For the Love of Troff

James K. Lowden

Symas Corporation

ABSTRACT

The state of Unix documentation has not advanced for decades. The venerable man page remains the backbone and, as Hoare might have predicted, an improvement over many of its successors.

The plethora of competing formats and systems has created a fragmented and confusing body of documentation, confusing to users who don't know where to look, and to writers who don't know where to start. One need only to search for HOWTO documents on writing documentation for conformation.

This paper reviews the state of the art, and critiques the variety of documentation systems in current use. It explores why new systems were invented, which are currently fashionable, and suggests a path to a better, unified documentation system based wholly and solely on troff.

The Stale State of the Art

In the beginning was the man page. It divided darkness from light. And it was good.

But apparently not good enough. Would-be successors include Texinfo, DocBook, Doxygen, and Markdown. Formats, beside the terminal console, include PDF and HTML. Locations might be /usr/share/man, or /usr/share/doc or /usr/share/info or /usr/share/html, or pretty much anywhere else in /usr/share, or only on the web.

To each its own silo. There is no comprehensive cross-reference, no accepted universal structure or encoding or output format. There are different schools of thought about how to encourage better documentation to be written, what the best output medium is, and what system should be used to render the output.

Rather than progress toward a single, all-encompassing system that utilizes the full capacity of large memories, Unicode, and bit-mapped displays, documentation technology is regressing and fragmenting. The built-in typesetting system, troff, is considered by many to be vestigial and archaic. Yet every alternative is less functional, or more complicated, or both. On most BSD systems troff has been replaced by the less functional mandoc. GNU continues to promote Texinfo. Perl had pod. Python has pydoc and promotes Sphinx. Subversion goes post-modern: web-only. Many use Markdown, a system with no standard, never mind support for footnotes.† X.org began with a set of manuals in troff format. Instead updating their content, the project opted to re-encode them in Docbook and/or Doxygen.

The dysfunctional Unix documentation system has hindered progress in other ways, too. For example, command-line shells cannot rely on documentation files to provide interactive help. The user cannot type

$ **ssh** help

to get a reminder of the available options, or — heaven forfend — a *form* to facilitate making the choices. The bare-bones experience at the bash prompt is in part the product of the lack of a unified, programmatically accessible body of documentation.

† Footnotes! Dude! You can say other stuff that's pertinent but not strictly necessary, without interrupting the explication.

The origins of our Tower of Babel are historical and psychological. There is a certain amount of Not Invented Here, some issues with copyright and license, opinions on what is modern and what the future will look like, and strong brew of technical arrogance.

Probably the tide cannot be held back. That ship, I'm often told, has already sailed. And surely one audience reading one paper at one conference will not stem the tide, or unsail the ship, or whatever. But perhaps if we chart the course we have been on, we can, slowly, steer the ship and sail toward a better destination, instead of straight to Terra Incognita, with no one at the helm.

History of Everything

The history of everything is re-invention. The beginning programmer re-invents simple tools and eventually discovers very few simple *sui generis* tools are better than those found in /usr/bin. After about five years, though, the beginning programmer becomes an intermediate programmer, and there disaster awaits.

It is hard today to find any Unix subsystem or function that hasn't been *repeatedly* re-invented, often to little or no effect. lpd(8) gave way to CUPS. /etc/rc.conf gave way to /etc/rc.d. init(8) led to systemd(8). Naturally, *there were reasons* — perhaps even good reasons — to replace system *ante* with sytem *post*. But the observable advancement, from the user's point of view or by any objective measure, was close to nil. Very few examples can be found where the functionality offered by the replacement could not have been added to the replaced, with less dislocation and need to "upgrade".

So it is with documentation. To invent a system for encoding and rendering documentation is not difficult. Neither are the problems inherent in any given system hard to see (although the obvious ones are usually not the important ones). The goal seems equally simple, even if it prove harder to achieve than articulate. So it is not surprising we have a succession of tools and systems all hoping to slay the dragon, and make documentation safe in our time.

History of Documentation Systems

The halcyon days of documentation systems were the 1990s, just before Bill Gates discovered the Internet. There was a brief and glorious struggle — well, not all that glorious, really. The dust settled just about when the dotcom bubble burst, in 2000. Each team retreated to its corner, and a cold truce settled on the world.

Perhaps the most widely used on a commercial scale was from IBM: the *Standard General Markup Language* (SGML). It was SGML that invented what we now call *semantic* tags, and distinguished the declared meaning from the rendered formatting style. SGML is still with us today. It's still supported by Jade, and still with us — in humbler form — in XML and HTML. Docbook began as an "SGML application", a standardized set of tags designed for technical documentation. Latterly, Docbook development has favored XML, with its simplified schema-definition language.

The system developed by Bell Labs, *troff*, was as different as could be: a direct typesetter with no imposed structure and no explicit machinery to enforce separation of semantics from style. Instead of dictionaries of opaque tags, troff provided sets of macros that users were free to extend or ignore or abuse. The wonderful, programmable flexibility was a feature to user and a bug to management: freedom for one is to the other loss of control. It should be no surprise that large, top-down organizations such as IBM and the military preferred SGML to troff.

A third rival in the fray came from what might be called an organization: the Free Software Foundation. Richard Stallman developed *Texinfo*, and the GNU project still considers that the One True Format for documentation. Because it is designed only for technical manuals, it bears none of the overhead of SGML and has none of the flexibility of troff. †

Not to be overlooked are language-specific documentation systems and their emulators: java and *javadoc*, perl and *perldoc*, Python and *pydoc*, etc. And of course Doxygen, which might be viewed as javadoc for C++.

† Texinfo can be used as a preprocessor for another system not discussed here in any detail: TEX. TEX was developed not as a *documentation* system, but as a system for producing documents. It has never played much of a role is Unix documentation except to generate printed matter from Texinfo.

Owing to the popularity of JavaScript and GitHub, in recent years *markdown* has become popular. Popular, but simplistic: no cross-references, no table of contents, no index. In the battle for semantic markup, markdown has no dog: it's pure formatting.

Finally, we should mention *mandoc*, which the BSD systems have generally adopted for the base system to render man pages. mandoc exists for that exact purpose: not to replace troff, but to render man pages without it by processing the *man* and *mdoc* troff macros.

Strengths of Documentation Systems

Let's summarize the relative strengths of each system.

SGML → consistency
: SGML divides the informational content from typesetting considerations. By presenting the author only with opaque semantic tags, it reserves for the organization decisions about font and color and layout.

troff → typesetting
: With its macro facility, troff allows, but does not enforce, the use of semantic tags. The author has complete control over every typesetting choice if so desired. troff is the most ambitious typesetter, too: not only tables and footnotes, but mathematics, pictures, and graphs.

Texinfo → accessibility
: Alone among these system, Texinfo includes a reader (two, if you count emacs). Not only does it support an index, but the user can access an indexed item from the command line or by typing I in the viewer.

Doxygen → convenience
: The source-code documentation systems all share the philosophy that some documenttion is better than no documentation, and that programmers are more apt to update the documentation if it's staring them in the face when they change the code.†

markdown → simplicity
: Nothing (almost) to learn, easy to read. Get something up in a hurry, and don't worry about consistency or the printed page.

mandoc → just man pages
: mandoc has only one objective: to render man pages. Well, two objectives: also to render the man page using software with a BSD license.

Feature Set Comparison

System	VT-100	HTML	PDF	Semantic tags	Indexed	Hyperlinks
SGML	no	✓	✓	✓	no	✓
groff man	✓	meh	✓	✓	no	✓
groff mdoc	✓	meh	✓	✓	no	✓
Texinfo	✓	✓	✓	meh	✓	✓
Doxygen	no	✓	?	meh	✓	✓
markdown	meh	✓	no	no	no	✓
mandoc	✓	✓	✓	✓	no	✓

Myths of Documentation Systems

Documentation is easy, or should be. To write documentation is to set aside telling the computer what to do, and to take up the reader's point of view: What does it do, and how do I use it? No *system* can make that transition easy; it has to be accepted, undertaken, and practiced. It's not for everyone, and it's

† The evidence in support of that proposition is nil or negative. Lousy documentation based on such system abounds, and excellent results are rare indeed. Observe that Unix itself was documented with separate documentation files, the method still used by the BSDs today, to generally positive reviews. If lazy programmers are reluctant to write documentation, making the effort less onerous, on the evidence, does nothing for the quality of the results.

definitely not for the programmer while programming.

Write once, render everywhere. The *promise* of many systems is that, if the author will only write the fine manual, the system will repay the effort by rendering it in every desired format and venue. Your Name In Lights. The reality is different:

- The quality of the rendered output is uneven. *No* system today is equally good at producing PDF and HTML output, and rendering usefully in a terminal window.
- The properties of each medium are different, and affect how the content is best presented. Book publishers give consideration to how text and figures relate on the printed page. Sidebars are helpful, except in a terminal. Floating menus can work wonders in HTML, not so much on the printed page.

Proof: rare is the document that renders equally well in two formats out of three. Nonexistent is the one that does so without much thought and effort on the part of the author. It wasn't written *once*; it was written many times, carefully.

In planning a better documentation system, there is no media-neutral solution. Without considered effort, *every* document will be inherently biased to work best in one medium or another. It's useful to understand that your preferred medium influences what you think is the "best" system.

Indexing is automatic. Good documentation is curated, and no system indexes keywords automatically. Sure, the whole text can be indexed indiscriminately. But the job of deciding where a term is defined and discussed remains, in 2018, the job of an editor.‡

The web is the future. The web is not your machine; it's not even you virtual machine in the cloud. The most reliable documentation for the system you are using is that which is included. Furthermore, index- and keyword-searches are not supported by the general-purpose web browser, nor by standard HTML.

troff is obsolete. troff refuses to go gentle into that good night. The "blame" is usually placed on the existing man pages: who will rewrite them in the new, "modern" system? But the fact is that new man pages are written every day, not least because troff is so darn functional.

Myths of Troff

troff is dead. There are three active troff projects: GNU, Heirloom, and neatroff. That excludes the active projects trying to *make* it obsolete.

troff is inferior to TEX. Science students are generally advised to learn TEX, and told it's the cat's pajamas for preparing manuscripts. And it might be; Knuth developed TEX in part because of his dismay at the quality of the output of mathematical notation in troff. On the other hand, modern TEX systems are very complex, 100 times the size of any troff implementation, and the technical advantage of TEX is easy to overstate. I challenge the reader of the present document, for example, to find a paragraph that would be better rendered by the paragraph-at-once algorithm instead of the line-at-a-time one that was used.

As an author, troff is ugly and hard to use.

Compared to what? Pound for pound, troff is easier to use and more capable than anything else.

The best SGML systems are enormously complicated. And expensive. They support great catalogs of tags and transformations, plus a rich UI to let the user connect the abstract tagging sytem to a concrete concurrent rendering.

Nothing resembling those SGML systems is available as free software. The UI presented by DocBook is, for the most part, the same one that is presented by C++: a text editor, a command-line utility to "compile" it, and a web browser or PDF viewer. No Docbook author revels in the sea of tags that overwhelm the content.

‡ An editor with a nose, not an init file.

```
<table id="tab.Protocol.by.Product">
    <title>Versions of the <acronym>TDS</> Protocol, by Product</title>
    <tgroup cols="3">
        <thead>
            <row>  <entry>Product</entry>
                <entry>TDS Version</entry>
                <entry>Comment</entry>
                </row>
            </thead>
        <tbody>
            <row>
                <entry>Sybase before System 10,
                       Microsoft SQL Server 6.x</entry>
                <entry>4.2</entry>
                <entry>Still works with all products,
                       subject to its limitations. </entry>
```

troff was and is the only typesetting language designed expressly for the user to input “by hand”, without the aid of a sophisticated UI and tag dictionary. In constrast to the verbosity of SGML, troff is minimalist. The same table fragment above is entered in troff as:

```
.ds TDS \f[CW]TDS\f[]
.SH
Versions of the \*[TDS] Protocol, by Product
.PP
.TS
LB LB LB
L  L  L  .
Product   TDS Version    Comment
_
Sybase before System 10, Microsoft SQL Server 6.x\
     4.2\
     Still works with all products, subject to its limitations.
.TE
.NL
```

Half the typing: 229 characters versus 586 for Docbook. And, by minimizing the markup, it’s easier to see the forest for the trees.

Without doubt, the idiosyncrasies of troff are off-putting. ’Twas ever thus.

> *In spite of some obvious deficiencies — a rebarbative input syntax, mysterious and undocumented properties in some areas, and a voracious appetite for computer resources (especially when used with macro packages and preprocessors like EQN and TBL) — TROFF has been the basis of document preparation at Bell Labs for some years, and is likely to remain so for years to come.*
> — Brian Kernighan, *A Typesetter-independent TROFF*

The new troff user is immediately affronted by the “rebarbative syntax”. Like parenthesis in Lisp and whitespace in Python, though, the leading dots and two-letter commands† pose no real problem, and quickly become “just the way it is”. I have yet to meet the user who spent several hours with troff and then swore off it because of its offensive syntax.

The Trouble with Troff

The trouble with troff is in the *experience.*

† technically, *requests*, or macros.

Development of the troff UI ceased, along with just about the whole Unix UI, 20 years ago. The last specialized man page viewer was *xman*, a toy from the MIT Athena project. Computing power has increased by 3 orders of magnitude, and all we got was a really fast **less**(1).

Lack of a convenient viewer might be *the* central reason for documentation fragmentation.

Appearance

The author does not breathe who does not care how the prose is rendered on the page. If appealing output didn't matter, you'd be reading this page in a monospace font on greenbar paper, and the Zen CSS Garden wouldn't exist.

Convenience

While typing *man foo* is surely the fastest way to get to a document, it's far from the best way to peruse documentation. Try *man X* or *man git*, for example. Hyperlinks have their place and purpose, and their state of the art today is undeniably in HTML.

But, really, is that enough? Can we ask no more of our documentation system than that new pages come up when we click on the blue underlined text?

The Ideal Documentation System

Man pages look the way they do, and are used they way they are, because the available software mostly emulates the state of the art circa 1976.

Close your eyes now and imagine a documentation system that does what you want. What does it look like? A bit like an e-book reader, but smarter, and with a keyboard.

Fast It finds the page quickly, scrolls quickly, and searches quickly.

Accurate Index searches locate definitions and discussions, not incidental uses of the same word.

Convenient Hyperlinks to internal and external references can be followed at a keystroke or mouse click. *See the description of shopt below under SHELL BUILTIN COMMANDS* isn't just a suggestion; it's an affordance. The table of contents let you jump to the relevant section. Index searches can jump to the referenced use, or display a list of context sentences that are themselves links into the body of the document. The SEE ALSO section can be searched concurrently with the main document.

Attractive The full faith and credit of the GUI is behind it. Colors and font sizes are configurable. Gone are monospace fonts, except where they add information or make the text easier to read.

Flexible There are different ways and needs to read a document. Sometimes it's the first time. Sometimes it's time to dig in and really understand. Sometimes it's a quick lookup of an option or keyword.

Consistent Consistent document structure, consistent document placement, consistent document format, consistent document rendering. The more consistent is the documentation, the easier it is to access, to understand, and to use.

Graphical For those times when a picture or a diagram or an equation helps tell the story.

Integrated If documenting a command, where is the executable, and what does *it* say about its version? If a function, where is the object code, and where is the supporting source?

Strengths of Troff

GNU troff (groff) already provides all the artifacts needed by the ideal documentation system. It can output plain text, HTML, Postscript, and PDF directly. There are macro sets for books, reports, manuals, and presentations. It supports HTML and PDF hyperlinks.

Man pages, while they vary in quality, already do contain all the information needed by the ideal documentation system. The syntax — whether command-line options or argument functions — is fully described.

Much of the power of troff currently goes unexploited. Unlike *any* other freely available documentation system, troff has built-in support for mathematics and graphics. It's also quite trivial to construct tables

automatically from database queries, even on the fly.

Arresting Observation

> In 30 years, no *better* system has been invented. Equivalent systems, arguably, perhaps, depending on how one rates the missing and added features. But no system completely subsumes the feature set, let alone improves on it.

That fact alone is a testament to the durability and utility and functionality of troff.

Better Documentation Through Troff

A better viewer would motivate better man pages.† As things stand, any discussion of "better" tags is academic: while it may be clearer, in some sense, to say that a bit of text is an argument or a function name, knowing the system will underline it or boldface it, it's also one more level of indirection to learn and use. The user sees no difference. If, on the other hand, the system somehow used the fact that the name is an option name and not merely emphasized text, the author now has a concrete reason to provide the information.

Other systems could be brought into the fold. Programmatic conversion of Texinfo, for example, is already supported and could doubtless be improved. The content is there, even if the approach is inconsistent.

The language-specific systems, such as javadoc, can certainly be converted to troff man pages with a certain amount of patience and willingness to make assumptions. Continuous, automatic conversion is probably not feasible; a one-time conversion with touch-ups is.

Arbitrary formats usually in fact have sufficient information to make a creditable conversion to man page format. Your humble author produced man pages from Subversion help text. Someone else converted SQLite's web documentation to man pages. The standard structure of a man page lends itself to programmatic generation if a little boilerplate is tolerable.

My Kingdom for a Viewer

The first step in correcting any problem is recognizing it. No one who has examined the state of Unix documentation would defend it as satisfactory, much less ideal. For that last 15 years, the answer to the way forward has been hands in pockets while whistling a happy tune.

Let's not keep our light under a basket anymore.

Despite being the best we have, such as it is, troff has untapped capacity. That capacity goes unrealized because it is, literally, unrealized: it almost never appears on screen. To the extent that troff's typesetting capacity is revealed in PDF form, it's also *trapped* there, in a system that is slow to load and render, ill-suited for searching, and generally not integrated into the rest of the system.

Put **man**(1), **info**(1), **xman**(1), **xpdf**(1), and your favorite web browser into a blender. Render man pages in proportional font directly, not through Postscript or PDF, to save time. Assign keys to jump to section headers. Automatically search the SEE ALSO section. Make it easy and enjoyable to follow links, forward and back again.

In fact, today's browser has the functionality to interpret and render troff documents. No need for HTML or PDF.

Build it, and they will come. A better viewer will engender better documentation, and a better user experience. One day, perhaps they will say:

I switched to your system because at least I could understand it.

---

† The *mdoc* macro package has much better support for semantic tags (and therefore more *meaningful* tags) than the "standard" *man* macros that GNU and Linux half-heartedly recommend.