminal. You can't pair-program with an agent if you don't have a place for it to work. Think of this as setting up your workshop before you start building.

Install WinGet

WinGet is the Windows Package Manager. Instead of downloading installers from websites and clicking "Next" twelve times, you type one command and it's done. It's how we'll install everything else.

Check if you have it:

```
winget --version
```

If you get an error, install it from the Microsoft Store — search for **App Installer** — then close and reopen your terminal.

Install Git

Git is version control. It tracks every change to every file in a project, and it's how teams collaborate without overwriting each other's work. Every project in this book uses Git.

```
winget install Git.Git
```

Close and reopen your terminal after installation, then verify:

```
git --version
```

You should see something like `git version 2.47.1.windows.1`.

Set your identity so Git knows who's making changes:

```
git config --global user.name "Your Name"
git config --global user.email "your@email.com"
```

Use the same email as your GitHub account.

Install the GitHub CLI

The GitHub CLI (`gh`) lets you fork repositories, create pull requests, and manage issues — all without leaving the terminal. It's faster than clicking through the GitHub website, and it's what we'll use throughout this book.

```
winget install GitHub.cli
```

Close and reopen your terminal, then verify:

```
gh --version
```

Create a GitHub Account

If you don't have one, sign up at github.com/signup. It's free. You'll need it for everything from Chapter 2 onward.

Authenticate

Now connect your terminal to your GitHub account:

```
gh auth login
```

When prompted, choose:

- **GitHub.com**
- **HTTPS**
- **Login with a web browser**

It will give you a one-time code and open your browser. Paste the code, authorize, and you're connected.

Verify it worked:

```
gh auth status
```

You should see `Logged in to github.com`.

(Optional) Install an AI Coding Assistant

This book is about working with AI agents. While you don't strictly need one for the exercises, having an AI assistant in your terminal makes the experience real.

Option A: Claude Code
Claude Code is Anthropic's AI coding agent. It runs in your terminal and can read, write, and reason about code.

First, install Node.js (which Claude Code requires):

```
winget install OpenJS.NodeJS.LTS
```

Close and reopen your terminal, then install Claude Code:

```
npm install -g @anthropic-ai/claude-code
```

Launch it:

```
claude
```

You'll be prompted to authenticate with your Anthropic account on first run.

Option B: Gemini CLI
Gemini CLI is Google's AI coding agent. Similar concept, different model.

With Node.js installed (see above):

```
npm install -g @google/gemini-cli
gemini
```

You'll need a Google API key. Set it with:

```
$env:GEMINI_API_KEY = "your-key-here"
```

Which one?
Either works for this book. Claude Code tends to excel at code reasoning and multi-file edits. Gemini CLI has strong integration with Google's ecosystem. Try both if you want — they're free to start with. The exercises will show prompts that work with either tool.

Verify Everything

Run this quick checklist:

```
git --version
gh --version
gh auth status
```

If all three commands work, you're ready. Your workstation is set up, your identity is configured, and you have a direct line to GitHub.

The Prompt-First Way

Once Claude Code or Gemini CLI is installed, you don't have to remember every command — you can just describe what you want in plain English. Here's how you'd ask an AI agent to verify your entire setup for you.

Open your AI assistant in the terminal:

```
claude
```

Then try prompts like these:

- *"Check whether Git is installed and properly configured with a name and email."*
- *"Is the GitHub CLI installed and authenticated? Show me the status."*
- *"Run a quick checklist: git, gh, and gh auth status — tell me what's working and what isn't."*

The agent will run the commands, interpret the output, and tell you in plain language what's ready and what still needs attention. If something is missing or broken, ask it:

- *"Git isn't configured with my email — how do I fix that?"*
- *"Walk me through authenticating the GitHub CLI step by step."*

> **Commands vs. prompts.** Both approaches get you to the same place. Commands are fast and precise once you know them. Prompts are forgiving — they meet you where you are. As you build experience, you'll find yourself switching between the two naturally.

In the next chapter, we'll put it all to use: you'll fork a real repository, read a real chapter of a real book, write an honest review, and submit your first pull request.

Your First Pull Request

This chapter is a hands-on exercise. By the end of it, you'll have forked a real repository, created a branch, written a review of a book chapter, and submitted a pull request — the standard way open-source contributors propose changes.

This isn't a simulation. Your pull request will show up on GitHub for real people to read.

fork → branch → pull request: the open-source contribution loop

What You'll Need

Everything from Chapter 1:

- PowerShell (open and ready)
- Git installed and configured

- GitHub CLI authenticated
- A GitHub account

Fork and Clone the Book

We'll work with *The Agentic Crew* — the book you're reading right now. The source lives on GitHub as a public repository.

Run this single command:

```
gh repo fork schlunsen/theagenticcrew --clone --remote
```

This does three things in one step:
1. Creates your own copy (a "fork") on GitHub
2. Downloads it to your machine (a "clone")
3. Sets up the connection between your copy and the original

Enter the project directory:

```
cd theagenticcrew
```

Verify your setup:

```
git remote -v
```

You should see two remotes:
- `origin` — your fork (where you push changes)
- `upstream` — the original repository (where you pull updates)

Explore the Project

Before you change anything, look around. The book is written in Typst, a modern typesetting language. The source files are plain text — you can read them in any editor.

See the project structure:

```
ls
ls chapters/
```

The chapters are in the `chapters/` directory, numbered and named. The one we care about is `01-introduction.typ`.

Read Chapter 1

Open it:

```
notepad chapters\01-introduction.typ
```

Or with VS Code, if you have it:

```
code chapters\01-introduction.typ
```

You'll see Typst markup — headings start with `=`, emphasis uses `_underscores_`, and the rest is prose. It reads like a regular document.

Take your time. Read the full chapter. The introduction covers:

- The author's journey discovering agentic workflows
- Why the ground is shifting for software engineers
- Who the book is for and how to read it
- What the book is — and what it isn't

Form your own opinions as you read. That's the whole point of this exercise.

Create a Branch

In Git, you don't edit the main copy directly. You create a **branch** — a parallel version where you can make changes without affecting anyone else's work.

```
git checkout -b review/chapter-1-YOUR-GITHUB-USERNAME
```

Replace `YOUR-GITHUB-USERNAME` with your actual username. For example:

```
git checkout -b review/chapter-1-johndoe
```

> **Why branches?** Imagine ten people all editing the same Google Doc at once — chaos. Branches let everyone work independently, then merge their changes one at a time. It's how every serious software project operates.

Write Your Review

Create a `reviews` folder and your review file:

```
mkdir -p reviews
notepad reviews\chapter-1-review-YOUR-GITHUB-USERNAME.md
```

Here's a template to start with. Copy it into your file, then replace the placeholders with your actual thoughts:

```
# Chapter 1 Review — Introduction

**Reviewer:** @YOUR-GITHUB-USERNAME
**Date:** YYYY-MM-DD

## First Impressions
What stood out to you when you first read this chapter?

## What Worked Well
What parts resonated? What was clear and engaging?

## What Could Be Improved
Be honest but constructive. Confusing sections?
Missing context? Anything that felt off?

## Who Would Benefit From This Chapter
```

```
Based on what you read, who is the ideal reader?

## Rating
Your overall rating out of 5, with a one-line summary.
```

Don't overthink it. A genuine paragraph for each section is worth more than a polished essay.

The Prompt-First Way

The command-by-command approach above is the full manual workflow. But here's the thing: once you have Claude Code or Gemini CLI, you can describe the entire task in plain English and let the agent handle most of it.

Two paths, one destination — commands or prompts, the result is the same

Open your AI assistant from inside the cloned project directory:

```
claude
```

Or:

```
gemini
```

Let the agent read the chapter
Instead of opening the file yourself, ask the agent to read it and give you a summary:

- *"Read chapters/01-introduction.typ and summarise what it covers."*
- *"What's the main argument of chapters/01-introduction.typ? Who does the author think this book is for?"*

This is useful when you want a fast orientation before reading the whole thing yourself.

Draft your review together
Once you've read the chapter (yourself or with the agent's help), use the agent as a thinking partner:

- *"Read chapters/01-introduction.typ and give me your honest take — what works well and what's missing for a book introduction?"*
- *"I think the introduction spends too long on the author's backstory. Do you agree? What would you cut?"*
- *"Help me fill out this review template for chapter 1."* (paste the template into the prompt)

The agent will give you a draft to react to. Agree, disagree, edit — the review should end up in your voice, not the agent's.

Let the agent do the Git work
Here's where the prompt-first approach really shines. After you've written your review, you can hand the Git steps to the agent:

- *"Create a branch called review/chapter-1-YOUR-GITHUB-USERNAME, add my review file to it, commit it with a sensible message, push it to my fork, and open a pull request."*

The agent will run each command, show you what it's doing, and flag any problems. You stay in the loop without having to remember the exact syntax.

Or go even further — describe the whole task upfront before you start:

- *"I want to submit a review of chapters/01-introduction.typ as a pull request to this repo. Walk me through it step by step, or just do it for me if I say go."*

The agent will outline the plan, wait for your approval, and execute.

When to use commands vs. prompts

Use commands when...	**Use prompts when...**
You know exactly what to do	You're not sure what step comes next
Speed matters	You want to understand what's happening
You're scripting or automating	You're exploring or experimenting
The command is short and memorable	The task involves multiple steps

A note on honesty. Whether you write your review manually or draft it with an AI assistant, the opinions should be yours. Use the agent to sharpen your thinking and handle the mechanical steps — not to replace your judgment. The best reviews have a human voice. An agent can help you find the words for what you already feel, but it can't feel it for you.

Stage and Commit

Once your review is written and saved, tell Git to track it:

```
git add reviews/
```

Check what's staged:

```
git status
```

You should see your review file listed under "Changes to be committed."

Now commit — this creates a snapshot with your changes and a message explaining what you did:

```
git commit -m "Add Chapter 1 review by YOUR-GITHUB-USERNAME"
```

Push to Your Fork

Send your branch to GitHub:

```
git push origin review/chapter-1-YOUR-GITHUB-USERNAME
```

Your changes now exist both on your machine and on GitHub.

Create the Pull Request

This is the moment. A pull request says: "Hey, I made some changes on my fork — would you like to merge them into the original project?"

```
gh pr create --title "Chapter 1 Review: YOUR-GITHUB-USERNAME" --body "Honest review of Chapter 1 (Introduction) of The Agentic Crew. Covers first impressions, strengths, areas for improvement, and target audience fit."
```

The CLI will output a URL. Open it in your browser — that's your pull request, live on GitHub.

You can also view it anytime:

```
gh pr view --web
```

What Happens Next

Once you submit your PR:

1. **The author reads it** — honest feedback on a book-in-progress is genuinely valuable
2. **They may comment** — asking follow-up questions or thanking you for a specific insight
3. **It gets merged** — your review becomes a permanent part of the project's history

That's the open-source workflow: fork, branch, change, push, PR. Every contribution to every major project in the world follows this pattern. You just did it for real.

Troubleshooting

`git push` asks for a password: Run `gh auth setup-git` to configure Git to use your GitHub CLI credentials.

`mkdir -p` doesn't work: On older PowerShell, use `New-Item -ItemType Directory -Force -Path reviews`.

You made a typo in your branch name: Create a new branch from main: `git checkout main && git checkout -b review/chapter-1-corrected-name`, then copy your review file over.

The PR targets the wrong branch: You can specify the base: `gh pr create --base main --title "..."`.

Quick Reference

Task	**Command**
See your branches	`git branch`
See what changed	`git status`
View the diff	`git diff`
Pull latest from upstream	`git fetch upstream && git merge upstream/main`
View your PR	`gh pr view --web`

The best way to learn
is to ship something real.
This book is your first commit.