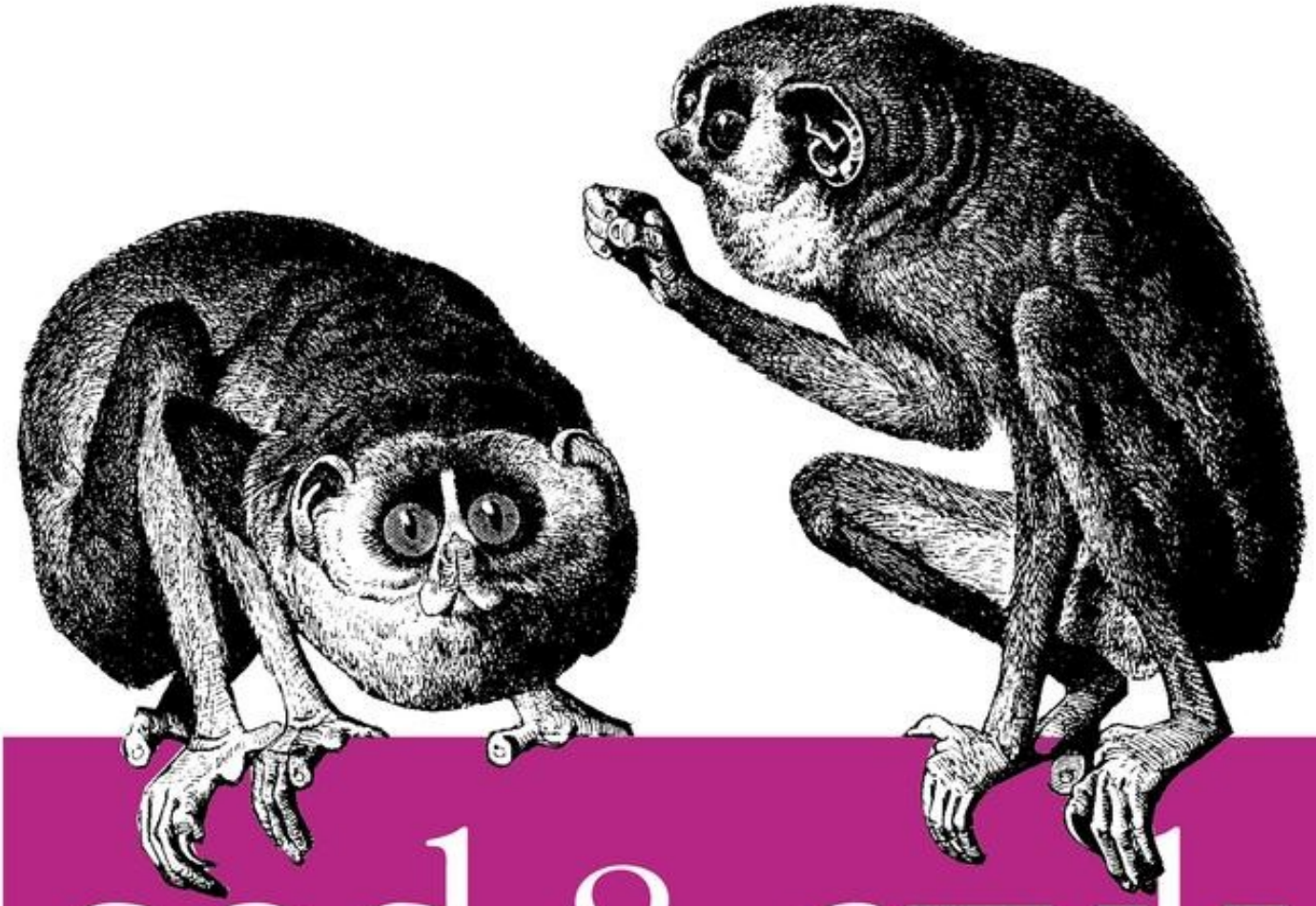
UNIX Power Tools

2nd Edition



sed & awk

O'REILLY®

Dale Dougherty & Arnold Robbins

sed & awk, 2nd Edition

Dale Dougherty

Arnold Robbins

Editor

Dale Dougherty

A Note Regarding Supplemental Files

Supplemental files and examples for this book can be found at http://examples.oreilly.com/9781565922259/. Please use a standard desktop web browser to access these files, as they may not be accessible from all ereader devices.

All code files or examples referenced in the book will be available online. For physical books that sh with an accompanying disc, whenever possible, we've posted all CD/DVD content. Note that while w provide as much of the media content as we are able via free download, we are sometimes limited by licensing restrictions. Please direct any questions or concerns to booktech@oreilly.com.

Dedication

To Miriam, for your love and patience.

—Arnold Robbins

Preface

This book is about a set of oddly named UNIX utilities, **sed** and **awk**. These utilities have many thing in common, including the use of regular expressions for pattern matching. Since pattern matching is such an important part of their use, this book explains UNIX regular expression syntax very thoroughly. Because there is a natural progression in learning from **grep** to sed to awk, we will be covering all three programs, although the focus is on sed and awk.

Sed and awk are tools used by users, programmers, and system administrators—anyone working with text files. Sed, so called because it is a stream editor, is perfect for applying a series of edits to a number of files. Awk, named after its developers Aho, Weinberger, and Kernighan, is a programming language that permits easy manipulation of structured data and the generation of formatted reports. This book emphasizes the POSIX definition of awk. In addition, the book briefly describes the origin version of awk, before discussing three freely available versions of awk and two commercial ones, al of which implement POSIX awk.

The focus of this book is on writing scripts for sed and awk that quickly solve an assortment of problems for the user. Many of these scripts could be called "quick-fixes." In addition, we'll cover scripts that solve larger problems that require more careful design and development.

Scope of This Handbook

Chapter 1, Power Tools for Editing, is an overview of the features and capabilities of sed and awk.

Chapter 2, Understanding Basic Operations, demonstrates the basic operations of sed and awk, showing a progression in functionality from sed to awk. Both share a similar command-line syntax, accepting user instructions in the form of a script.

Chapter 3, Understanding Regular Expression Syntax, describes UNIX regular expression syntax in full detail. New users are often intimidated by these strange expressions, used for pattern matching. I is important to master regular expression syntax to get the most from sed and awk. The pattern-matching examples in this chapter largely rely on **grep** and **egrep**.

Chapter 4, Writing sed Scripts, begins a three-chapter section on sed. This chapter covers the basic elements of writing a sed script using only a few sed commands. It also presents a shell script that simplifies invoking sed scripts.

Chapter 5, Basic sed Commands, and Chapter 6, Advanced sed Commands, divide the sed command set into basic and advanced commands. The basic commands are commands that parallel manual editing actions, while the advanced commands introduce simple programming capabilities. Among th advanced commands are those that manipulate the hold space, a set-aside temporary buffer.

Chapter 7, Writing Scripts for awk, begins a five-chapter section on awk. This chapter presents the primary features of this scripting language. A number of scripts are explained, including one that modifies the output of the **ls** command.

Chapter 8, Conditionals, Loops, and Arrays, describes how to use common programming constructs such as conditionals, loops, and arrays.

Chapter 9, Functions, describes how to use awk's built-in functions as well as how to write user-defined functions.

Chapter 10, The Bottom Drawer, covers a set of miscellaneous awk topics. It describes how to execut UNIX commands from an awk script and how to direct output to files and pipes. It then offers some (meager) advice on debugging awk scripts.

Chapter 11, A Flock of awks, describes the original V7 version of awk, the current Bell Labs awk, GNU awk (gawk) from the Free Software Foundation, and mawk, by Michael Brennan. The latter thr all have freely available source code. This chapter also describes two commercial implementations, MKS awk and Thomson Automation awk (**tawk**), as well as VSAwk, which brings awk-like capabilities to the Visual Basic environment.

Chapter 12, Full-Featured Applications, presents two longer, more complex awk scripts that together demonstrate nearly all the features of the language. The first script is an interactive spelling checker. The second script processes and formats the index for a book or a master index for a set of books.

Chapter 13, A Miscellany of Scripts, presents a number of user-contributed scripts that show differer styles and techniques of writing scripts for sed and awk.

Appendix A is a quick reference describing sed's commands and command-line options.

Appendix B is a quick reference to awk's command-line options and a full description of its scripting language.

Appendix C presents the full listings for the **spellcheck.awk** script and the **masterindex** shell script described in Chapter 12.

Availability of sed and awk

Sed and awk were part of Version 7 UNIX (also known as "V7," and "Seventh Edition") and have bee part of the standard distribution ever since. Sed has been unchanged since it was introduced.

The Free Software Foundation GNU project's version of sed is freely available, although not technically in the public domain. Source code for GNU sed is available via anonymous FTP[1] to the host *ftp.gnu.ai.mit.edu*. It is in the file *ftp://ftp.gnu.ai.mit.edu/pub/gnu/sed-2.05.tar.gz*. This is a tar fi compressed with the **gzip** program, whose source code is available in the same directory. There are many sites world-wide that "mirror" the files from the main GNU distribution site; if you know of on close to you, you should get the files from there. Be sure to use "binary" or "image" mode to transfer the file(s).

In 1985, the authors of awk extended the language, adding many useful features. Unfortunately, this new version remained inside AT&T for several years. It became part of UNIX System V as of Releas 3.1. It can be found under the name of nawk, for new awk; the older version still exists under its original name. This is still the case on System V Release 4 systems.

On commercial UNIX systems, such as those from Hewlett-Packard, Sun, IBM, Digital, and others, the naming situation is more complicated. All of these systems have some version of both old and ne awk, but what each vendor names each program varies. Some have **oawk** and **awk**, others have **awk** and **nawk**. The best advice we can give is to check your local documentation.[2] Throughout this book we use the term **awk** to describe POSIX awk. Specific implementations will be referred to by name, such as "gawk," or "the Bell Labs awk."

Chapter 11 discusses three freely available awks (including where to get them), as well as several commercial ones.

Note ✎

Since the first edition of this book, the awk language was standardized as part of the POSIX Command Language and Utilities Standard (P1003.2). All modern awk implementations aim to be upwardly compatible with the POSIX standard.

The standard incorporates features that originated in both new awk and gawk. In this book, you can assume that what is true for one implementation of POSIX awk is true for another, unless a particular version is designated.

DOS Versions

Gawk, mawk, and GNU sed have been ported to DOS. There are files on the main GNU distribution site with pointers to DOS versions of these programs. In addition, gawk has been ported to OS/2, VMS, and Atari and Amiga microcomputers, with ports to other systems (Macintosh, Windows) in progress.

egrep, sed, and awk are available for MS-DOS-based machines as part of the MKS Toolkit (Mortice Kern Systems, Inc., Ontario, Canada). Their implementation of awk supports the features of POSIX

awk.

The MKS Toolkit also includes the Korn shell, which means that many shell scripts written for the Bourne shell on UNIX systems can be run on a PC. While most users of the MKS Toolkit have probably already discovered these tools in UNIX, we hope that the benefits of these programs will be obvious to PC users who have not ventured into UNIX.

Thompson Automation Software[3] has an awk compiler for UNIX, DOS, and Microsoft Windows. This version is interesting because it has a number of extensions to the language, and it includes an awk debugger, written in awk!

We have used a PC on occasion because Ventura Publisher is a terrific formatting package. One of th reasons we like it is that we can continue to use **vi** to create and edit the text files and use sed for writing editing scripts. We have used sed to write conversion programs that translate **troff** macros in Ventura stylesheet tags. We have also used it to insert tags in batch mode. This can save having to manually tag repeated elements in a file.

Sed and awk are also useful for writing conversion programs that handle different file formats.

Other Sources of Information About sed and awk

For a long time, the main source of information on these utilities was two articles contained in Volume 2 of the *UNIX Programmer's Guide*. The article *awk—A Pattern Scanning and Processing Language* (September 1, 1978) was written by the language's three authors. In 10 pages, it offers a brief tutorial and discusses several design and implementation issues. The article *SED—A Non-Interactive Text Editor* (August 15, 1978) was written by Lee E. McMahon. It is a reference that give a full description of each function and includes some useful examples (using Coleridge's *Xanadu* as sample input).

In trade books, the most significant treatment of sed and awk appears in *The UNIX Programming Environment* by Brian W. Kernighan and Rob Pike (Prentice-Hall, 1984). The chapter entitled "Filter not only explains how these programs work but shows how they can work together to build useful applications.

The authors of awk collaborated on a book describing the enhanced version: *The AWK Programming Language* (Addison-Wesley, 1988). It contains many full examples and demonstrates the broad range of areas where awk can be applied. It follows in the style of the *UNIX Programming Environment*, which at times makes it too dense for some readers who are new users. The source code for the example programs in the book can be found in the directory ftp://netlib.bell-labs.com/netlib/research/awkbookcode on *netlib.bell-labs.com*.

The IEEE Standard for Information and Technology Portable Operating System Interface (POSIX) Part 2: Shell and Utilities (Standard 1003.2-1992)[4] describes both sed and awk.[5] It is the "official" word on the features available for portable shell programs that use sed and awk. Since awk is a programming language in its own right, it is also the official word on portable awk programs.

In 1996, the Free Software Foundation published *The GNU Awk User's Guide*, by Arnold Robbins. This is the documentation for gawk, written in a more tutorial style than the Aho, Kernighan, and Weinberger book. It has two full chapters of examples, and covers POSIX awk. This book is also published by SSC under the title *Effective AWK Programming*, and the Texinfo source for the book comes with the gawk distribution.

It is one of the current deficiencies of GNU sed that it has no documentation of its own, not even a manpage.

Most general introductions to UNIX introduce sed and awk in a long parade of utilities. Of these books, Henry McGilton and Rachel Morgan's *Introducing the UNIX System* offers the best treatment of basic editing skills, including use of all UNIX text editors.

UNIX Text Processing (Hayden Books, 1987), by the original author of this handbook and Tim O'Reilly, covers sed and awk in full, although we did not include the new version of awk. Readers of that book will find some parts duplicated in this book, but in general a different approach has been taken here. Whereas in the textbook we treat sed and awk separately, expecting only advanced users t tackle awk, here we try to present both programs in relation to one another. They are different tools that can be used individually or together to provide interesting opportunities for text processing.

Finally, in 1995 the Usenet newsgroup *comp.lang.awk* came into being. If you can't find what you nee to know in one of the above books, you can post a question in the newsgroup, with a good chance that someone will be able to help you.

The newsgroup also has a "frequently asked questions" (FAQ) article that is posted regularly. Besides answering questions about awk, the FAQ lists many sites where you can obtain binaries of different versions of awk for different systems. You can retrieve the FAQ via FTP in the file called ftp://rtfm.mit.edu/pub/usenet/comp.lang.awk/faq from the host *rtfm.mit.edu*.

Sample Programs

The sample programs in this book were originally written and tested on a Mac IIci running A/UX 2.0 (UNIX System V Release 2) and a SparcStation 1 running SunOS 4.0. Programs requiring POSIX awk were re-tested using gawk 3.0.0 as well as the August 1994 version of the Bell Labs awk from the Bel Labs FTP site (see Chapter 11 for the FTP details). Sed programs were retested with the SunOS 4.1.3 sed and GNU sed 2.05.

---

[1] If you don't have Internet access and wish to get a copy of GNU sed, contact the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 U.S.A. The telephone number is 1-617-542-5942, and the fax number is 1-617-542-2652.

[2] Purists refer to the new awk simply as **awk**; the new one was intended to replace the original one. Alas, almost 10 years after it was released, this still has not really happened.

[3] 5616 SW Jefferson, Portland, OR 97221 U.S.A., 1-800-944-0139 within the U.S., 1-503-224-1639 elsewhere.

[4] Whew! Say *that* three times fast!

[5] The standard is not available online. It can be ordered from the IEEE by calling 1-800-678-IEEE(4333) in the U.S. and Canada, 1-908-981-0060 elsewhere. Or, see http://www.ieee.org/ from a Web browser. The cost is U.S. $228, which includes Standard 1003.2d-1994—Amendment 1 for Batc Environments. Members of IEEE and/or IEEE societies receive a discount.

Obtaining Example Source Code

You can obtain the source code for the programs presented in this book from O'Reilly & Associates through their Internet server. The example programs in this book are available electronically in a number of ways: by FTP, Ftpmail, BITFTP, and UUCP. The cheapest, fastest, and easiest ways are listed first. If you read from the top down, the first one that works for you is probably the best. Use FTP if you are directly on the Internet. Use Ftpmail if you are not on the Internet, but can send and receive electronic mail to Internet sites (this includes CompuServe users). Use BITFTP if you can send electronic mail via BITNET. Use UUCP if none of the above works.

FTP

To use FTP, you need a machine with direct access to the Internet. A sample session is shown, with what you should type in boldface.

```
$ ftp ftp.oreilly.com
Connected to ftp.oreilly.com.
220 FTP server (Version 6.21 Tue Mar 10 22:09:55 EST 1992) ready.
Name (ftp.oreilly.com:yourname): anonymous
331 Guest login ok, send domain style e-mail address as password.
Password: yourname@domain.name    (Use your user name and host here)
230 Guest login ok, access restrictions apply.
ftp> cd /published/oreilly/nutshell/sedawk_2
250 CWD command successful.
ftp> binary    (Very important! You must specify binary transfer for compressed files.)
200 Type set to I.
ftp> get progs.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for progs.tar.gz.
226 Transfer complete.
ftp> quit
221 Goodbye.
```

The file is a **gzip** compressed **tar** archive; extract the files from the archive by typing:

```
$ gzcat progs.tar.gz | tar xvf -
```

System V systems require the following **tar** command instead:

```
$ gzcat progs.tar.gz | tar xof -
```

If **gzcat** is not available on your system, use separate **gunzip** and **tar** commands.

```
$ gunzip progs.tar.gz
$ tar xvf progs.tar
```

Ftpmail

Ftpmail is a mail server available to anyone who can send electronic mail to and receive it from Internet sites. This includes any company or service provider that allows email connections to the Internet. Here's how you do it. You send mail to ftpmail@online.oreilly.com. In the message body, give the FTP commands you want to run. The server will run anonymous FTP for you and mail the files back to you. To get a complete help file, send a message with no subject and the single word

"help" in the body. The following is a sample mail session that should get you the examples. This command sends you a listing of the files in the selected directory and the requested example files. Th listing is useful if there's a later version of the examples you're interested in.

```
$ mail ftpmail@online.oreilly.com
Subject:
reply-to yourname@domain.name       (Where you want files mailed)
open
cd /published/oreilly/nutshell/sedawk_2
dir
mode binary
uuencode
get progs.tar.gz
quit
.
```

A signature at the end of the message is acceptable as long as it appears after "quit."

BITFTP

BITFTP is a mail server for BITNET users. You send it electronic mail messages requesting files, an it sends you back the files by electronic mail. BITFTP currently serves only users who send it mail from nodes that are directly on BITNET, EARN, or NetNorth. To use BITFTP, send mail containing your **ftp** commands to BITFTP@PUCC. For a complete help file, send **HELP** as the message body. The following is the message body you send to BITFTP:

```
FTP  ftp.oreilly.com  NETDATA
USER  anonymous
PASS  yourname@yourhost.edu    Put your Internet email address here (not your BITNET address)
CD /published/oreilly/nutshell/sedawk_2
DIR
BINARY
GET  progs.tar.gz
QUIT
```

Once you've got the desired file, follow the directions under FTP to extract the files from the archive Since you are probably not on a UNIX system, you may need to get versions of **uudecode**, **gunzip**, **atob**, and **tar** for your system. VMS, DOS, and Mac versions are available.

UUCP

UUCP is standard on virtually all UNIX systems and is available for IBM-compatible PCs and Apple Macintoshes. The examples are available by UUCP via modem from UUNET; UUNET's connect-tim charges apply. If you or your company has an account with UUNET, you have a system somewhere with a direct UUCP connection to UUNET. Find that system, and type:

```
uucp uunet\!~/published/oreilly/nutshell/sedawk_2/progs.tar.gz yourhost\!~/yourname/
```

The backslashes can be omitted if you use a Bourne-style shell (**sh**, **ksh**, **bash**, **zsh**, **pdksh**) instead of **csh** or **tcsh**. The file should appear some time later (up to a day or more) in the directory */usr/spool/uucppublic/yourname*. If you don't have an account, but would like one so that you can get electronic mail, contact UUNET at 703-206-5400. It's a good idea to get the file */published/oreilly/ls lR.Z* as a short test file containing the filenames and sizes of all the files available. Once you've got

the desired file, follow the directions under FTP to extract the files from the archive.

Conventions Used in This Handbook

The following conventions are used in this book:

Bold

is used for statements and functions, identifiers, and program names.

Italic

is used for file and directory names when they appear in the body of a paragraph as well as for da types and to emphasize new terms and concepts when they are introduced.

`Constant Width`

is used in examples to show the contents of files or the output from commands.

`Constant Bold`

is used in examples to show command lines and options that should be typed literally by the user. (For example, **`rm foo`** means to type "rm foo" exactly as it appears in the text or the example.)

""

are used to identify a code fragment in explanatory text. System messages and symbols are quote as well.

$

is the UNIX Bourne shell or Korn shell prompt.

[]

surrounds optional elements in a description of program syntax. (The brackets themselves should never be typed, unless otherwise noted.)

...

stands for text (usually computer output) that's been omitted for clarity or to save space.

□

indicates a literal space. This symbol is used to make spaces visible in examples, as well as in the text.

•

indicates a literal TAB character. This symbol is used to make tabs visible in examples, as well a in the text.

The notation CTRL-X or ^X indicates use of control characters. It means hold down the "control" key while typing the character "x". We denote other keys similarly (e.g., RETURN indicates a carriage return). All examples of command lines are followed by a RETURN unless otherwise indicated.

About the Second Edition

Since this book was first published in 1990, it has become one of the most fundamental of the O'Reil & Associates Nutshell Handbooks. Three important events occurred after it was written. The first wa the publication of the POSIX standard for sed, and more importantly for awk. The second (perhaps du to the first) was the widespread availability of some version or other of new awk on all modern UNIX systems, both commercial ones and the freely available UNIX-like systems such as NetBSD, FreeBSD, and Linux. The third was the source code availability of GNU sed, and three versions of awk, instead of just gawk.

For these and other reasons, O'Reilly & Associates decided that this handbook needed to be updated. The goals of the revision were to keep the flavor of the book intact ("if it ain't broke, don't fix it"), reorient the awk part of the book around POSIX awk, correct mistakes, and bring the book up to date.

I would like to thank Gigi Estabrook, Chris Reilley, and Lenny Muellner of O'Reilly & Associates fo their help, Marc Vauclair, the French translator of the first edition, for many helpful comments, and John Dzubera for his comments on the first edition. Michael Brennan, Henry Spencer, and Ozan Yigi acted as technical reviewers for this edition, and I would like to thank them for their input. Ozan Yig in particular, deserves extra thanks for forcing me to be very rigorous in my testing. Pat Thompson o Thompson Automation Software graciously provided an evaluation copy of **tawk** for review in this book. Richard Montgomery of Videosoft provided me with information about VSAwk.

The following people provided the scripts in Chapter 13: Jon L. Bentley, Tom Christiansen, Geoff Clare, Roger A. Cornelius, Rahul Dhesi, Nick Holloway, Norman Joseph, Wes Morgan, Tom Van Raalte, and Martin Weitzel. Their contributions are gratefully acknowledged.

Thanks also to the staff at O'Reilly & Associates. Nicole Gipson Arigo was the production editor and project manager. David Sewell was the copyeditor, and Clairemarie Fisher O'Leary proofread the book. Jane Ellin and Sheryl Avruch performed quality control checks. Seth Maislin wrote the index. Erik Ray, Ellen Siever, and Lenny Muellner worked with the tools to create the book. Chris Reilley fine-tuned the figures. Nancy Priest and Mary Jane Walsh designed the interior book layout, and Edi Freedman designed the front cover.

My in-laws, Marshall and Elaine Hartholz of Seattle, deserve special thanks for taking our children camping for a week, allowing me to make significant progress during an important phase of the update. ☺

Finally, I would like to thank my wonderful wife Miriam for her patience during this project.

Arnold Robbins

Acknowledgments from the First Edition

To say that this book has been long anticipated is no understatement. I published three articles on aw in *UNIX/World* in the spring and summer of 1987, making the mistake of saying that these articles were from the upcoming Nutshell Handbook, *Sed & Awk*. I proposed to Tim O'Reilly that I adapt the articles and create a book as a project I could work on at home shortly after the birth of my son, Benjamin. I thought I'd finish it in several months. Well, my son turned three around the time I was completing the first draft. Cathy Brennan and the customer service representatives have been patient handling requests for the book ever since the *UNIX/World* articles appeared. Cathy said that she even had people call to order the book, swearing it was available because they knew other people who had read it. I owe a debt of gratitude to her and her staff and to the readers I've kept waiting.

My thanks to Tim O'Reilly for creating a great company in which one can easily get sidetracked by a number of interesting projects. As editor, he pushed me to complete the book but would not allow it t be complete without his writing all over it. As usual, his suggestions made me work to improve the book.

Thanks to all the writers and production editors at O'Reilly & Associates, who presented interesting problems to be solved with sed and awk. Thanks to Ellie Cutler who was the production editor for the book and also wrote the index. Thanks to Lenny Muellner for allowing me to quote him throughout th book. Thanks as well to Sue Willing and Donna Woonteiler for their efforts in getting the book into print. Thanks to Chris Reilley who did the illustrations. Thanks to the individual contributors of the sed and awk scripts in Chapter 13. Thanks also to Kevin C. Castner, Tim Irvin, Mark Schalz, Alex Humez, Glenn Saito, Geoff Hagel, Tony Hurson, Jerry Peek, Mike Tiller, and Lenny Muellner, who sent me mail pointing out typos and errors.

Finally, dearest thanks to Nancy and Katie, Ben and Glenda.

Dale Dougherty

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
1-800-998-9938 (in the U.S. or Canada)
1-707-829-0515 (international or local)
1-707-829-0104 (FAX)

You can send us messages electronically. To be put on the mailing list or request a catalog, send ema
to:

info@oreilly.com

To ask technical questions or comment on the book, send email to:

bookquestions@oreilly.com

Chapter 1. Power Tools for Editing

My wife won't let me buy a power saw. She is afraid of an accident if I use one. So I rely on a hand saw for a variety of weekend projects like building shelves. However, if I made my living as a carpenter, I would have to use a power saw. The speed and efficiency provided by power tools would be essential to being productive. [D.D.]

For people who create and modify text files, sed and awk are power tools for editing. Most of the things that you can do with these programs can be done interactively with a text editor. However, using sed and awk can save many hours of repetitive work in achieving the same result.

Sed and awk are peculiar and it takes time to learn them, but the capabilities they provide can repay the learning many times over, especially if text editing is a normal part of your trade.

May You Solve Interesting Problems

The primary motivation for learning sed and awk is that they are useful for devising general solution to text editing problems.[1] For some people, myself included, the satisfaction of solving a problem is the difference between work and drudgery. Given the choice of using **vi** or sed to make a series of repeated edits over a number of files, I will choose sed, simply because it makes the problem more interesting to me. I am refining a solution instead of repeating a series of keystrokes. Besides, once I accomplish my task, I congratulate myself on being clever. I feel like I have done a little bit of magic and spared myself some dull labor.

Initially, using sed and awk will seem like the long way to accomplish a task. After several attempts you may conclude that the task would have been easier to do manually. Be patient. You not only have to learn how to use sed and awk but you also need to learn to recognize situations where using them pays off. As you become more proficient, you will solve problems more quickly and solve a broader range of problems.

You will also begin to see opportunities to find general solutions to specific problems. There is a way of looking at a problem so you see it related to a class of problems. Then you can devise a solution that can be reused in other situations.

Let me give you an example (without showing any program code). One of our books used a cross-referencing naming scheme in which the reference was defined and processed by our formatting software (**sqtroff**). In the text file, a reference to a chapter on error handling might be coded as follows:

```
\*[CHerrorhand]
```

"CHerrorhand" is the name giving the reference and "*[" and "]" are calling sequences that distinguish the reference from other text. In a central file, the names used for cross references in the document are defined as **sqtroff** strings. For instance, "CHerrorhand" is defined to be "Chapter 16, Error Handling." (The advantage of using a symbolic cross-referencing scheme like this, instead of explicit referencing, is that if chapters are added or deleted or reordered, only the central file needs to be edited to reflect the new organization.) When the formatting software processes the document, the references are properly resolved and expanded.

The problem we faced was that we had to use the same files to create an online version of the book.

Because our **sqtroff** formatting software would not be used, we needed some way to expand the cross references in the files. In other words, we did not want files containing "*[CHerrorhand]"; instead w wanted what "CHerrorhand" referred to.

There were three possible ways to solve this problem:

1. Use a text editor to search for all references and replace each of them with the appropriate litera string.
2. Use sed to make the edits. This is similar to making the edits manually, only faster.
3. Use awk to write a program that (a) reads the central file to make a list of reference names and their definitions, (b) reads the document searching for the reference calling sequence, and (c) looks up the name of the reference on the list and replaces it with its definition.

The first method is obviously time-consuming (and not very interesting!). The second method, using sed, has an advantage in that it creates a tool to do the job. It is pretty simple to write a sed script tha looks for "*[CHerrorhand]" and replaces it with "Chapter 16, Error Handling" for instance. The same script can be used to modify each of the files for the document. The disadvantage is that the substitutions are hard-coded; that is, for each cross reference, you need to write a command that makes the replacement. The third method, using awk, builds a tool that works for *any* cross reference that follows this syntax. This script could be used to expand cross references in other books as well. I spares you from having to compile a list of specific substitutions. It is the most general solution of th three and designed for the greatest possible reuse as a tool.

Part of solving a problem is knowing which tool to build. There are times when a sed script is a bette choice because the problem does not lend itself to, or demand, a more complex awk script. You have to keep in mind what kinds of applications are best suited for sed and awk.

---

[1] I suppose this section title is a combination of the ancient Chinese curse "May you live in interesting times" and what Tim O'Reilly once said to me, that someone will solve a problem if he finds the problem interesting. [D.D.]

A Stream Editor

Sed is a "non-interactive" stream-oriented editor. It is stream-oriented because, like many UNIX programs, input flows through the program and is directed to standard output. (**vi**, for instance, is not stream-oriented. Nor are most DOS applications.) Input typically comes from a file but can be directed from the keyboard.[2] Output goes to the terminal screen by default but can be captured in a file instead. Sed works by interpreting a script specifying the actions to be performed.

Sed offers capabilities that seem a natural extension of interactive text editing. For instance, it offers search-and-replace facility that can be applied globally to a single file or a group of files. While you would not typically use sed to change a term that appears once in a particular file, you will find it ver useful to make a series of changes across a number of files. Think about making 20 different edits in over 100 files in a matter of minutes, and you get an idea of how powerful sed can be.

Using sed is similar to writing simple shell scripts (or batch files in DOS). You specify a series of actions to be performed in sequence. Most of these actions could be done manually from within **vi**: replacing text, deleting lines, inserting new text, etc. The advantage of sed is that you can specify all editing instructions in one place and then execute them on a single pass through the file. You don't have to go into each file to make each change. Sed can also be used effectively to edit very large files that would be slow to edit interactively.

There are many opportunities to use sed in the course of creating and maintaining a document, especially when the document consists of individual chapters, each placed in a separate file. Typicall after a draft of a document has returned from review, there are a number of changes that can be applied to all files. For instance, during the course of a software documentation project, the name of the software or its components might change, and you have to track down and make these changes. With sed, this is a simple process.

Sed can be used to achieve consistency throughout a document. You can search for all the different ways a particular term might be used and make them all the same. You can use sed to insert special typesetting codes or symbols prior to formatting by **troff**. For instance, it can be used to replace quotation marks with the ASCII character codes for forward and back double quotes ("curly quotes" instead of "straight" quotes).

Sed also has the ability to be used as an editing *filter*. In other words, you could process an input file and send the output to another program. For instance, you could use sed to analyze a plain text file ar insert **troff** macros before directing the output to **troff** for formatting. It allows you to make edits on the fly, perhaps ones that are temporary.

An author or publisher can use sed to write numerous conversion programs—translating formatting codes in Scribe or files into **troff**, for example, or converting PC word processing files, such as WordStar. Later on, we will look at a sed script that converts **troff** macros into stylesheet tags for use in Ventura Publisher. (Perhaps sed could be used to translate a program written in the syntax of one language to the syntax of another language.) When Sun Microsystems first produced Xview, they released a conversion program for converting SunView programs to XView, and the program largely consisted of sed scripts, converting the names of various functions.

Sed has a few rudimentary programming constructs that can be used to build more complicated scripts. It also has a limited ability to work on more than one line at a time.

All but the simplest sed scripts are usually invoked from a "shell wrapper," a shell script that invokes sed and also contains the commands that sed executes. A shell wrapper is an easy way to name and execute a single-word command. Users of the command don't even need to know that sed is being used. One example of such a shell wrapper is the script **phrase**, which we'll look at later in this book. It allows you to match a pattern that might fall over two lines, addressing a specific limitation of **grep**.

In summary, use sed:

1. To automate editing actions to be performed on one or more files.
2. To simplify the task of performing the same edits on multiple files.
3. To write conversion programs.

---

[2] Doing so, however, is not particularly useful.

A Pattern-Matching Programming Language

Identifying awk as a programming language scares some people away from it. If you are one of them consider awk a different approach to problem solving, one in which you have a lot more control over what you want the computer to do.

Sed is easily seen as the flip side of interactive editing. A sed procedure corresponds closely enough how you would apply the editing commands manually. Sed limits you to the methods you use in a tex editor. Awk offers a more general computational model for processing a file.

A typical example of an awk program is one that transforms data into a formatted report. The data might be a log file generated by a UNIX program such as **uucp**, and the report might summarize the data in a format useful to a system administrator. Another example is a data processing application consisting of separate data entry and data retrieval programs. Data entry is the process of recording data in a structured way. Data retrieval is the process of extracting data from a file and generating a report.

The key to all of these operations is that the data has some kind of structure. Let us illustrate this wit the analogy of a bureau. A bureau consists of multiple drawers, and each drawer has a certain set of contents: socks in one drawer, underwear in another, and sweaters in a third drawer. Sometimes drawers have compartments allowing different kinds of things to be stored together. These are all structures that determine where things go—when you are sorting the laundry—and where things can be found—when you are getting dressed. Awk allows you to use the structure of a text file in writing the procedures for putting things in and taking things out.

Thus, the benefits of awk are best realized when the data has some kind of structure. A text file can b loosely or tightly structured. A chapter containing major and minor sections has some structure. We' look at a script that extracts section headings and numbers them to produce an outline. A table consisting of tab-separated items in columns might be considered very structured. You could use an awk script to reorder columns of data, or even change columns into rows and rows into columns.

Like sed scripts, awk scripts are typically invoked by means of a shell wrapper. This is a shell script that usually contains the command line that invokes awk as well as the script that awk interprets. Simple one-line awk scripts can be entered from the command line.

Some of the things awk allows you to do are:

- View a text file as a textual database made up of records and fields.
- Use variables to manipulate the database.
- Use arithmetic and string operators.
- Use common programming constructs such as loops and conditionals.
- Generate formatted reports.
- Define functions.
- Execute UNIX commands from a script.
- Process the result of UNIX commands.
- Process command-line arguments more gracefully.

- Work more easily with multiple input streams.

Because of these features, awk has the power and range that users might rely upon to do the kinds of tasks performed by shell scripts. In this book, you'll see examples of a menu-based command generator, an interactive spelling checker, and an index processing program, all of which use the features outlined above.

The capabilities of awk extend the idea of text editing into computation, making it possible to perfor a variety of data processing tasks, including analysis, extraction, and reporting of data. These are, indeed, the most common uses of awk but there are also many unusual applications: awk has been used to write a Lisp interpreter and even a compiler!

Four Hurdles to Mastering sed and awk

There are a number of introductory UNIX books that will acquaint you with sed and awk. The goal of this book is to take you much further—to help you *master* sed and awk and to reduce the amount of time and effort that it takes you to reach that goal.

There are four hurdles on the way to mastering sed and awk. You must learn:

1. *How to use sed and awk.* This is a relatively low hurdle to clear because, fortunately, sed and aw work in a very similar manner, based on the line editor **ed**. Chapter 2, covers the mechanics of using sed and awk.
2. *To apply UNIX regular expression syntax.* Using UNIX regular expression syntax for pattern matching is common to both sed and awk, and many other UNIX programs. This can be a difficult hurdle for two reasons: the syntax is arcane, and though many people have some experience using regular expressions, few have persevered to master the complete syntax. The more facile you are in using this syntax, the easier sed and awk are to use. That is why we spend good deal of time covering regular expressions in Chapter 3.
3. *How to interact with the shell.* While not directly related to sed and awk themselves, managing the interaction with the command shell is often a frustrating problem, since the shell shares a number of special characters with both programs. If you can, avoid the problem by putting your script in a separate file. If not, use a Bourne-compatible shell for your scripts (the quoting rules are more straightforward), and use single quotes to contain your script. If you are using **csh** as your interactive shell, remember to escape any exclamation points with a backslash ("\!"). There is no other way to get **csh** to leave the exclamation point alone.[3]
4. *The knack of script writing.* This is the most difficult, rather like a series of high hurdles. Becau of this, the bulk of the book is devoted to script writing. With sed, you have to learn a set of single-letter commands. With awk, you have to learn the statements of a programming language To get the knack of script writing, though, you simply must pore over lots of examples and, of course, must try your hand at writing scripts yourself.

If you were running the high hurdles, the ability to clear the hurdles does not win the race—clearing them swiftly does. In writing scripts, learning the scripting command set or language is simply clearing the hurdle. Acquiring the ability to attack interesting problems with your scripts is running fast enough to compete.

---

[3] Well, you can set the `histchars` variable. See the **csh** man page.

Chapter 2. Understanding Basic Operations

If you are starting out to learn sed and awk, you can benefit from looking at how much they have in common.

- They are invoked using similar syntax.
- They are both stream-oriented, reading input from text files one line at a time and directing the result to standard output.
- They use regular expressions for pattern matching.
- They allow the user to specify instructions in a script.

One reason they have so much in common is that their origins can be found in the same line editor, **e** In this chapter, we begin by taking a brief look at **ed** and show how sed and awk were logical steps towards the creation of a programmable editor.

Where sed and awk differ is in the kind of instructions that control the work they do. Make no mistak —this is a major difference, and it affects the kinds of tasks that can best be performed with these programs.

This chapter looks at the command-line syntax of sed and awk and the basic structure of scripts. It also offers a tutorial, using a mailing list, that will give you a taste of script writing. It is valuable to see sed and awk scripts side-by-side before you concentrate on either one of them.

Awk, by Sed and Grep, out of Ed

You can trace the lineage of awk to sed and **grep**, and through those two programs to **ed**, the original UNIX line editor.

Have you ever used a line editor? If so, it will be much easier for you to understand the line orientation of sed and awk. If you have used **vi**, a full-screen editor, then you are familiar with a number of commands that are derived from its underlying line editor, **ex** (which in turn is a superset of the features in **ed**).

Let's look at some basic operations using the line editor **ed**. Don't worry—this is an exercise intended to help you learn sed and awk, not an attempt to convince you of the wonders of line editors. The **ed** commands that are shown in this exercise are identical to the **sed** commands you'll learn later on. Fee free to experiment with **ed** on your own to get a sense of how it works. (If you're already familiar wit **ed**, feel free to skip to the next section.)

To use a line editor, you work on one line at a time. It is important to know what line you are positioned at in the file. When you open a file using **ed**, it displays the number of characters in the fil and positions you at the last line.

```
$ ed test
339
```

There is no prompt. If you enter a command that **ed** does not understand, it prints a question mark as an error message. You can enter the print command, **p**, to display the current line.

```
p
```