Perspectives in Technical Writing

Technical writers share their experiences

Table of Contents

Introduction

Technically We Write is a community of technical writers, technical editors, copyeditors, web content writers, and all other roles in technical communication.

We want to celebrate all forms of technical and professional writing. Our readers have a wide range of interests, from writing tools and technologies, editing, usability, web SEO, and anything else you can put under the "technical and professional writing" umbrella.

We are fortunate to have such a talented and engaged community who share their tips, best practices, and "how-tos" about technical writing. We wanted to share a selection of the most popular articles from Technically We Write in this book, as our way to thank and recognize everyone for what they do.

Perspectives on technical writing

Technically We Write is an open community about technical communication. One of our first community members is Seth Kenlon, who recently contributed articles about 5 things you didn't know about styles in your word processor and Tables and data structure. I asked Seth about his approach to writing, and about his work in technical writing.

Let's start with an introduction.

I'm Seth Kenlon. I tinker with open source technology, and I often write about what I learn.

You've written a ton of technical articles about all kinds of topics, including shell scripting, programming, and Linux desktops. What is your motivation for writing about these topics?

My greatest motivation is my own interest in a subject. I tend to pay special attention to what I'm "distracted" by in life. That distraction is often the thing I'm most passionate about, whether I recognize that passion yet or not. Whenever possible, I follow most of my distractions to their logical end. Sometimes that takes years; I got distracted from film by Linux nearly years ago, and exploring it still is as fun as it was the first time I booted it. Other times, it produces a fun programming project or a seemingly random item of curiosity like an album of synth music, a card game, or rules for a wargame.

Whatever form a distraction takes, documenting the steps to reproduce it is a great way to reinforce what I've learned along the way, and to help others follow that distraction for themselves.

When someone's paying me to document something very specific, I have to focus on a challenge. My usual challenge is to explain a concept in the most efficient but effective way possible. The perfect technical document leaves no room for interpretation, is a pleasure to read, and is only as long as absolutely required. That's the challenge I try to meet any time I'm hired to write tech documents.

What is your process for writing an article?

Even when I know a topic well, research is always the first step. At the very least, I verify what I think I know about the topic. Are there different ways to perform a specific task? Is it effective to document every possible way to perform a task, or is it better to demonstrate a specific workflow? Or do I need two documents: one for workflow and one for complete documentation of every available feature?

In addition to that kind of thinking, I sometimes have to learn how to do the thing I'm writing about, too. That's tricky, because when you're the one documenting something for the first time, there are likely no tips on the Internet yet to tell you how it works. You have to talk to the programmer or designer, or you have to read the code, or you just have to figure it out.

I like outlines, but I admit I don't always write them down. When I do, they usually take the form of section headings with a sentence or two about what the section demonstrates.

I rarely change my first draft, because my real first draft was the outline. I do a lot of editing as I go, and I do an edit after the document is complete, but the end result is usually just a cleaner version of the first draft. No matter what, though, I always appreciate a good set of eyes to review my work before it goes live. A good editor is a real asset.

I'm referring purely to technical writing. Stuff I write for fun, like blog posts about my favorite tabletop games or movies, never gets that much forethought or review.

You've also worked in film. How did you get your start?

I was on a subway on the way to a gig, and I was reading a film journal to kill time. I recall the journal interviewed Phil Tippett and Joe Letteri, and they were talking about the render farm they'd built for a recent movie, probably the original Jurassic Park or Toy Story. They said it had changed the way they worked, because with Unix you could have lots of computers access the same file to contribute to a render.

They were talking about a Beowulf cluster or something like that, but it was magical and utterly mystifying to me. I couldn't get it out of my head. I was captivated. I wanted to make a computer work without opening a video editor or 3D modeler.

Seven years later, I was working for Joe Letteri at Weta Digital as a Linux specialist.

What are your favorite tools and/or technologies for writing?

Currently, I prefer writing in AsciiDoc with GNU Emacs. AsciiDoc is a sort of scripting language for Docbook XML, which I enjoy and trust. Emacs allows for a lot of customization, and it's really flexible in how you can use it.

I find that AsciiDoc succeeds in all the ways Markdown fails, except in the way it looks. Markdown looks like "normal" text, and I think it feels very natural to read it. AsciiDoc looks like structured text, and sometimes is strange for casual reading.

For instance, this is a URL in Markdown:

```
Go to my [website](http:// example.com)
```

This is a URL in AsciiDoc:

```
Go to my http://example.com[website]
```

Markdown expresses the concept in exactly the way you'd write or say it normally, except that it insists that you use brackets and parentheses in a predictable manner. AsciiDoc bizarrely inverts the way you're likely to read the statement, and feels less structured for the lack of markup.

Your journey in technical writing

One aspect I love about the website is I get to interview lots of interesting people about what they do in technical communication. Thanks to AmyJune Hineline for this interview about her journey in technical writing.

Let's start with an introduction. What is your background in technical writing?

I'm AmyJune Hineline, a technical community advocate in the open source space and hospice nurse by trade.

My background in documentation really started while in nursing and having to take patient notes. Care notes need to be detailed and concise so the next nurse could understand and perform a treatment.

In the technology sector, I started contributing to the Drupal project's documentation in the form of READMEs in the code repositories. I also helped maintain several external documentation guides for contributed modules.

What is your role at Drupal?

My role in Drupal has evolved from being a contributor to now leading the Mentoring Initiative. I organize many local and regional camps and organize and lead workshops on how to contribute back to the project - there is space to collaborate and contribute no matter your role, skill set, or passion.

I also am on the Community Working Group's Community Health team where we work on tools and language around positive and collaborative communication.

How can folks use Drupal to manage a website?

Drupal is very robust, accessible out of the box, and allows for dynamic content, all while being open source. It integrates so well with so many other web technologies:

APIs, content marketing tools, SEO, and so on. If you need something beyond the core code, chances are there's a module for it!

How do you approach technical writing on the web?

Technical writing on the web is different because your content consumer's attention span is shorter, they tend to scroll, scan, and read headings. Perhaps they are on their phone and the viewport is smaller, so economy of words is important.

Accessibility is very important for technical writing. How can authors create accessible content?

A few recommendations:

- Use clear concise words and phrases
- Use familiar language and avoid buzzwords, cultural references, and jargon
- Break up content into digestible chunks, use headings and then bullet points
- Define acronyms, numeronyms, abbreviations upon introduction
- Use predefined reading levels. I recommend a 9th grade level.
- Use clear fonts and avoid special characters and emoticons. Think about low vision and folks who use screen readers.
- Add alternative text to anything visual, such as charts, images, tables, screenshots
- Use code blocks instead of images of code so folks can copy and paste

What is your favorite tool for technical writing?

I use BBEdit, a bare bones editor, and tend to write in markdown before I enter the content into my Git interface or Drupal editor.

WordStar for professional writing

WordStar was a popular desktop word processor, originally published by MicroPro for the CP/M operating system in the late 1970s. WordStar was later released for MS-DOS in the early 1980s where it gained the height of its popularity, noted for its ease of use and advanced formatting.

While discontinued in the early 1990s, WordStar is still popular in 2024 for many professional writers. Science-fiction author Robert J. Sawyer is one such user, who recently released a repackaged version of WordStar 7, the last version of WordStar for MS-DOS. We asked him about how he uses WordStar, and what makes WordStar a great word processor:

Let's start with an introduction: Who are you and what do you do?

I'm a Hugo and Nebula Award-winning science-fiction writer living just outside Toronto. I suppose I'm best known for the *FlashForward* TV series on ABC, which was based on my novel of the same name, and for which I was one of the scriptwriters. My most recent book, my 25th, is *The Downloaded*, which is available for free to Audible subscribers and is also available in print and ebook forms.

What version of WordStar do you use, and how do you run it?

I started with WordStar 2.26 on an Osborne 1 CP/M computer in December 1983, and I updated as each new version came along. The final version, WordStar 7.0, is what I've been using since 1992. Since WordStar was first released in 1978, fourteen years earlier, I'll point out that WordStar 7.0 has been the *current* version of WordStar for thirty-two years now. Although WordStar 4.0, which some users cling to (hi, George!), was certainly a wonderful program, WordStar 7.0 is so much more sophisticated. I run WordStar under DOSBox-X (*not* plain DOSBox, which is aimed mostly at gaming) on a Windows 11 Pro computer, specifically a Framework 13 laptop, with two external monitors and an ergonomic keyboard attached.

What makes WordStar a great word processor?

I outline this in depth on my website.

But to just mention one significant thing: WordStar is ideal for touch-typists. You can do everything—*everything*—without taking your hands off the home typing row. In Microsoft Word, even something as quotidian as backspacing to correct a mistyped character requires breaking the flow of typing by forcing you to lift your right hand from the home-typing row to hit the backspace key; in WordStar, backspacing is a simple home-typing-row command that doesn't break the creative rhythm at all.

How easily can you customize what WordStar can do?

WordStar's interface is very customizable, much more so than is Word's. I prefer a clean screen, so I have just a single line at the top of the screen for WordStar status messages—the very specific messages I want to see—and a single dim character at the far right showing me my relative position in the document. WordStar had the ability, if you wanted, to have a night mode *decades* before Word offered that; I write with cyan letters on a black background, and I have WordStar fill the *entire* screen: no Windows crap visible *at all*. It's a totally distraction-free setup.

How do you share WordStar files with others?

WordStar 7.0 comes with a program that can convert its files to Word .doc format, as well as .rtf, which can be read by any modern word processor. But I'm *very* picky about manuscript formatting and layout, so ages ago I created my own system for conversion, using WordStar macros, to produce `.rtf` files that are perfect, better, in fact than what you'd get using Word; type "Election '24" in Word, and you'll get a single open quote at the beginning of the abbreviated year instead of an apostrophe—madness! I then just load the `.rtf` file into Word and save it as `.docx`. In fact, I wrote the answers to this interview in WordStar but used the system I just described to produce the Word file I've sent you containing them.

What are some things that are easy to do in WordStar that are more difficult to do in another word processor?

Here's a simple one. Mark a block of text. See that in the middle of the block, you've used the word "inane" instead of "insane"—both good words for describing the Microsoft Word interface—and move your cursor to add the missing s. In WordStar you get the logical result: the block stays marked, and the word is changed, and you can go do whatever the heck else you wish to do. But in Word, the moment you type that s, *all the text*—possibly thousands of words!—is *gone*. And God forbid in Word that you should mark a block and then decide *not* to deal with it right away. WordStar lets me do operations in *any order I wish*, not in a prescribed sequence of steps that *must* be carried out in exact order, as you have to do in Word.

And, of course, WordStar lets you see the formatting codes (or hide them; WordStar *invented* reveal codes, a feature WordPerfect later adopted but Word never got), and, in WordStar, you can treat formatting codes as you would any other part of the document. Try to block mark a font designation or a color designation in Word and then move it or copy it somewhere else. It simply can't be done in Bill Gates's monstrosity, but it's effortless in WordStar.

When you write your books in WordStar, how do you format your documents?

WordStar actually has *three* kinds of formatting codes: simple ASCII control codes for toggling bold or italics, or putting in a nonbreak space, and so forth; dot commands (lines that begin with a dot, or period, are treated as commands in WordStar), and formatting tags for things like style sheets, font choices, footnotes, and more. And I use them *all*.

The dot commands can be very powerful; they can contain math equations or conditional statements—so you can, for instance, edit on an eight-inch ruler line but print out on a 6.5-inch one; try *that* in Word—or, with a dot command, convert all the footnotes in a document to endnotes, and so much more.

I use the ruler dot command to set margins and tabs so much more quickly than you can do it in Word. You want paragraphs indented half an inch, the right margin at three inches, and tabs at, say, one and one-and-a-half inch? I can just type a line that does that:

```
L----P----!----!------------R
```

Want to change any aspect of that? Simply edit the ruler-line dot command as you would edit any text. Want to convert a regular tab to one that lines up on the decimal point instead? Just overtype the `!` with an #. You can do all that in seconds just by typing, rather than going through a complex multitabbed dialog box, as you must in Word.

But, yes, my most-used dot command is the "`..`" command, which lets me insert a comment to myself. And, of course, if you consider the dot commands a visual distraction, just hide them until you want to see them again.

By the way, although obviously the creations I've made with WordStar that I'm most proud of are my twenty-five novels, I'll point out that my website, with over a million words of text spread over 800 documents, was entirely hand-coded in HTML with WordStar. Please give it a visit at **sfwriter.com**.

Technical writing in project management

Everyone does technical and professional writing, no matter your role in an organization. For example, project managers need to write project plans, status updates, and other forms of documentation. We asked a project manager about how good written communication plays a part in their role:

Let's start with an introduction: Who are you, and what do you do?

My name is Tim Eiler. Though my career started with work at NASA (training astronauts and flight/mission controllers, to be specific) I have been managing projects and teams of project managers for the past 26 years.

I'm a mechanical and industrial engineer by education, and I also have an MBA. I think this gives me a really unique perspective on solving problems in a business setting. I've often said that I built my career on standing at the intersection of technology and business and that I can speak both languages as needed. My industrial engineering background, with its focus on process improvement and quality, really provides the linch pin for that!

The largest project I've managed had a $150 million budget, with about 200 people working on it, and was projected to take 36 months. The people were spread across five different sites, mostly in the US, but also in Europe. The smallest had about three people working on it, a budget of about $35 thousand, and lasted about two months.

The project management office (PMO) teams I've managed in healthcare, finance, and other verticals, have ranged in size from five to fifteen project managers. In addition to the daily management of project work, this has included working directly with organization leaders to select and maintain the constituent project components of a portfolio of active projects. Of course, it required writing a lot of process documentation, too.

I grew up in a very small town (population 400) in rural Minnesota. I think this directly influences my work on a daily basis. I work hard and expect others to do the same. I am fairly direct, but in true Scandinavian descendant fashion, I work to soften that directness so as to avoid driving a wedge between others and me by causing hurt feelings.

Everyone does technical and professional writing. What are some typical things you write all the time in your role as project manager?

There is, certainly, the process documentation I mentioned earlier. Documentation like that includes instructions about what department policies are regarding how specific aspects of each project are to be handled - both “happy path” and deviations therefrom. It also includes written standard project processes and procedures.

As a project manager (PM) working directly on managing individual projects, though, I typically find I write project proposals, project scope statements, project requirements documents, project plans - both prose and Gantt-type or other “schedules” - risk management plans, project budgets, organizational change management (OCM) plans, and technical change requests, to name a few. As PM on larger projects, I only contribute to some of those documents, rather than writing them outright myself. The OCM plan and project requirements documents are specific examples of those, as they are often written by an OCM professional and business analyst (BA), respectively.

Then, of course, there are the ubiquitous status reports that need to be distributed regularly. These are perhaps the most difficult of the documents to tackle. They undoubtedly have multiple types of audiences, from C-level executives to individual contributors on the project team itself. Satisfying all of those types of people in just a short document is very tricky.

What’s a type of document you write all the time?

I write status reports of various types all the time. Many people think the purpose is to convey data about what the team has accomplished and is planning to accomplish. That’s only the “data” part, though. It’s the PM’s job to help leaders do their jobs. The purpose of a status report is simple: To let decision-makers (the project sponsor, for example) determine whether they need to intervene in some way to help keep the project moving at the needed velocity.

That difference may not seem like much until you're a PM writing a status report.

How important is it for a project manager to write well? What does it mean to "write well"?

It's incredibly important that a PM write well. That's less about whether that PM can write perfectly according to Strunk & White, though. "Well" in the PM's world is about being able to convey the right, needed, key information simply. Not that a PM can get away with terrible grammar and punctuation, mind you! Generally speaking, though, so long as the work is intelligible and usable, a PM can get away with it. Now, that said, the better PMs - the ones who most frequently get the promotions, for instance - write both incredibly useful and timely content and write that content in clear language that typically follows good writing standards of punctuation and grammar.

In terms of "writing a project plan," every company's format and content for a project plan will differ. So "good" is not so much about how a PM "fills out the template" as it is about how well written the content is.

Project schedules, all by themselves, don't really have a lot of writing involved. However, I find that it's incredibly important for a PM to write the titles of the tasks involved in the schedule extremely clearly. Sloppy formation of the task title is a sure way to get what you didn't want. Writing task titles typically in verb-noun or noun-verb structure most often is what I use. Many PMs write simply in the "noun alone" form. That doesn't tell the person who'll perform the task what's to be done to that noun.

In terms of other parts of an overall project, such as the risk management plan, communication, and such, the same holds true. How one writes the structure of risks, action steps, and such, defines how those things are measured and managed. If the PM doesn't write those things well, the structures of how they're implemented will be less than ideal and the outcomes will be negatively affected.

Most importantly, many parts of a project plan are not meant for the PM, but for other members of the project team. Write using PM jargon, or just badly in general, and the PM will confuse the other document users - or worse.

What other writing and communication skills does a good project manager need to have?

The first point I want to make here is that there is a difference between data and information. Sure, a PM has to ensure that the necessary data about the project is available, but a truly good PM will be excellent at summarizing the key parts of that data into information that a receiver of that information can use to make choices, decisions, etc. So, that data and information are the key tools in a PM's toolkit.

Communication from a PM's perspective needs to be incredibly concise. A lot of communication today favors tools like Slack and Discord over email, and while "PowerPoint" presentations are still a thing, they seem to be headed toward the eventual dustbin. No matter what medium is used, I use a "tool" called SBAR - Situation, Background, Analysis, Recommendation. It puts the *need* or *expectation* up front and then provides the supporting information in the back.

Most of the project documentation I do is created using a tool like Microsoft Word. Tracking is done in Word, Excel, or a more-specialized application.

A lot of planning, including initiation schedule planning is done in tools like Miro or Mural. A lot of PM communication isn't terribly "formal." It *is* pragmatic, though!

What should someone know if they want to become a project manager?

I've often said that project management is really just codified common sense. The tools used are pretty simple because of that. A word processor (like Microsoft Word), a spreadsheet (such as Microsoft Excel), a flowcharting application (like Microsoft Visio): All of those are basic and, though a new PM wouldn't be expected to be a wizard-level user, solid skills are expected.

Scheduling is a ubiquitous part of a PM's role, so it's critical to have a reasonable level of basic skills with one or more scheduling tools. Such tools might include packages like Microsoft Project, Monday, SmartSheet, and any one of a dozen others. (In the 1990s, I probably would have said hundreds!)

Communication is 90% of a good PM's job. Projects are a team sport, after all! The other tools above serve to provide the key info to communicate to the overall project team and those impacted by the project. Were a PM to simply create those documents, budgets, schedules, etc., and then not share information about what they "say," it would be a wasted effort to have created them.

Technical writing and technical editing

Damon Garn spent about 20 years in the classroom, delivering technical training on Windows Server, Linux, security and CompTIA products. He also spent time as a network administrator for U.S. Figure Skating. In recent years, he left the training industry and now provides freelance writing and editing services for companies. I asked Damon about his role as a technical writer and editor.

Let's start with an introduction: Who are you and what do you do?

I'm Damon Garn, owner of Cogspinner Coaction, LLC. My organization provides technical writing and editing in the Information Technology field. I spent over twenty years as a technical trainer, delivering courses on Windows and Linux servers, security, and configuration management. I also spent time in the IT workforce as a network administrator.

I decided to translate my technical knowledge and writing skills into a thriving business that develops learning materials to educate future IT practitioners.

Is it just technical writing, or do you work on other projects for clients?

My primary technical writing opportunities are instructional guides for computer-industry certifications. I've written five Official Courseware guides for CompTIA, with a sixth planned for later in 2024. My role is not just authoring but also course design, translating exam objectives into a functional and logical outline, and advising on hands-on opportunities related to the content.

My business also offers tutorial-style articles covering server configuration on Linux and Windows, logging, security, and more. I also write many conceptual articles that explain the whys rather than the hows. I really enjoy this aspect of writing. I'm able to suggest new areas and hone new skills.

What does it take to be a good technical writer? Do technical writers have to be very "technical"?

I don't believe technical writers have to be experts in the topic. They certainly need some context within the field. For example, a beginner programmer could write a "how to learn programming" article, but they must still anticipate the challenges new developers face and possess some basic vocabulary terms. That said, research is a big part of writing prior to the authoring stage. Another crucial skill is project management.

However, I have two other observations about good technical writers.

The first is an ability to translate complex and role-specific terms and concepts into everyday language the reader can easily absorb. The writer is a guide and advocate for the reader, not the content. I honed this skill as a technical trainer and translated it into written communication.

Second, the writer must know when a visual aid is a better delivery tool than the written word. Screenshots, recorded demonstrations (I use many animated GIFs), and diagrams are all critical to IT professionals. I believe that many successful IT people are visual and such images help them grasp concepts. Technical writers can cater to this audience.

Are there any downsides to your technical writing career?

I've dabbled in writing fiction for most of my life. I even have two novel drafts around the 80,000-word mark (one is high fantasy, and the other is steampunk). However, once my daily job involved technical writing, I could no longer find the creative energy to sit in front of the computer and invent fictional worlds.

I really hope that changes someday! I hate that I've had to sacrifice creative writing for technical authoring.

You're also a technical editor. What does an editor do?

If you Google for types of editors, you'll find nice, neat matrices and job descriptions. Organizations don't necessarily structure their teams so rigidly in the real world. This becomes especially true when the company consolidates the content team, as is happening often right now.

My primary editor role ended up being very restricted. I served as a freelance technical editor for an organization for nearly four years. I was brought on for my technical knowledge and experience with Linux and given significant leeway to organize and structure article content. By the time that contract ended at the start of 2024, I was a glorified spell-checker. Other editors had taken on the technical role and were uninterested in input.

In the early years of that project, I enjoyed the ability to work on behalf of the reader to help the author develop a piece that communicated their knowledge in a way a less familiar reader could follow. I think the editor's job is to hone the edge the author has created.

What does a typical day or week look like for you as an editor?

I'm a believer in productive time. In college, I was most creative and productive at night. I often wrote papers and conducted research in the 6:00 pm to 2:00 am timeframe. Today, that productive time is reversed, and I am most efficient and energetic from about 5:00 am until 1:00 pm. I tend to do my editing as soon as I get up (I'm between editing gigs at the moment). I would rise, pour a cup of coffee, and open the articles I needed to work on. That process got me warmed up for my own writing assignments, which I would do from mid-morning until the end of my workday.

What skills does it take to be a good technical editor?

I believe a good technical editor (as opposed to a good copy editor) needs to see the big picture. What's the author's overall goal? Is the article structured in a way that communicates that goal effectively? The other role of a technical editor is the ability to comprehend the technical aspects of the project. For example, I often caught command typos in the articles I reviewed or noticed code snippets containing mistakes.

Copy editor skills are critical and no less important. I see this job as more detail-oriented, focusing on the proper use of commas, eliminating extra spaces between sentences, etc.

Both jobs are essential and may or may not be completed by the same person.

What tools or technologies do you use as an editor?

I use a variety of tools. I do a surprising amount of drafting in Vim, a Linux-based text editor. Vim eliminates the distractions more robust programs like Microsoft Word inflict. Obviously, Microsoft Word (and LibreOffice Writer) is essential. I don't tend to like track changes, and often, by the time a piece gets to me for editing, I apply the final changes. I do appreciate comments in Word and Google Docs.

I'm really interested in the use of Git for content authors. It's clearly a critical tool for developers but it has also become a valid utility for content creators. I haven't moved my writing process to Git yet, but I'm strongly considering it.

Here's a fun question: Do you have any editing pet peeve?

My editing pet peeves are pretty simple. I'll mention two.

1. Authors who cannot properly spell their own project. I saw this all the time with Red Hat OpenShift. An author might submit a piece with the sentence, "I work on the Openshift team…" and I'd have to correct the word to OpenShift. C'mon, it's *your* project!
2. I am also driven to distraction by two spaces after a period before the start of a new sentence. I learned to type on a traditional typewriter and was taught then to include two spaces. That convention is long gone.

Celebrating LibreOffice at 40 years

LibreOffice is a powerful open source office suite that supports all platforms: Windows, Mac, and Linux. LibreOffice in one form or another has been around since 1984, so it just turned 40 years old in 2024. We interviewed Italo Vignoli to learn more about LibreOffice and to celebrate this milestone anniversary:

Let's start with an introduction: Who are you and what do you do at LibreOffice?

I'm one of the founders of the LibreOffice project, the only Italian, after contributing to the OpenOffice project for the previous seven years. I've been a business communications professional since 1981, so I've always been involved in marketing, communications and media relations for both the OpenOffice and LibreOffice projects. I am also a spokesperson for the project and help organise LibreOffice's presence at FOSDEM and the LibreOffice Conference.

How did LibreOffice get started?

The roots of LibreOffice are lost in history. The first version of StarWriter dates back to 1984, developed by a German student: Marco Boerries. The great success of the software was the basis for the subsequent birth of StarOffice and the company StarDivision, based in Hamburg. StarOffice was a proprietary product, very similar to today's open source office suite.

StarOffice was a German product, although it was also available in English. Because of its strong German roots, it achieved a significant market share in Germany, second only to Microsoft Office. In my opinion, one of the reasons for its success was the fact that it was not a Microsoft clone, but an office suite with its own characteristics. It was also available for Linux from 1996, which made it very popular in the open source environment.

StarDivision was acquired by Sun in 1999. In 2000, the StarOffice source code was donated to the community and the OpenOffice project was born. For almost ten years, OpenOffice continued to gain market share as the main competitor to Microsoft Office. In 2010, following Oracle's acquisition of Sun, the community decided to take over development of the software and forked the project, creating LibreOffice and its home: The Document Foundation.

Why are people moving to LibreOffice? What draws them?

The first reason people switch from Microsoft Office to LibreOffice is that it is free. For the majority of users who have paid to use Microsoft Office, this is the main difference. Of course, there are many other benefits, but you have to use the software to discover them. For example, in addition to being open and standard (whereas Microsoft's OOXML is standard but proprietary), the native document format is more robust and much more secure.

By the way, ODF turned twenty years old on 1 May 2025. Quite an achievement.

Once you start using LibreOffice, if you're not blocked by resistance to change and you're consistent enough in learning its functions, then all the strengths of the open source office suite emerge: the flexible user interface, the optimised use of available screen space, and the robustness.

What makes LibreOffice unique is the LibreOffice Technology Platform, the only one on the market that allows the consistent development of desktop, mobile and cloud versions - including those provided by companies in the ecosystem - capable of producing identical and fully interoperable documents based on the two available ISO standards: the open ODF or Open Document Format (ODT, ODS and ODP) and the proprietary Microsoft OOXML (DOCX, XLSX and PPTX).

LibreOffice is the best office suite for users who want to retain control over their individual software and documents, thereby protecting their privacy and digital life from the commercial interference and the lock-in strategies of Big Tech.

Who are the typical users of LibreOffice?

LibreOffice is downloaded anonymously, as we don't ask for names, email addresses or other personal information, so it's very difficult to build up a profile of users. It is possible to know a little more from the emails sent to support, as some users provide some information to help us solve their problems, but it's too small a sample to make reliable assumptions.

LibreOffice certainly has a large number of individual users, many of whom have had problems with Microsoft Office, but this is fairly obvious. LibreOffice also has a large number of users in public administration, especially in countries where there is a greater focus on digital sovereignty and privacy. Finally, LibreOffice is used in many universities, again in those universities where there is a greater focus on digital sovereignty and privacy.

Among the public administrations using LibreOffice are the region of Schleswig-Holstein in Germany, several ministries in France, the Ministry of Defence in Italy, the Government of Taiwan, some large cities such as Valencia in Spain, Bari and Bologna in Italy, and many smaller organisations.

What features do you love about LibreOffice?

I am a heavy user of LibreOffice Writer to write articles and press releases, as well as longer documents such as white papers and e-books. I always use the native LibreOffice document format, which is open, standard and extremely robust, and thanks to this I have never lost a single document. I use styles for almost everything, plus advanced page formatting features such as the automatic table of contents.

I am also a heavy user of LibreOffice Impress to prepare and deliver my presentations. It is a good presenter tool that avoids the often useless bells and whistles of PowerPoint to make life easier for the presenter.

And I can easily exchange documents with the rest of the world using both ISO standard ODF and proprietary Microsoft formats. Of course, I am very careful to use fonts that are installed on all PCs, or their open source equivalents, to avoid the document being screwed up just because the font is missing on the receiving PC. I also use LibreOffice Draw to open PDF files and make small changes.

What's next for LibreOffice?

There are two major anniversaries in 2025: ODF, LibreOffice's native document standard format, will be 20 years old, and LibreOffice will be 15 years old. If we think about all the problems we have overcome in that time, these are two major achievements. ODF, although not as widely adopted as it should be, is recognised as the only standard format, and LibreOffice is still alive and growing, despite all the attempts to kill it by various companies using all kinds of strategies over the years.

In terms of functionality, of course, we are looking very carefully at integration with artificial intelligence tools and the development of desktop-level collaboration features, as well as improving existing features to keep up with the market-leading software.

How to publish a book

As technical writers, the written word is pretty important. And while we can create documents and other material in a variety of online formats, sometimes the printed word is the way to go. For example, training workbooks are best delivered as printed material.

These days, there are a lot of ways to generate printed documents. For small runs that will be distributed in-house, a multifunction copier and a three-ring binder might do the job. But a fully bound book can deliver a more professional look. We reached out to one printer that specializes in these custom print jobs to ask what people create and how to prepare content for print.

Let's start with an introduction. Who are you, and what do you do?

My name is Paul. I'm the Senior Content Manager at Lulu.com. That means I oversee the content we create, from blog posts and website pages to social media, emails, videos, and podcasts.

While I'm responsible for all the written content Lulu produces, I focus on writing long-form content for our blog.

Lulu is one of the world's biggest independent self-publishing and print-on-demand providers. Founded in 2002, Lulu came about after Red Hat CEO Bob Young wrote his own book. He was struggling to find a publisher who would pick it up. So he self-published with a vanity press—costing him a ton of money and leaving him with boxes of unsold books. We still have bookcases full of those books in the Lulu offices.

After that experience, Bob founded Lulu to make publishing accessible to anyone.

Since 2002, we've been focused on print-on-demand. Lulu offers other publishing options (like ebooks and retail distribution to sell books on sites like Amazon), but we've stayed with print as our main product.

As we've grown and the publishing world has changed, Lulu shifted from making print-on-demand accessible to facilitating various alternative methods for selling books. Lulu Direct is our ecommerce business line, featuring connections to popular ecommerce services (like Shopify, Wix, and WooCommerce) so creators can sell books directly to their fans.

What is the process to self-publish on Lulu?

Publishing on Lulu follows a pretty similar process to all on-demand printers. You'll need to have a file prepared and formatted. That means you've written, edited, reviewed, revised, and *fully* designed your book's files before coming to Lulu.

This step is a challenge for many new authors and creators because it requires learning the ins and outs of page layout. How to add a running header or design for full-bleed printing is not a skill most of us naturally develop. Once a creator does get over the learning curve for file design (or they hire a professional designer to do it for them), publishing with Lulu is really easy.

You upload that file, enter all the metadata for your book (like the title, ISBN, and description), and finally, make a cover with our designer or upload your own cover design.

A few things I think anyone self-publishing (whether on Lulu or elsewhere) need to know:

1. Finish writing the book before you start looking into publishing options. This is a huge area where new authors and creators tend to mess up. Don't worry about publishing until you've finished writing the book!
2. Once your book is done and you're into the editing phase, pick out a self-publisher (Lulu, obviously) and start learning their process. That might mean watching videos, reading guides, or uploading a test project (grab some files from Gutenberg or just make your own blank files). Most importantly, get your hands on some templates (for interior pages and the cover) and design around the specs. Building your files without thinking about the printing specifications will (almost) always lead to frustration and more work.
3. Review, revise, and review again. I don't think anyone has ever self-published a book without something that could be fixed. Thankfully, POD makes this process

easier and less expensive—you can simply order a copy of your own book, review it thoroughly, and once you're done with the edits, upload a new PDF file.

Lulu is, at our core, a 100% DIY platform. We print books, ship books, and facilitate your selling those books.

We do not have editorial standards, nor do we directly edit books published using Lulu. Decisions about how to edit and design your book rest entirely with you.

That said, we do have a page featuring our preferred and vetted partners. This page includes several professional editors, cover designers, and marketing services.

What things can someone print through Lulu? For example, every year, my wife and I collect twelve great photos of our cats and we print a calendar. I've also thought about printing a custom day planner as a special giveaway.

I do a pet calendar every year too!

And you're right that 'books' often mean something different than most of us imagine. When I think of a 'book,' my mind goes to a bookstore or a library—a place filled with novels and reference materials.

But the majority of content Lulu prints doesn't really fit that definition. We print a *lot* of notebooks, planners, guides, journals, workbooks, and any other kind of book that gets used rather than read.

Lulu, by and large, doesn't print a lot of fiction.

We print all those 'low content' books I mentioned above, as well as cookbooks, guides, marketing books, and 'thought leader' books. Over the last ten years, experts and entrepreneurs have published differently, and we've adapted our printing options to meet those needs.

Realistically, no one is competing with Amazon for fiction authors. It's just too embedded. But creators who aren't reliant on big retailers are a large (and growing) group. I'm talking about bloggers, podcasters, artists, photographers, influencers, and anyone with a dedicated, niche audience.

They're having the same thoughts you are: adding a custom day planner to your business is a terrific (and low-cost) way to delight your clients. Most creators even go a step further, giving away copies to their clients and offering print-on-demand copies to anyone visiting their website.

What things does an author need to consider when preparing a book interior?

We've spent years creating content to answer that exact question. For anyone who wants an in-depth, detailed answer with downloadable guides, videos, and FAQs, we provide several Guides and Templates.

For this interview, I'll give you a high-level view of what must be done before a file is ready to publish on Lulu.

1. Writing - I mentioned this earlier, but the first step is writing the book. Even if we're talking about a journal or planner, you've got to start with any text for your book. Write, edit, revise, and finalize.

2. Formatting - You have to make four choices about your book before you start formatting:

1. Size - The final size of your printed book (like A5, US Trade, and Letter). I strongly recommend taking a tape measure to a bookstore and looking at similar books in your genre to make this decision. Also, think about your readers' expectations. If you're a math teacher, your readers are probably students expecting a hardcover workbook, and if you're a science fiction novelist, your fans expect a trade paperback.
2. Ink - Color or black & white, both premium and standard.
    - Standard B&W, for novels or any book using only text
    - Premium B&W, for books with graphs, tables, or charts
    - Standard Color, for books with a few images
    - Premium Color, for photo books and art books
3. Paper - Lulu offers 60#, 80#, and 100# paper for a variety of our books. This is usually a simple choice: 60# for standard inks, and 80# for premium inks.
4. Binding - Most authors publish paperbacks, but you can also select hardcover, coil-bound, or saddle stitch. The latter two are for notebooks, workbooks, coloring books, magazines, and comics.

Okay, so once you've decided on your formatting, you'll need a template from your publisher. Lulu's is best downloaded from our Calculator (so you can precisely design your book and see the print costs before downloading the template).

This is the hard part. You'll need some design skills and software like Adobe InDesign, Affinity Publisher, etc. Please don't use Microsoft Word or Google Docs to format your print files. It can be done, but you're so limited there's a great chance you won't be happy with the results.

Designing involves setting fonts, styling your text (for chapter titles, quotes, etc.), and adding the front and back matter (like copyright pages, about the author, an index, etc.). It's a lot to get into here, but if you plan to self-publish and are unsure about your page layout skills, you need to start learning.

There are some services out there that offer quick formatting. One that comes to mind is Atticus. There, you upload a mostly unformatted manuscript, select fonts, styles, header/footer content, and the like, and they export a PDF for you. This is a great way to skip almost everything I've mentioned here about formatting, but be aware that you'll also sacrifice some of your options and control over the design of your file.

3. Publishing - Now you take the interior file and your cover file (or not; you can design a cover while publishing, too) and upload them to Lulu. We'll ask for other information, like the ISBN, author name, copyright year, and description, and we'll set up a payment account if you're going to be selling the book.

Your PDF must be a single-page (no spreads) file, and the first page must always be printed by itself on the right. We worked really hard and advocated with our Developers to get a previewer built into Lulu's publishing process. Please use it or whatever preview option your platform of choice offers. The more you review the file, the fewer errors you'll find in your proof copy.

Additionally, the proof copy deserves mention.

You would not believe how many authors go through all of the process to write, edit, format, and publish, and then don't order a copy to review before making it available for sale.

Does every book need a cover page or other front matter?

The answer is both yes and no.

You could upload a file without any front or back matter, publish it on the Lulu Bookstore or through your own site, and sell it without any problems. When you self-publish, things like a copyright page, title and half-title, acknowledgments, table of contents, description, and so on are optional.

The asterisk on that statement means that it is only true if you sell on Lulu or directly to your fans. If you want to publish and list your book on retail sites like Amazon or the Ingram network, you will need all that standard front matter.

What about back matter like a glossary?

The back matter is much less important than the front matter. You can't publish and distribute your book to retailers without a title, half-title, and copyright page, but an about-the-author page, glossary, or index is not required.

I think most authors don't need an index. Fiction books rarely have information that needs listing (unless you're a fantasy or sci-fi writer detailing magic systems or alien species). The index and glossary are more relevant for nonfiction or reference books.

What is your favorite font for printed work?

My personal rule is that serifs belong in print and don't belong on my screen.

I'm also big on keeping fonts simple. For a good, readable body font, I love Baskerville or Century. I don't think authors should get fancy or weird with the body font. Your readers are going to be looking at this font for ten or more hours! Don't torture them with Comic Sans.

Now, regarding cover text and heading styles, I favor mixing them up. There's a ton of psychology behind fonts and what a typeface conveys to readers. People tend to associate serif fonts with adjectives like *traditional* or *elegant*. San serif tracks the other direction, with adjectives like *modern* and *clarity*.

For novelists, thinking about your fonts' impact creates another opportunity to develop your story. One of my favorite examples of using a font creatively comes from the British author Terry Pratchett. He introduces Death as a tangible character in his long series of campy fantasy novels. Only all of Death's dialog is in a smaller, caps-locked font, quite literally giving more weight to this character's words.

More commonly, you’ll use unique fonts for your chapter or section titles, and, of course, the book title.

DITA projects in Oxygen XML Editor

For my DITA projects, I use the Oxygen editor. I wanted to learn more about how to use Oxygen, so I reached out to Radu Coravu. Radu is a DITA XML expert working for Oxygen XML Editor. Radu's main focus is in the development of the visual XML Author editing environment and the specific-DITA support provided by Oxygen. He provides support for complex integrations and helps steer the product in the right direction—all this with some development on the side.

I asked Radu about how technical writers use Oxygen to work on DITA projects:

Let's start with an introduction to DITA. What is DITA?

DITA XML (like any XML standard) is about single sourcing, using a single set of documents to produce various output formats.

In addition DITA XML comes with lots of reuse possibilities, starting from big reuse (producing documentation for similar products from the same set of XML documents) to reuse of topics and elements.

DITA projects can be quite large (thousands of topics). How can technical writers collaborate to work together on DITA projects?

We have quite a lot of clients (both small, medium and very big companies) which use Git with Oxygen to work and collaborate on documentation projects.

Using free Git platforms like GitHub or GitLab, you can create technical documentation projects, collaborate on them using the features that Git provides to create branches, see the history of resources, and manage conflicts.

There are also plenty of commercial content management systems (many of which are integrated with Oxygen) which allow people to work and collaborate on a large technical documentation project.

What other kinds of projects can I manage with Oxygen?

Oxygen is a multi purpose XML editing tool. By default it comes bundled with support for XML vocabularies like DITA, DocBook, TEI, JATS or (X)HTML. But it can be customized to edit and validate any kind of XML vocabulary. We have clients who create XML vocabularies specific for their companies from the ground up, define their own elements, implement their own validation rules and create an Oxygen framework to visually edit such documents in the Oxygen Author visual editor using CSS to style the edited content.

Lately we also shifted some of our capabilities and added stronger support for other non-XML types, like support to edit and convert JSON documents, support to edit YAML configuration files.

I've only used Oxygen for DITA projects. What other cool features in Oxygen should I know about?

Oxygen has many strong points. For example, we provide support for the entire DITA XML specification along with special publishing plugins which allow you to produce quality online WebHelp and PDF outputs and to style these outputs using CSS. We even provide an online color themes/styles builder to customize the publishing styles

Oxygen allows defining custom validation rules using the Schematron standard which allow you to impose your own company specific restrictions.

We also provide lots of useful free add-ons, like an add-on for Terminology Checking or a Git client add-on for working with Git projects.

Why I write about open source software

Don Watkins is a longtime technical writer about open source software. Don is also part of the creative team behind Both.org, a new article-based community website about open source software. We asked Don about his background and why he started writing with Both.org.

What's your background in writing about open source?

I'm a retired public school teacher employed for twenty-seven years as a school district technology director at a small PK-12 school district in New York State. In that assignment, I taught students and staff about technology and how to use it.

In the process, I became familiar with Linux and open-source software and saw the unique opportunity it presented to those who used it to level the playing field. After retiring in 2013, I got involved with the local and regional public library system, where I began to teach younger patrons and librarians how to use Raspberry Pi computers and how to write programs on them with Python.

You are a prolific technical writer about open source software. How did you build your writing skills?

I started writing articles for Opensource.com in 2015 about my love of open source and contributed nearly three hundred pieces. While I was writing for Opensource.com I was approached by the editors who asked me if I would like to improve my writing skills. I told them I would, and over the next almost eight years, I became more proficient as a writer and was allowed to diversify the topics I wrote about within the open-source community.

This led me to interview several individuals from diverse backgrounds and also allowed me to explore the nuances of all things that are open in software, hardware, and ethos. As a writer, I benefited greatly from copy editors who reviewed and refined my work.

When Opensource.com ceased to be an avenue for my writing, I began to look for ways to improve the punctuation and accuracy of what I produce, which led me to use Grammarly. I have been a good speller for most of my life, but I never bothered to learn the art of punctuation. I purchased a premium account, which is quite reasonably priced, and it has been a boon to my technical writing. It points out where I've made mistakes in punctuation and usage and allows me to improve my writing.

I use Grammarly when I edit other folks' articles at Both.org, and it's been beneficial. Having a background in Linux and open source is also beneficial, as I need to know if the correct commands are being used in an article that provides the reader with a 'how-to' to perform specific tasks in an operating system or the use of an application.

You mentioned Both.org, which is a new website about open source software. How can new authors contribute articles to Both.org?

Currently, our workflow at Both.org is quite simple. We have five editors who review content created by folks who write for us. We have a mailing list for editors where we ask our peers to review what has been written. When we've assured each other of the accuracy and completeness of a particular article, David Both schedules it for publication on an open day.

A good editor has to be a good speller or have the technology to spot improperly spelled words. Punctuation and syntax are equally crucial to producing technical articles that will be used for resource materials by the folks who come to rely on the content we produce every day on our website.

Patience and the ability to adequately and politely explain how an article needs to be improved is important. Encouraging others to share their unique perspective on a particular topic or task is important.

Most folks suffer from Impostor Syndrome and don't think they can write or are good enough to create a good copy. It's our job as editors to encourage others to write and to look for ways to improve their writing.

How I write books about Linux

David Both is an experienced technical writer who has written several books and many articles about Linux, systems administration, and open source software. I asked David about his background as an author, the tools he uses, and what his writing process is like.

What's your background before you started writing books about Linux?

I'm one of the first baby-boomers born after the end of WWII. I grew up in the 1950s, reading science fiction written by some of the all-time great writers like Heinlein, Clarke, Asimov and more. These authors introduced me to technology and the concepts of computers and automation.

I was the kid in high school who carried my slide rule under my arm with my Chemistry and Physics books. Yeah, that guy!

In those ancient days, all electronics like TVs and radios came with schematic diagrams and I used to find them and read them. I'm not sure how much I really understood, but by the time I was ten I was fixing our TV and those of our friends and neighbors, too.

That, and other circumstances that are too long to relate here, led me to a job at IBM. From there it was a short leap to supporting PC hardware and software. I supported DOS and OS/2 for the IBM PC Company for several years

After leaving IBM I had several interesting jobs but eventually decided that OS/2 was going away and that I never wanted to learn enough about Windows to have a job that required me to support it. So I switched all my personal computers to Linux over a period of about a week and I've never looked back.

What is your writing process like?

I've written hundreds of articles for various print and on-line publications. Most of those were at Opensource.com and its sister publications.

I've also written five books about Linux for Apress. The current titles are *The Linux Philosophy for SysAdmins* (August 2018), *Linux for Small Business Owners* (August, 2022), and *Using and Administering Linux—From Zero to SysAdmin*, Second Edition (three volume set: October, 2023).

My writing process varies quite a bit. It also starts long before the idea of a book actually occurs.

My first book, *The Linux Philosophy for SysAdmins*, started as an idea I had after reading about the history of Unix and books by some of the original developers. One book I especially like is Mike Gancarz's book, *Linux and the Unix Philosophy*. As I read that book, it occurred to me that the Unix philosophy was for developers but that many of the tenets of the Unix philosophy were also applicable to system administrators.

My own mentors had helped me to formulate a philosophy of my own, one which applied to system administrators. However, there was no philosophy specific to us as system administrators. So I wrote a couple articles for Opensource.com about Linux philosophy, but for the system administrators.

"The impact of the Linux philosophy" is about why the philosophy of an operating system matters. But "How the 9 major tenets of the Linux philosophy affect you" was where I started toying with the idea of a separate Linux philosophy for system administrators.

Eventually I used those articles as the basis for my book.

As for my Linux self-study training series, *Using and Administering Linux—Zero to SysAdmin*, that developed from two directions. Over the years, I have encountered many challenges and problems when fixing problems with hardware and software. Because I found myself needing to figure out the same things multiple times, I started using a database as a memory aid.

Also, during the time from about 2001 through the 2010s, I taught some Linux classes that I had put together myself.

When I decided to write this series, I took both of those works, laid out the overall structure in chapter sequence, and chose appropriate points to split the work into three parts. I then added a few chapters to fill in some parts that needed more coverage.

Within each chapter I usually start with headings and subheadings for existing contents, and add more where bits are missing.

What tools and technologies do you use to write your books?

I use LibreOffice Write for my books. My publisher, Apress, prefers to get `.docx` files. I start by creating my chapter files in Open Document Text (`.odt`) format. That is always my reference copy. When I'm ready to submit a chapter, I save it in `.docx` and export it in PDF. LibreOffice can do that directly without extra software for either format.

I have started to use Markdown for some of my articles, but I still prefer LibreOffice because it has so many built-in tools to support the advanced formatting I like. Things like inserting graphics, footnotes, linking references to a section or graphic, and list renumbering. Things like that make writing complex documents much easier for me.

What recommendations would you give a new author?

The thing I tell new authors is to just get started. Get something down on the page and go from there.

I tend to switch between writing just to get something I've been thinking about onto the page and I can do that for hours. I do like to go back and edit after a session like that to make sure I said everything I wanted to and that the flow makes sense.

Before I submit my chapters, I like to go back over them a couple times at least in order to smooth out awkward phrasing, make sure I haven't missed anything I wanted to say, and delete stuff that's really irrelevant. My editors are right on the edge of brutal when it comes to excising the excess.

What does the author/editor working relationship look like?

That has to be a smoothly working team effort. I've had two or three editors for each of my books. That is, editors who actually review and make changes and suggestions as well as a senior editor. The senior editor is the person who manages the entire process, ensures that I have what I need for research, and is my primary contact at my publisher.

The development editor is more about grammar and style in the sense of whether to use Oxford commas or not. That's a definite yes. I like to write conversationally so I use pronouns like "you" and "we" a lot. That's a style that they like, so I'm good there.

Technical editors are there to ensure that the technical details are correct. And mine have done a great job on all my books. They have recommended removing sections or entire chapters or moving them from one part of the book to another in order to make more sense in the sequence for the readers. They point out technical stuff that was just plain incorrect.

They let me keep my voice. That is, say things that indicate my likes, dislikes, recommendations, and other feelings relating to what I've written. For example I write about my experiences and talk about the PHBs, the Pointy-Haired Bosses of Dilbert. My disdain for PHBs is quite evident so my writing can have more impact that way.

As I complete each chapter I submit it and the editors go to work. Sometimes I split my working day into a segment for new work and another segment for reviewing the editors suggestions. Once I'm well into the book, I usually spend part of the day working on new material and part on revisions.

How do you and your editor review edits to your books?

My editor sets up a space in a Cloud drive and I upload my chapter first drafts there. The editors review the docs, add notes to suggest changes, and move the files to another folder when they're ready for me to do the revisions. Using my original `.odt` files with Track Changes enabled, I make changes that I think make sense. I then upload my revised files to another folder in the cloud.

I also generate PDFs of the revised files and upload those as well. That's an accurate view of the special formatting I may want for tables, graphics, code listings, and output data streams. The production team in Chennai can use those to verify that they've got the typesetting correct.

It's not very often that I reject a suggestion to make a change. Editors are there for a reason and I think the suggestions they make have made all of my books much better.

From there, the files go to production.

What happens in the production process? Are you still involved at that point?

Yes, I'm still involved. The production team performs the typesetting and sends me the final proofs for each chapter in PDF format. At this point I can only make minor changes. I can change spelling or add or delete a few words. Any change that would affect the pagination is not possible at this late stage.

I use Okular, which is a free and open source document viewer, to mark up or comment on the PDF files and upload them to yet another folder in the cloud.

Then I wait for my author's copies to show up on my porch.

What is it like to get a bunch of edits or edit suggestions back from an editor? Is it a hit to the "writer ego" that the editor asked you to change things, or is that just part of the process?

I do have an ego or I probably wouldn't write books and articles. But my ego wants my books to be the best they can possibly be.

I have a great relationship with Apress and the editors of my books. They all want the books to succeed as much as I do and we work together as a team to make that happen.

My technical editors have been the best. My latest work, *Using and Administering Linux—Zero to SysAdmin*, is a good example. Seth Kenlon was my technical editor for all three volumes. We've worked together before and have met here in Raleigh at All Things Open on a few occasions, so we have a good rapport. I once joked with him that he'd been "on the ragged edge of brutal" for his edits on a chapter. He responded, "I was trying for *completely* brutal."

Reading is an important part of writing. What are your favorite books or websites about Linux?

Well, most of them are about Unix which is why we've needed more books about Linux and open source. That's why I started writing articles and books - to fill that gap. These are the ones that have influenced me the most:

- Eric Raymond's *The Art of Unix Programming*
- Mike Gancarz's *Linux and the Unix Philosophy*
- Brian W. Kernighan's *Understanding the Digital World*, Second Edition

Writing about programming: FreePascal from Square One

We love to share stories about technical writing. One recent example caught our attention: Jeff Duntemann is a longtime technical writer and author, and he recently announced a free ebook about programming. Jeff's *FreePascal From Square One* is a distillation of several of his other Pascal books, and an excellent starting point for anyone who wants to learn about programming.

We asked Jeff about his background as a technical writer, how he wrote his new book, and what readers can learn:

Let's start with an introduction

I'm Jeff Duntemann. As best I know (and I have traced the Duntemann line back to a chap born in 1687) I'm the only Jeff Duntemann who has ever lived. That's a big plus for a writer; I've known three guys named Mike Smith. Fortunately for them, none are writers. I'm a contrarian, and enjoy defying conventional wisdom. This allows me to live my life as I damned well want to. As for what I do, it's a long list. First and foremost, I'm a writer. I'm also an editor and layout artist, though I use the term "artist" loosely here. I read, and read all sorts of things. Not all readers are writers, but all writers are readers. (It's a kind of metaphysical law.)

I coordinate a writers group. I program—(almost entirely) in Pascal. I'm a ham radio op, callsign K7JPD. I tinker with electronics. I build and use telescopes. Big ones. I build and fly kites, doubtless a habit inherited from my childhood in the Windy City. I work out. My great-grandmother made it to 96. That's my target. And on an irregular but frequent basis, I sit back in my big cushy chair and *think*.

You have a long history in technical writing. How did you get started?

When I was 8 (1960) I wrote a story about my toy dogs building a spaceship and beating the Russians to the Moon. It was my very first story. My journey as a writer actually began when I was 6, when my father took me to the local Chicago library branch to get my own library card. The librarian refused; the rules said kids had to be 7. My father threatened to call the alderman. I got the card.

I continued to write stories, and the fiction I wrote incorporated the nonfiction that I had read in library books. Rockets. Space. Astronomy, Volcanoes. Submarines. Much else. At 11, I taught myself electronics from library books, and built crystal sets, 1-tube radios, and a Geiger counter, among other things. In 1973, I began selling science fiction short stories to science fiction magazines and anthologies. Later on, in 1976, I wire-wrapped my first computer, out of an article in Popular Electronics about RCA's 1802 microprocessor. 1-tube radios were cool in their way, but computers, yikes! I did a lot of wire-wrapping in the late '70s, and in 1978 built a robot with two computers in it. I took Cosmo the Robot to a science fiction convention, and a reporter for Look Magazine who I met there included Cosmo in an article about robots in Look. Now that I knew how computers worked, right down to the metal, I began writing articles about them—on a *typewriter*, sheesh. My technical articles appeared in Creative Computing, Kilobaud, Byte, 73 (a ham radio magazine) and a few newsletters and anthologies.

I bought a "real" computer in 1980, an S100 box with floppy drives, a word processor, and a printer. I retired my typewriter. A year later, I bought an IBM PC. The PC was my programming machine, on which I taught myself BASIC, FORTH, and Pascal, using the now long-extinct compiler Pascal/MT+. I began selling technical articles to Ziff-Davis' PC Tech Journal in 1983, and later went on to work there as technical editor.

A colleague introduced me to an acquisitions editor at the office of book publisher Scott, Foresman in 1983, and I pitched him a book on Pascal. He bought it on the spot. It took most of a year to write *Pascal/MT+ From Square One*. While I was writing it, this new Pascal product popped up out of nowhere, and rapidly drove most other Pascal compilers into the sea. When I turned in the manuscript, my editor asked if I would rewrite the book to focus on Turbo Pascal instead of Pascal/MT+. By then I was already programming in Turbo Pascal, so although it delayed the book for almost another year, I did the job and turned it in. Alas, Scott, Foresman changed the title without consulting me, to *Complete Turbo Pascal*. That didn't hurt sales. The book went through four editions, the last one, in 1993, to a different publisher, after Scott, Foresman was acquired and closed down its tech books division. Bantam Books allowed me to change the title to *Borland Pascal 7 from Square One*. I had written an advanced topics book about

Turbo Pascal in 1988, mostly about using assembly language from the compiler. Turbo Pascal Solutions didn’t sell all that well, but it re-kindled my interest in Intel x86 assembly language.

Assembly language. Yes. It was a hidden passion, rooted in having to hand-code binary executables on my wire-wrapped 1802 boards back in the ’70s. I mentioned that experience to my new editor at Scott, Foresman, and *she* pitched *me* a series about assembly language, with me as author of one title and editor of the series, in exchange for a fair chunk of money. I knew a lot of technical authors from my years at PC Tech Journal, and planned the series as a beginner book, an intermediate book, and an advanced book. I wrote the beginner book, *Assembly Language from Square One*. The other two books came in well, especially the third book, Michael Abrash’s *Zen of Assembly Language*, which became legendary. But alas, as I mentioned earlier, Scott, Foresman was acquired by Harper Collins in 1991, and our series was put out of print. I took my beginner book to John Wiley & Sons, who published four editions starting in 1992. They changed the title to *Assembly Language Step By Step*, and it has sold very strongly up to the current day. I rewrote it heavily to include x64 assembly for the 2024 fourth edition.

In 1989 I co-founded The Coriolis Group with writer Keith Weiskamp. It was a publishing company, established to publish a programmers’ magazine and (a little later) technical books. In 1996 I co-wrote *The Delphi 2 Programming Explorer* with Jim Mischel and Don Taylor and published it under the Coriolis imprint. The book was a great success, as it also taught object oriented programming and Windows event-driven programming, foreshadowing FreePascal and Lazarus.

I wrote the “Structured Programming” column in *Dr. Dobb’s Journal* from 1989 to 1993. I touched on Modula 2 here and there, but in truth almost everything I wrote for *DDJ* was focused on Pascal.

I sold my first science fiction novel to a small press in 2005. After we closed down Coriolis in 2002, I self-published several more volumes of my own science fiction and fantasy. My fiction has done well in the Kindle era, but I’m best known for my technical writing.

Where did the FreePascal From Square One book project come from? Why write this book?

I discovered FreePascal and Lazarus in the late oughts. At that time I was still programming in Delphi, mostly for fun, and as Delphi's price tag began to soar, my interest in it waned. I teach programming, and I can't teach programming using a product that costs $1000 or more per user. Turbo Pascal? Fifty bucks, sure. A thousand bucks or more? No way.

Early in my tech writing career I discovered that I had a talent for explaining technical concepts to newcomers. By 2010, all my Pascal material was long out of print and slowly being forgotten. With FreePascal becoming popular, I had a chance to create a new beginner book for Pascal. I decided to release it for free. Why free? I was paid for that material four times, and paid rather well. It was my way of thanking everyone for buying my Pascal books down through the years.

The project began in 2010, and I tinkered with it on and (mostly) off for fifteen years. Beginning with *Borland Pascal 7 from Square One*, I removed all the platform-specific stuff, like interrupt and system calls, the Borland Graphics Interface, and anything else focused on DOS. I also decided early on that I wouldn't attempt to teach object oriented programming or GUI events. The plan was to have two goals:

1. Teach the fundamental nature of programming, for people who had never programmed before.
2. Teach the basics of the Pascal language, without going into object oriented programming or exotica like generics.

I wanted to keep the length of the book to 350 pages in the A4 trim size. The final PDF that I released on May 7, 2025 came to 354 pages. Nailed it. The idea was to make it long enough to be useful but not so long as to be intimidating. Now, object oriented programming is important, especially for GUI work using component-oriented GUI builders like Lazarus. I have the material extracted from my parts of my Delphi book. It will go into building a new book that will assume familiarity with programming and Pascal itself, and go further.

This time it won't take fifteen years.

You mentioned that this book is a distillation and update from your Turbo Pascal books. What else have you added?

What I did was a close developmental edit on my Turbo Pascal material to make it line up with FreePascal. In truth, I mostly took extraneous material *out* of *Borland Pascal 7 From Square One*. What I added I can rightly call *focus*, not only on FreePascal but on the ideas behind programming in general. I want the book to be accessible to hobbyists and especially homeschoolers. That focus, and an emphasis on brevity, allow the book to be used by sharp kids as young as 12 or 13.

I also added text explaining how to use Lazarus as a text IDE for FreePascal, without going into the Lazarus GUI builder. That will star in my current tech writing project: *Lazarus from Square One*.

What's your process for writing a book like this?

The first step after I get an idea for a book is to draw a conceptual line between what's in it and what's not in it. My discussions with people learning programming from books has taught me that focus is *very* important. Too many wandering threads tend to obscure the core material. So in a sense I write a description, then edit the description to remove things that don't contribute to the book's focus. I distill this description into an outline, where each major point in the outline is a chapter, and secondary headings naming subtopics are sections of the chapter.

The toughest decision I made in outlining *FreePascal from Square One* was stopping short of object oriented programming. Object orientation is important, critically important, but it's a subtle business and should be described only after the student has worked through the fundamentals of the language. The example programs in the book are short and fairly simple. The idea is to get across things like `IF`/`THEN`/`ELSE`, `REPEAT`/`UNTIL`, simple data types up to and including records, and so on. OOP is about conceptualizing a program under a specific paradigm. Starting with that paradigm requires a separate and more advanced book, one I'm working on.

What word processor or other software did you use to write this book?

I've used Microsoft Word as my word processor for 35 years now. Before 1990 I used WordPerfect, and prior to 1985 I used Wordstar. Word was originally created in 1983 by people hired away from Xerox. I worked at Xerox my first ten years out of college, programming in-house languages that are now best forgotten. (Not Mesa or Cedar. I *wish*.) But I mingled with people who used and developed GUI apps, and I played with Smalltalk on the Alto machine in our building's lab. Late in my career there I taught staffers how to use the Xerox Star workstations being installed in our building. Xerox laid the foundations for GUI software, and it shows in Word.

Layout for print books and PDF ebooks I do in Adobe InDesign. Layout for reflowable ebooks (which I do not do for technical books because of reflowing hassles) I do in Jutoh, a British layout program designed specifically for creating reflowable ebooks.

And a fun question: Everyone has a favorite font. What are your favorite fonts for print?

I bought Albertina when I started laying out my own books in the mid-late oughts, because I wanted to be sure that I had the rights to embed the font in the books. All my new material is done in Times New Roman, as rights issues for that font have been settled. As for title/spine fonts, my favorite font is the free Good Times font from DaFont.com for science fiction, and for fantasy books I use various others. I use free fonts to avoid rights issues. For me, Good Times is the best.

Technical writing with GNU groff

Many technical writers of my era may have started with a text formatting system rather than a word processor. Nroff and troff were the original text formatters on Unix systems, and remain in use today, typically implemented as GNU groff. With nroff and troff, technical writers add formatting instructions in the body of a document, such as `.br` to create a line break or `.sp 3` to add a few lines of empty space between paragraphs or sections. Troff also provides tools like `eqn` to generate equations and `tbl` to build tables within documents.

I asked Brian Kernighan about the history of nroff and troff, and how technical writers can explore these tools today. You may recognize Brian Kernighan's name. He is the "K" in Awk, the "K" in "K&R C" (he co-wrote the original "Kernighan and Ritchie" book about the C programming language), and he has authored and co-authored many books about Unix, programming, and technology. On my own bookshelf, I can find several of Kernighan's books, including *Understanding the Digital World*, *Unix: A History and a Memoir*, *The Unix Programming Environment* (with Rob Pike), *The Awk Programming Language* (with Alfred Aho and Peter J. Weinberger), and *The C Programming Language* (with Dennis M. Ritchie).

What are nroff and troff, and where did they come from?

Nroff is for basically ASCII text in a fixed-width font; it mimics ancient terminals like the Model 37 Teletype and line printers of the 1970s. Troff is for typesetters, which provide variable-width characters in multiple fonts, which is what we see in print today. That includes books, magazines, newspapers, and so on.

The first such program was Runoff, written by Jerry Saltzer at MIT in maybe 1966; it was (in my experience) the first such formatting program and provided the model for a lot of subsequent ones. In 1968, I wrote a simpler version for printing my PhD thesis at Princeton that I called "roff," but I think Doug McIlroy at Bell Labs wrote a different version in the same style that he also called "roff." Nroff and troff adopted the same style of formatting commands, but were significantly more powerful and could do more complicated formatting, especially page layout.

The original version of troff by Joe Ossanna in about 1973 was aimed at a particular typesetter, the Graphics Systems Model C/A/T. Ditroff ("device independent troff") was my updated version of troff that produced output that was independent of any particular typesetter. It generated output that was parameterized for specific devices, and separate drivers produced final output on different typesetters.

How can people get started with groff to write documents? Would you recommend a popular macro set like -me or -ms or -mm?

Absolutely start with a macro package. I use `-ms` since that's the one that I'm familiar with, but there are several others that are widely used. Troff is straightforward for simple formatting but page layout is harder, and the macro packages take care of that, along with providing convenient shorthands for standard formatting operations like paragraphs, headings, numbered items, and so on. Floating figures is an example of something that is challenging to get right, but the macro packages take care of that.

Nroff and troff can generate a variety of printed materials. What kinds of documents did you write in nroff and troff during your time at Bell Labs?

We used nroff and troff for pretty much everything. There were a lot of internal documents at Bell Labs; the Unix manuals were done in nroff and later in troff; a fair number of books used troff (including all books that I have ever written); patent applications were done with nroff (that was the original killer app for Unix internally); and even already-formatted documents for journals long before that became the standard. Lorinda Cherry and I published a paper about `eqn` in *Communications of the ACM* in 1975 that was formatted with troff and published in exactly that form; so far as I know, it was the first such paper.

What kinds of documents do you write in groff today?

Groff is a great piece of work. It's compatible with troff, including all the old warts, and it comes with new versions of `eqn` (equations), `tbl` (tables), `pic` (pictures), `grap` (diagrams) and other standard preprocessors. At the same time, it adds new capabilities, a bigger namespace for user commands, and a variety of convenience features. I am eternally grateful to James Clark for his work on groff, and to the others who continue to maintain it.

Today, my use of groff is entirely for books. Most recently, I have done a second edition of the original Awk book with Al Aho and Peter Weinberger; that should appear in a couple of months. I also did a second edition of *Understanding the Digital World*, and the *Unix: A History and a Memoir* three or four years ago. All groff.

What groff macros do you use to write your books?

Groff is just like troff in this respect: it's a toolkit, not a specific set of macros. What I do is to take some basic macro package (`-ms` in my case), then modify existing commands for headings, titles, and so on, and add new macros for special purposes. For example, I almost always add new macros for things like starting and ending blocks of code; those macros change the font to a monospace font like Courier, set the point size, vertical spacing and indent, turn off line filling, and so on, then revert to normal text at the end. I do similar things for other kinds of displays, and I usually have macros for inline font changes for including monospace program words in running text.

I often supplement all of this with code in Awk or Python to do cross references in text, set up running headers for pages, indexing, and so on. So printing a document is more involved than just running a groff command - it's a shell script with a bunch of steps.

When I write in groff, I find I get into a flow where I don't notice the macros, I just focus on the content. What is your workflow when writing with groff?

My style is perhaps a bit different. You're right that after a bit, you don't see the commands much. But what I do is to edit the text, save it, run a troff command (usually groff with various ancillary tools), and look at the result in a PDF viewer like Preview on a Mac. The whole process only takes maybe fifteen to thirty seconds even for a book of a couple of hundred pages, so it's reasonably interactive.

Technical editing in a community

Seth Kenlon is a longtime technical writer and open source contributor. Seth is also part of Both.org, a new article-based community website about Linux and open source software. We asked Seth about his role there and what it's like to serve as editor.

You've been a technical editor for some time, in different capacities. What is that like?

People tell me that I have a knack for explaining complex concepts in a relatively simple way. In 2016, I got hired by Red Hat to help build training courses, and later I joined the team managing a blogging site called Opensource.com—discontinued now, but happily its content remains online.

During my tenure as technical editor, I had the privilege of reading literally thousands of articles, and the further privilege of working with authors to ensure that the articles were accurate and clear. I do the same thing now in a volunteer capacity for the Both.org tech community.

At Opensource.com, you served as both technical editor and Technical Community Advocate. What was that role about?

As the Technical Community Advocate, it was my job to ensure that authors were getting real benefits from their contribution. The site's supporting company was paying all the bills, which was amazing, but all the authors were contributing as volunteers. I was tasked with ensuring that the authors were getting out of that deal as much as, or more than, they were putting in.

To do that, my team and I turned the spotlight on the author. That's easy to do, because every contributor in more than twelve years of the website was essentially a subject matter expert. You don't have to know *everything* to be an expert, because nobody does. Even when you're writing your very first article ever about the simplest Python "hello world" script, you're a subject matter expert on *being a new Python programmer*. As popular as Python is, most people in the world are also new to it, so that new author who hasn't even discovered what a function is yet has a lot of valuable information to share.

When I edited an article, I looked for places where it could be more succinct. I looked for places that needed to be simplified for clarity. And of course, I tested everything and shared my findings with the author. It was an amazing experience for me - and I hope for the authors as well.

Now you're also volunteering as an editor at Both.org. What is your role as editor like?

Well, first of all: the most important thing about technical articles is that to be effective, they have to render success. If you look up "how to exit Vim" and find twenty articles about it, the best article is the one that works. It can be overly verbose, it can have atrocious grammar and spelling - but as long as it gets you out of Vim, you're happy.

We see proof of this today with ChatGPT, which just makes up answers half the time. But because they're expressed in a way that asserts authority, people accept them. But anybody who's tried to get ChatGPT to write code for them knows that just because something is very nearly right, it doesn't mean it actually works.

For Both.org, it's my goal to ensure that articles are accurate. I don't necessarily have the luxury of rewriting each sentence three times until I find the most efficient and clearest way to express an idea. But as long as the article ends in success, then I think Both.org is bringing real value to the tech community.

Of course, there's room for "water cooler" chat on Both.org, too. Not every article has to have a point other than musing about updating RFC 527. I love idle tech articles, and for those kinds of articles I scrub them for grammar and spelling and then give them the stamp of approval.

What skills does someone need to be a good editor?

First, I look at an article and try to understand what single task it's trying to achieve, even if that single task has two or three stops along the way. For instance, an article about "how to install Wordpress and share it with the world" actually involves installing Wordpress, configuring a web server, and opening a port in your firewall, but there's still only one driving point to the article.

If there's not a single point to an article, I tell the author that it needs to be two (or more!) articles. In Linux, we have the saying "do one thing and do it well", which is really just a fun way of supporting modular design. An article, in my opinion, should do one thing. If you want to do more than one thing, write more articles and string them together.

Having established the singular point of an article, I then look at each paragraph. It may not seem like it, but technical editing is a form of programming. The English language (in my case) is the code, and the paragraphs are the functions. A paragraph has an *input* and an *output*. The input is generally the previous paragraph. The paragraph's outcome is its output. If a paragraph hasn't changed your input state, then there's the probability that the paragraph isn't necessary. A good [bad] example is the paragraph above this one. I'd already made my point that an article should only do one thing, but then I wrote a whole other paragraph that reiterates the same principle. Were I editing this interview, I would delete that paragraph, and I wouldn't apologize to the author for it.

After determining the validity of a paragraph, I turn my attention to the sentences within the paragraph. Sentences are the lines of code that build the function, and usually the less logic you include, the less chance there is for failure. I like short sentences, and not too many of them. I tend to cut out adjectives and adverbs. I cut out conjecture and most of any opinions or idle musings.

It's programming with natural language, in the attempt to create a set of instructions that a human can follow to achieve the exact same outcome, every time. It's a recipe, or a game.

Here's a fun question: What's your top editing pet peeve?

The process of technical writing often involves fits of inspiration combined with frantic research, trial and error, and revision. If you dare edit the result of that, then you must be ready to accept anything. At the time of writing, that particular choice, however bad it might seem now to you as the editor, really did make sense to the author.

Saying that another way: everything is my top editing pet peeve. As an editor, nothing's ever written well. And no matter how hard you try, you'll never be able to make it foolproof. Such is the human condition!

About the Editors

Jim Hall is an open source software advocate and technical writer. At work, Jim is CEO of Hallmentum, an IT executive consulting company that provides hands-on IT Leadership training, workshops, and coaching.

Will Decker is a technical writing student at the University of Minnesota and is a co-editor of this book. He is interested in copyediting and scientific writing. Will will graduate from the University in May, 2026, with a degree in Technical Writing & Communications and a minor in geography.

Technically We Write is a community-based, article-driven website about technical writing, technical editing, usability, and technical communication. We welcome everyone to share an article with us.

You don't have to have a “technical writer” title to share an article on Technically We Write. Everyone writes technical or professional documentation, of some kind, no matter their role, organization, or affiliation. Technical and professional writing is everywhere.

We’d love to share your story! Your article can be about anything related to technical communication, such as tools, tips, how you got started, what you’ve learned, or other topics in technical communication. Join us at *technicallywewrite.com* to share your article idea.