INTRODUCTION

Few open source software developers would deny that GNU Autoconf, Automake, and Libtool (the *Autotools*) have revolutionized the open source software world. However, although there are many thousands of Autotools advocates, there are also many software developers who *hate* the Autotools—with a passion. I believe the reason for this dread of the Autotools is that when you use them without understanding the underlying infrastructure they manage, you find yourself fighting against the system.

This book solves this problem by first providing a framework for understanding the underlying infrastructure of the Autotools and then building on that framework with a tutorial-based approach to teaching Autotools concepts in a logically ordered fashion.

Who Should Read This Book

This book is primarily for the open source software package maintainer who wants to become an Autotools expert. That said, this book also provides instructions to end users who wish to understand what's happening during the process of downloading, unpacking, and building software packages whose build processes are managed by the Autotools. Existing material on the subject is limited to the GNU Autotools manuals and a few internet-based tutorials. For years, most real-world questions have been answered on the Autotools mailing lists, but mailing lists are an inefficient form of teaching because the same answers to the same questions are given time and again. This book provides a cookbook-style approach, covering real problems found in real projects.

How This Book Is Organized

The book starts with an end-user perspective on the Autotools and then moves from high-level development build concepts to mid-level use cases and examples, finally finishing with more advanced details and examples. As though you were learning arithmetic, we'll begin with some basic math—algebra and trigonometry—and then move on to analytic geometry and calculus.

Chapter 1 provides an end-user perspective on the Autotools. It covers topics that a Linux power user, who is not necessarily a software developer, needs to understand in order to take full advantage of the features of Autotools-managed source packages (so-called "tarballs") downloaded from project websites containing perhaps the latest beta version of some software the user would like to try out. Often, Linux users find that the solution to a software problem involves updating to a version that contains the fix for that problem, only to discover that the version they need is so new there is no RPM or Debian package for that version in any of the package repositories for their Linux distribution of choice. Chapter 1 provides relief to the newbie who needs to know what to do with that *tar.gz* file containing that `configure` script and all those *.c* source files.

Chapter 2 begins the discussion of concepts of interest to software developers. It presents a general overview of the packages considered part of the GNU Autotools. This chapter describes the interaction between these packages and the files consumed by and generated by each one. In each case, figures depict the flow of data from hand-coded input to final output files.

Chapter 3 covers open source software project structure and organization. This chapter also goes into some detail about the *GNU Coding Standards (GCS)* and the *Filesystem Hierarchy Standard (FHS)*, both of which have played vital roles in the design of the GNU Autotools. This chapter presents some fundamental tenets upon which the design of each of the Autotools is based. With these concepts, you'll better understand the theory behind the architectural decisions made by the Autotools designers.

In this chapter, we'll also design a simple project, Jupiter, from start to finish using hand-coded makefiles. We'll add to Jupiter in a stepwise

fashion as we discover functionality that we can use to simplify tasks and to provide features that open source software users have come to expect.

Chapters 4 and 5 present the framework designed by the GNU Autoconf engineers to ease the burden of creating and maintaining portable, functional project configuration scripts. The GNU Autoconf package provides the basis for creating complex configuration scripts with just a few lines of information provided by the project maintainer.

In these chapters, we'll quickly convert our hand-coded makefiles into Autoconf *Makefile.in* templates and then begin adding to them in order to gain some of the most significant Autoconf benefits. Chapter 4 discusses the basics of generating configuration scripts, while Chapter 5 moves on to more advanced Autoconf topics, features, and uses.

Chapter 6 begins by converting the Jupiter project *Makefile.in* templates into Automake *Makefile.am* files. Here, you'll discover that Automake is to makefiles what Autoconf is to configuration scripts. This chapter presents the major features of Automake in a manner that will not become outdated as new versions of Automake are released.

Chapters 7 and 8 explain the basic concepts behind shared libraries and show how to build shared libraries with Libtool—a standalone abstraction for shared-library functionality that can be used with the other Autotools. Chapter 7 begins with a shared-library primer and then covers some basic Libtool extensions that allow Libtool to be a drop-in replacement for the more basic library generation functionality provided by Automake. Chapter 8 covers library versioning and the runtime dynamic module management abstraction provided by Libtool.

Chapter 9 presents a relatively new addition to the Autotools—autotest. The autotest functionality in Autoconf allows you to easily create and manage integration test execution frameworks for your projects. In previous chapters, we will have covered unit testing in individual makefiles. Autotest provides a mechanism for adding more global testing that depends on multiple components in your project. Honestly, autotest can be used to do about any sort of testing you want. We'll focus on adding autotest suites that ensure your project works the way you believe it should—automatically.

Chapter 10 discusses the concepts of finding compile- and link-time dependencies and adding the appropriate references to build tool command lines. Specifically, this chapter introduces `pkg-config`, which has become a de facto standard in Linux software development, providing the framework for easily finding and consuming components that your package depends on. This chapter shows you how to both consume `pkg-config` *.pc* files to find your dependencies and how to play nicely in the sandbox by providing *.pc* files for your projects.

Chapters 11 and 12 discuss internationalization (abbreviated *i18n*) and localization (*l10n*), respectively—the ability to easily manage text strings and other locale-specific attributes (such as references to numbers, money, and dates) within your project that should be different for localized releases of your project.

Chapter 13 talks about obtaining maximum portability in your projects by using Gnulib.

Chapters 14 and 15 illustrate the transformation of an existing, fairly complex open source project (FLAIM) from using a hand-built build system to using an Autotools build system. This example will help you to understand how you might *autoconfiscate* one of your own existing projects.

Chapter 16 provides an overview of the features of the M4 macro processor that are relevant to obtaining a solid understanding of Autoconf. This chapter also considers the process of writing your own Autoconf macros.

Chapter 17 discusses using the Autotools to build software designed to run on Microsoft Windows platforms. I'll show you how to cross compile on Linux for Windows, and how to install and use the three most popular Windows-based POSIX platforms—Cygwin, Msys2, and MinGW—to build Windows software using GNU tools, including the Autotools.

Microsoft has a great set of free tools for building Windows software, but if your package is already working on Linux and being built with POSIX build tools, using the Autotools to build for Windows can be a great way to get you up and running there fast. From there, you can decide whether what you have is good enough for your project or if you need to provide a native build environment for Windows.

Chapter 18 is a compilation of tips, tricks, and reusable solutions to Autotools problems. The solutions in this chapter are presented as a set of individual topics or items. Each can be understood without context from the surrounding items.

Chapters 3 through 9 are built around the Jupiter project. Chapters 14 and 15 cover the FLAIM project. Chapters 11 and 12 cover the gettext project and Chapter 13 covers the b64 project. These projects are found on the GitHub NSP-Autotools site at *https://github.com/NSP-Autotools*.

Except for the FLAIM project, each of these repositories are tagged at commits representing topic transitions. The tags are called out within the margins of the chapters pertaining to these projects. You can easily follow along by checking out the tagged commit in the repository when you see one in the book.

Conventions Used in This Book

This book contains hundreds of program listings in roughly two categories: console examples and file listings. Console examples have no captions, and user input is **bolded**.

Often, I'll use the Linux `ls` command with various options to show the contents of a directory before or after changes are made. Different Linux distributions often ship with an alias for `ls` enabled by default. I'm using Linux Mint 18 with the Cinnamon desktop to write this book; my predefined alias for `ls` is:

```
$ alias
--snip--
alias ls='ls --color=auto'
$
```

You may find you have an `ls` alias on your system that provides a different default set of functionality that will defeat your attempts to exactly duplicate my console examples. Just be aware of the reasons for these possible differences.

File listings contain full or partial listings of the files discussed in the text. All named listings are provided in the associated git repositories. I've tried to provide enough context around modified portions of partial listings so that you can easily see where lines are added or changed. However, there are a few listings where lines are deleted. In these cases, I've called out the deleted lines in the text near the listing.

Listings without filenames are entirely contained in the printed listing itself, are meant to be considered independently without context, and are not part of the provided source repositories. In general, text that remains the same as a previously listed version of the file will be grayed out, whereas modified areas will be in black text.

For listings that do relate to the Jupiter and FLAIM projects, the caption first specifies the path of the file relative to the project root directory and then provides a description of the changes made to that file in the listing.

Throughout this book, I refer to the GNU/Linux operating system simply as *Linux*. It should be understood that by the use of the term *Linux*, I'm referring to GNU/Linux, its actual official name. I use *Linux* simply as shorthand for the official name.

Autotools Versions Used in This Book

The Autotools are always being updated—on average, a significant update of each of the three most important tools, Autoconf, Automake, and Libtool, is released every year and a half, and minor updates are released every three to six months. The Autotools developers attempt to maintain a reasonable level of backward compatibility with each new release, but occasionally something significant is broken, and older documentation simply becomes out-of-date. More recently, the Autotools have been considered mature and complete; release cycles have slowed and major changes seldom happen anymore. This is good for the community and for you, the reader, as it means that the material you find in this book will remain relevant for a long time to come.

Although I describe new, major features of recent releases of the Autotools, in my efforts to make this book more evergreen, I've tried to stick to descriptions of Autotools features (Autoconf macros, for instance) that have been in widespread use for several years. Minor details change occasionally, but the general use has stayed the same through many releases.

At appropriate places in the text, I mention the versions of the Autotools I used for this book, but I'll summarize here. I used version 2.69 of Autoconf, version 1.15 of Automake (the latest version as of this writing is actually 1.16.1), and version 2.4.6 of Libtool. Through the publication process, I was able to make minor corrections and update to new releases as they became available.

Additionally, I used version 0.19.7 of GNU gettext (the latest is 0.20.1) and version 0.29.1 of `pkg-config` (the latest is 0.29.2). The GNU portability library, Gnulib, is not distributed as a package but rather as a set of code snippets that are downloaded directly from the GNU website (*https://www.gnu.org/software/gnulib/*).