

Preface

Linux has no shortage of tools. Many are inherited from Unix, with cryptic two-letter names that conjure up images of developers trying to preserve space on a punch card. Happily, those days are long gone, but the legacy remains.

Many of those old tools are still quite useful. Most are highly specialized. Each may do only one thing but does it very well. Highly specialized tools often have many options that can make them intimidating to use. Consider the first time you used `grep` and learned what a regular expression was. Perhaps you haven't mastered regular expression syntax yet (don't worry; no one else has, either). That's not important, because you don't need to be a master of regular expressions to put `grep` to good use.

If there's one thing that I hope you learn from this book, it's that there are many tools out there that you can use without having to master them. You don't need to invest an enormous amount of time reading manuals before you can be productive. I hope you will discover new tools that you may not have been familiar with. Some of the tools this book looks at are quite old and some are new. All of them are useful. As you learn more about each tool, you will find more uses for it.

I use the term *tool* loosely in this book. To me, creating tools is as important as using tools, so I have included various APIs that are not usually covered in much detail in other books. In addition, this book provides some background on the internal workings of the Linux kernel that are necessary to understand what some tools are trying to tell you. I present a unique perspective on the kernel: the user's point of view. You will find enough information to allow you to understand the ground rules that the kernel sets for every process, and I promise you will not have to read a single line of kernel source code.

What you will not find in this book is reconstituted man pages or other documentation stitched into the text. The GNU and Linux developers have done a great

job of documenting their work, but that documentation can be hard to find for the inexperienced user. Rather than reprint documentation that will be out of date by the time you read this, I show you some ingenious ways to find the most up-to-date documentation.

GNU/Linux documentation is abundant, but it's not always easy to read. You can read a 10,000-word document for a tool and still not have a clue what the tool does or how to use it. This is where I have tried to fill in the missing pieces. I have tried to explain not just how to use each tool, but also why you would want to use it. Wherever possible, I have provided simple, brief examples that you can type and modify yourself to enhance your understanding of the tools and Linux itself.

What all the tools in this book have in common is that they are available at no cost. Most come with standard Linux distributions, and for those that may not, I have included URLs so that you can download them yourself.

As much as possible, I tried to keep the material interesting and fun.

Who Should Read This Book

This book is written for intermediate to advanced Linux programmers who wish to become more productive and gain a better understanding of the Linux programming environment. If you're an experienced Windows programmer who feels like a fish out of water in the Linux environment, then this book is for you, too.

Non-programmers should also find this book useful because many of the tools and topics I cover have applications beyond programming. If you are a system administrator, or just a Linux enthusiast, then there's something for you in this book, too.

The Purpose of This Book

I wrote this book as a follow-up to an article I wrote for the *Linux Journal* entitled "Ten Commands Every Linux Developer Should Know." The inspiration for this article came from my own experience as a Linux programmer. In my daily work I make it a point to invest some of my time in learning something new, even if it means a temporary lull in progress on my project. Invariably this strategy has paid off. I have always been amazed at how many times I learned about a tool or feature that I concluded would not be useful, only to find a use for it shortly afterward. This has always been a powerful motivation for me to keep learning. I hope that by reading this book, you will follow my example and enhance your skills on a regular basis.

It's also just plain fun to learn about this stuff. If you are like me, you enjoy working with Linux. Motivating yourself to learn more has never been a problem. Because Linux is open source, you have the opportunity to understand all of its inner workings, which is not possible with closed source environments like Windows. In this book I present several freely available resources available to help you learn more.

How to Read This Book

The chapters are presented such that each chapter can stand on its own. Later chapters require some background knowledge that is presented in the earlier chapters. Wherever possible, I have cross-referenced the material to help you find the necessary background information.

I believe the best way to learn is by example, so I have tried to provide simple examples wherever possible. I encourage the reader to try the examples and experiment.

How This Book Is Organized

Chapter 1, *Downloading and Installing Open Source Tools*, covers the mechanisms used to distribute open source code. I discuss the various package formats used by different distributions and the advantages and disadvantages of each. I present several tools used to maintain packages and how to use them.

Chapter 2, *Building from Source*, covers the basics of building an open source project. I present some of the tools used to build software and alternatives that are emerging. There are several tips and tricks in this chapter that you can use to master your use of `make`. I also show you how to configure projects that are distributed with GNU's `autoconf` tools so that you can customize them to meet your needs. Finally, I cover the stages of the build that are often misunderstood by many programmers. I look at some of the errors and warnings you are likely to encounter and how to interpret them.

Chapter 3, *Finding Help*, looks at the various documentation formats tucked away in your Linux distribution that you may not know about. I look at the tools used to read these formats and discuss effective ways to use them.

Chapter 4, *Editing and Maintaining Source Files*, discusses the various text editors available for programmers as well as the advantages and disadvantages of each. I present a set of features that every programmer should look for in an editor and measure each editor against these. This chapter also covers the basics of revision control, which is vital for software project management.

Chapter 5, *What Every Developer Should Know about the Kernel*, looks at the kernel from a user's perspective. In this chapter you will find the necessary background information required to understand the workings of a Linux system. I introduce several tools that allow you to see how your code interacts with the kernel.

Chapter 6, *Understanding Processes*, focuses on processes, their characteristics, and how to manage them. I cover a good deal of background required to introduce the tools in this chapter and understand why they are useful. In addition, this chapter introduces several programming APIs that you can use to create your own tools.

Chapter 7, *Communication Between Processes*, introduces the concepts behind inter-process communication (IPC). This chapter contains mostly background information required for Chapter 8. Along with each IPC mechanism, I introduce the APIs required to use it along with a working example.

Chapter 8, *Debugging IPC with Shell Commands*, presents several tools available to debug applications that use IPC. It builds on the information from Chapter 7 to help you interpret the output of these tools, which can be difficult to understand.

Chapter 9, *Performance Tuning*, introduces tools to measure the performance of your system as well as the performance of individual applications. I present several examples to illustrate how programming can impact performance. I also discuss some of the performance issues that are unique to multi-core processors.

Chapter 10, *Debugging*, presents several tools and techniques that you can use to debug applications. I look at some open source memory debugging tools including Valgrind and Electric Fence. I also take an in-depth look at the capabilities of `gdb`, and how to use it effectively.