The AWK Programming Language

Second Edition

The AWK Programming Language

Second Edition

Alfred V. Aho

Brian W. Kernighan

Peter J. Weinberger

Addison-Wesley

Hoboken, New Jersey

Cover image: "Great Auk" by John James Audubon from *The Birds of America, Vols. I-IV,* 1827–1838, Archives & Special Collections, University of Pittsburgh Library System

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2023941419

This book was formatted by the authors in Times Roman, Courier and Helvetica, using Groff, Ghostscript and other open source Unix tools. See https://www.awk.dev.

ISBN-13: 978-0-13-826972-2
ISBN-10: 0-13-826972-6

$PrintCode

To the millions of Awk users

Contents

Preface

Awk was created in 1977 as a simple programming language for writing short programs that manipulate text and numbers with equal ease. It was meant as a *scripting language* to complement and work well with Unix tools, following the Unix philosophy of having each program do one thing well and be composable with other programs.

The computing world today is enormously different from what it was in 1977. Computers are thousands of times faster and have a million times as much memory. Software is different too, with a rich variety of programming languages and computing environments. The Internet has given us more data to process, and it comes from all over the world. We're no longer limited to the 26 letters of English either; thanks to Unicode, computers process the languages of the world in their native character sets.

Even though Awk is nearly 50 years old, and in spite of the great changes in computing, it's still widely used, a core Unix tool that's available on any Unix, Linux, or macOS system, and usually on Windows as well. There's nothing to download, no libraries or packages to import — just use it. It's an easy language to learn and you can do a lot after only a few minutes of study.

Scripting languages were rather new in 1977, and Awk was the first such language to be widely used. Other scripting languages complement or sometimes replace Awk. Perl, which dates from 1987, was an explicit reaction to some of the limitations of Awk at the time. Python, four years younger than Perl, is by far the most widely used scripting language today, and for most users would be the natural next step for larger programs, especially to take advantage of the huge number of libraries in the Python ecosystem. On the web, and also for some standalone uses, JavaScript is the scripting language of choice. Other more niche languages are still highly useful, and "the shell" itself has become a variety of different shells with significantly enriched programming capabilities.

Programmers and other computer users spend a lot of time doing simple, mechanical data manipulation — changing the format of data, checking its validity, finding items that have some property, adding up numbers, printing summaries, and the like. All of these jobs ought to be mechanized, but it's a real nuisance to have to write a special-purpose program in a language like C or Python each time such a task comes up.

Awk is a programming language that makes it possible to handle simple computations with short programs, often only one or two lines long. An Awk program is a sequence of patterns and actions that specify what to look for in the input data and what to do when it's found. Awk searches a set of files that contain text (but not non-text formats like Word documents, spreadsheets, PDFs and so on) for lines that match any of the patterns; when a matching line is found, the corresponding action is performed. A pattern can select lines by combinations of regular expressions and comparison operations on strings, numbers, fields, variables, and array elements. Actions may perform arbitrary processing on selected lines; the action language looks like C but there are no declarations, and strings and numbers are built-in data types.

Awk scans text input files and splits each input line into fields automatically. Because so many things are automatic — input, field splitting, storage management, initialization — Awk programs are usually much shorter than they would be in a more conventional language. Thus one common use of Awk is for the kind of data manipulation suggested above. Programs, a line or two long, are composed at the keyboard, run once, then discarded. In effect, Awk is a general-purpose programmable tool that can replace a host of specialized tools or programs.

The same brevity of expression and convenience of operations make Awk valuable for prototyping larger programs. Start with a few lines, then refine the program until it does the desired job, experimenting with designs by trying alternatives quickly. Since programs are short, it's easy to get started, and easy to start over when experience suggests a different direction. And if necessary, it's straightforward to translate an Awk program into another language once the design is right.

Organization of the Book

The goal of this book is to teach you what Awk is and how to use it effectively. Chapter 1 is a tutorial on how to get started; after reading even a few pages, you will have enough information to begin writing useful programs. The examples in this chapter are short and simple, typical of the interactive use of Awk.

The rest of the book contains a variety of examples, chosen to show the breadth of applicability of Awk and how to make good use of its facilities. Some of the programs are ones we use personally; others illustrate ideas but are not intended for production use; a few are included just because they are fun.

Chapter 2 shows Awk in action, with a number of small programs that are derived from the way that we use Awk for our own personal programming. The examples are probably too idiosyncratic to be directly useful, but they illustrate techniques and suggest potential applications.

Chapter 3 shows how Awk can be used for exploratory data analysis: examining a dataset to figure out its properties, identify potential (and real) errors, and generally get a grip on what it contains before expending further effort with other tools.

The emphasis in Chapter 4 is on retrieval, validation, transformation, and summarization of data — the tasks that Awk was originally designed for. There is also a discussion of how to handle data like address lists that naturally comes in multiline chunks.

Awk is a good language for managing small, personal databases. Chapter 5 discusses the generation of reports from databases, and builds a simple relational database system and query language for data stored in multiple files.

Chapter 6 describes programs for generating text, and some that help with document preparation. One of the examples is an indexing program based on the one we used for this book.

Chapter 7 is about "little languages," that is, specialized languages that focus on a narrow domain. Awk is convenient for writing small language processors because its basic operations support many of the lexical and symbol table tasks encountered in translation. The chapter includes an assembler, a graphics language, and several calculators.

Awk is a good language for expressing certain kinds of algorithms. Because there are no declarations and because storage management is easy, an Awk program has many of the advantages of pseudo-code but Awk programs can be run, which is not true of pseudo-code. Chapter 8 discusses experiments with algorithms, including testing and performance evaluation. It shows several sorting algorithms, and culminates in a version of the Unix `make` program.

Chapter 9 explains some of the historical reasons why Awk is as it is, and contains some performance measurements, including comparisons with other languages. The chapter also offers suggestions on what to do when Awk is too slow or too confining.

Appendix A, the reference manual, covers the Awk language in a systematic order. Although there are plenty of examples in the appendix, like most manuals it's long and a bit dry, so you will probably want to skim it on a first reading.

You should begin by reading Chapter 1 and trying some small examples of your own. Then read as far into each chapter as your interest takes you. The chapters are nearly independent of each other, so the order doesn't matter much. Take a quick look at the reference manual to get an overview, concentrating on the summaries and tables, but don't get bogged down in the details.

The Examples

There are several themes in the examples. The primary one, of course, is to show how to use Awk well. We have tried to include a wide variety of useful constructions, and we have stressed particular aspects like associative arrays and regular expressions that typify Awk programming.

A second theme is to show Awk's versatility. Awk programs have been used from databases to circuit design, from numerical analysis to graphics, from compilers to system administration, from a first language for non-programmers to the implementation language for software engineering courses. We hope that the diversity of applications illustrated in the book will suggest new possibilities to you as well.

A third theme is to show how common computing operations are done. The book contains a relational database system, an assembler and interpreter for a toy computer, a graph-drawing language, a recursive-descent parser for an Awk subset, a file-update program based on `make`, and many other examples. In each case, a short Awk program conveys the essence of how something works in a form that you can understand and play with.

We have also tried to illustrate a spectrum of ways to attack programming problems. Rapid prototyping is one approach that Awk supports well. A less obvious strategy is divide and conquer: breaking a big job into small components, each concentrating on one aspect of the problem. Another is writing programs that create other programs. Little languages define a good user interface and may suggest a sound implementation. Although these ideas are presented here in the context of Awk, they are much more generally applicable, and ought to be

part of every programmer's repertoire.

The examples have all been tested directly from the text, which is in machine-readable form. We have tried to make the programs error-free, but they do not defend against all possible invalid inputs, so we can concentrate on conveying the essential ideas.

Evolution of Awk

Awk was originally an experiment in generalizing the Unix tools `grep` and `sed` to deal with numbers as well as text. It was based on our interests in regular expressions and programmable editors. As an aside, the language is officially AWK (all caps) after the authors' initials, but that seems visually intrusive, so we've used Awk throughout for the name of the language, and `awk` for the name of the program. (Naming a language after its creators shows a certain paucity of imagination. In our defense, we didn't have a better idea, and by coincidence, at some point in the process we were in three adjacent offices in the order Aho, Weinberger, and Kernighan.)

Although Awk was meant for writing short programs, its combination of facilities soon attracted users who wrote significantly larger programs. These larger programs needed features that had not been part of the original implementation, so Awk was enhanced in a new version made available in 1985.

Since then, several independent implementations of Awk have been created, including Gawk (maintained and extended by Arnold Robbins), Mawk (by Michael Brennan), Busybox Awk (by Dmitry Zakharov), and a Go version (by Ben Hoyt). These differ in minor ways from the original and from each other but the core of the language is the same in all. There are also other books about Awk, notably *Effective Awk Programming*, by Arnold Robbins, which includes material on Gawk. The Gawk manual itself is online, and covers that version very carefully.

The POSIX standard for Awk is meant to define the language completely and precisely. It is not particularly up to date, however, and different implementations do not follow it exactly.

Awk is available as a standard installed program on Unix, Linux, and macOS, and can be used on Windows through WSL, the Windows Subsystem for Linux, or a package like Cygwin. You can also download it in binary or source form from a variety of web sites. The source code for the authors' version is at `https://github.com/onetrueawk/awk`. The web site `https://www.awk.dev` is devoted to Awk; it contains code for all the examples from the book, answers to selected exercises, further information, updates, and (inevitably) errata.

For the most part, Awk has not changed greatly over the years. Perhaps the most significant new feature is better support for Unicode: newer versions of Awk can now handle data encoded in UTF-8, the standard Unicode encoding of characters taken from any language. There is also support for input encoded as comma-separated values, like those produced by Excel and other programs. The command

```
$ awk --version
```

will tell you which version you are running. Regrettably, the default versions in common use are sometimes elderly, so if you want the latest and greatest, you may have to download and install your own.

Since Awk was developed under Unix, some of its features reflect capabilities found in Unix and Linux systems, including macOS; these features are used in some of our examples.

Furthermore, we assume the existence of standard Unix utilities, particularly `sort`, for which exact equivalents may not exist elsewhere. Aside from these limitations, however, Awk should be useful in any environment.

Awk is certainly not perfect; it has its full share of irregularities, omissions, and just plain bad ideas. But it's also a rich and versatile language, useful in a remarkable number of cases, and it's easy to learn. We hope you'll find it as valuable as we do.

Acknowledgments

We are grateful to friends and colleagues for valuable advice. In particular, Arnold Robbins has helped with the implementation of Awk for many years. For this edition of the book, he found errors, pointed out inadequate explanations and poor style in Awk code, and offered perceptive comments on nearly every page of several drafts of the manuscript. Similarly, Jon Bentley read multiple drafts and suggested many improvements, as he did with the first edition. Several major examples are based on Jon's original inspiration and his working code. We deeply appreciate their efforts.

Ben Hoyt provided insightful comments on the manuscript based on his experience implementing a version of Awk in Go. Nelson Beebe read the manuscript with his usual exceptional thoroughness and focus on portability issues. We also received valuable suggestions from Dick Sites and Ozan Yigit. Our editor, Greg Doench, was a great help in every aspect of shepherding the book through Addison-Wesley. We also thank Julie Nahil for her assistance with production.

Acknowledgments for the First Edition

We are deeply indebted to friends who made comments and suggestions on drafts of this book. We are particularly grateful to Jon Bentley, whose enthusiasm has been an inspiration for years. Jon contributed many ideas and programs derived from his experience using and teaching Awk; he also read several drafts with great care. Doug McIlroy also deserves special recognition; his peerless talent as a reader greatly improved the structure and content of the whole book. Others who made helpful comments on the manuscript include Susan Aho, Jaap Akkerhuis, Lorinda Cherry, Chris Fraser, Eric Grosse, Riccardo Gusella, Bob Herbst, Mark Kernighan, John Linderman, Bob Martin, Howard Moscovitz, Gerard Schmitt, Don Swartwout, Howard Trickey, Peter van Eijk, Chris Van Wyk, and Mihalis Yannakakis. We thank them all.

Alfred V. Aho
Brian W. Kernighan
Peter J. Weinberger