Introduction

Graphical user interfaces (GUIs) are great for a lot of things; they are typically much kinder to newcomers than the stark glow of a cold, blinking cursor. This comes at a price: you can get only so proficient at a GUI before you have to learn its esoteric keyboard shortcuts. Even then, you will hit the limits of productivity and efficiency. GUIs are notoriously hard to script and automate, and when you can, your script tends not to be very portable.

This is all beside the point; we are software developers, and we write programs. What could be more natural than using code to get our work done? Consider the following command sequence:

```
> cd ~/Projects/cli
> vi chapter2.md
```

While these two commands might strike you as opaque, they are a highly efficient means of editing a file.

For most of my career, the command line meant a UNIX shell, like `bash`. The `bash` shell provides some basic built-in commands, as well as access to many other standard (and nonstandard) commands that are shipped with any UNIX system. These commands are single-purpose, require no user interaction, and come with easy-to-use (but hard-to-learn) user interfaces. These attributes let you piece them together in a near-infinite number of ways. Automating sophisticated behavior, performing complicated analysis, and parsing a myriad of text files can be done easily and expediently. This was life for me early on in my career. And it was good.

Then, in the mid-1990s, as Java grew in popularity, the idea of stringing together UNIX command-line utilities to get things done came to be seen as archaic. Java programs eschewed simple text-based configuration and file-based input/output (I/O) for complex hierarchies of XML driven by RPC and HTTP I/O. This allowed for very sophisticated systems to be built, and GUI tools sprang up to abstract away the complexity of building and configuring these systems. Even the act of writing and building code got swallowed up