Forget reusability, aim for perfection

New lessons from mandoc

BSDCan, Ottawa, June 8, 2018

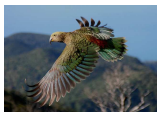
Ingo Schwarze <schwarze@openbsd.org>

EuroBSDCon, Paris, Sep. 24, 2017

Paris, Louvre, vue du Musee d'Orsay

Mandoc at previous conferences:

BSDCan 2011



BSDCan 2014

EuroBSDCon 2014

BSDCan 2015

EuroBSDCon 2015

The context of this talk

I occasionally present updates on documentation tools at BSD conferences.

General reminders about documentation

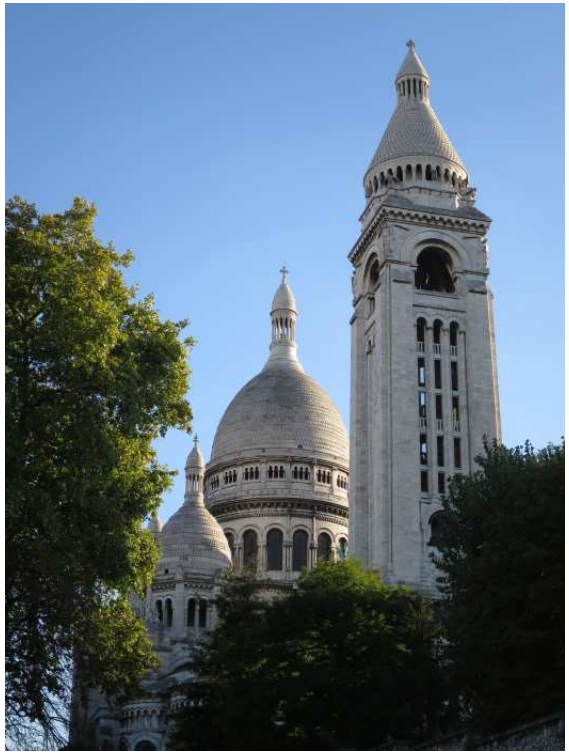
- Without documentation, code is unusable,
  and bad documentation is about as bad as bad code.
- Documentation must be correct, complete, concise, all in one place,
  marked up for display and search, easy to read, and easy to write.
- All BSD projects use the mdoc(7) markup language because it is by far the best language available: concise, simple, providing the right amount of semantic markup. Thanks to Cynthia Livingston. (USENIX, UC Berkeley CSRG 1990 to 1994)
  texinfo(1) and DocBook are excessively complicated, ill-designed, and unmaintained, DocBook also buggy as hell and sluggish; man(7), perlpod(1), and markdown provide no semantic markup.
- All BSD projects use the mandoc(1) toolbox because it is functional,
  free (no GPL), lightweight (no C++ or XML), portable, small, and fast.
  Five input formats, five output formats, two converters,
  very powerful searching, all integrated.
- See my presentation at BSDCan 2015 regarding how mandoc(1) became the standard toolbox. (and those at BSDCan 2011 and 2014, too)

The plan for this talk

Five different lessons from what was done with mandoc in 2016–2018.

But this talk is not about mandoc only:

1. Forget reusability:
   Why we deleted SQLite from the OpenBSD base system.
2. Document the inscrutable:
   LibreSSL and API design.
3. Use the strength of mdoc(7):
   Manual pages on the web.
4. Easily cope with language design from hell:
   The markdown output mode.
5. Serve the real world:
   Improvements for manual pages in ports.
6. Aim for perfection:
   The small things matter.
7. Summary:
   Completed and open tasks. Mandoc adoption.



Paris, Sacré-Cœur (2017)

The problem with SQLite

- Frequent releases: on average 5 feature and 6 bugfix releases per year. (2011–2017)
- Extensive changes in each release: about +15k −5k LOC to audit per year in 2015–2017, counting the directory src/ only. That's half the volume of the complete mandoc codebase — but every year anew!
- That volume of changes is impossible to audit.
- Also, the coding style is so radically different from OpenBSD that people would be unwilling to audit the code even if they had the time.
- Forking would be an absurd waste: SQLite upstream provides excellent maintenance and quality.
- So it is the ideal software for ports: trustworthy and stable without us checking and continuously re-checking all the details.
- But unfortunately, mandoc used it in base...

That's massive: Nantes, Fosse du Chateau (during p2k16)

What was the root of the problem? →

The root cause of the problem

Schematic, superficial approach to architecture.

Antipattern:

- What kind of task is at hand? — Database. (not completely wrong)
- Which is the simplest and highest quality standard toolkit for that kind of problem? — SQLite. (completely true)
- So reuse that as a dependency. (wrong, premature conclusion)

Three errors:

1. Requirements were not evaluated in detail, but only summarily → selection of excessively powerful, large, heavyweight tools.
2. Integration costs were not evaluated → the tool may save time and code lines for the functionality itself, but may require wasting effort on glue code.
3. Maintenance costs were not evaluated — and it turned out that maintenance was impossible, see the previous slide → that was the fatal error which forced us to redo the work.

The actual requirements

Frequent Reading
: Keep that in mind for the design.

Only specific query types
: ... whereas a strength of SQL is flexibility of queries and good performance on any kind of query, even those that were never anticipated.

Limited database size when reading
: ... whereas a strength of a full-scale database is scaling to huge database sizes.

Rare writing, only at system update and pkg_add times
: ... whereas a strength of a full-scale database is efficient writing.

Limited database size when writing
: ... whereas a strength of a full-scale database is scaling write performance.

Linear search is good enough
: ... whereas a strength of a full-scale database is optimized searching, for example using index and hashing techniques.

The most common forms of queries are apropos(1) substring and regular expression searches.
Even the simplest search optimizations like binary searching or hashing are hopeless with that.
Speed optimization for man(1) lookups would be possible but useless, it is very fast anyway.

The solution for mandoc: searching pages

Dedicated database format from scratch with minimal structural overhead.

A very fast and simple way to repeatedly read
the same parts of a file of moderate size from disk:

Use the kernel's buffer cache by simply mmap(2)ing the file into RAM.

Keep the data typically searched together contiguous, as close together as possible, to minimize the number of pages actually faulted into RAM.

For example, the list of one-line descriptions (used for apropos) looks like this:

```
ACME client\0convert addresses into file names and line numbers.\0
format floppy disks\0apply a command to a set of arguments\0...
```

Trivial lookup algorithm

- Do a linear search for the substring.
- When the N-th title matches, ...
- ... access the data struct for the N-th page.

Lookup: Nantes, Notre Dame, vue par les Anneaux (2016)

The solution for mandoc: access search results

For information retrieval, the file contains a list of records of pointers;
in C code, each record can be accessed as a struct.

For example, the record for the ACME client page simply contains the pointers:

- to the page name in the page name list ("acme-client")
- to the section in the section list ("1")
- to the architecture in the architecture list ("")
- to the one line description ("ACME client")
- to the filename ("man1/acme-client.1")

So, a command like “`apropos acme`”:

1. Maps and linearily searches the name and description lists.
2. Looks up the N-th record in the table of manual pages.
3. Follows the pointers to the name and description (already cached)...
4. ... and to the section (will be faulted into RAM at this point).
5. Assembles the line "acme-client(1) — ACME client" and prints it.

The solution for mandoc: program execution

The ktrace(1) is very short an clean:

1. let ld.so(1) load libc, libz, libutil
2. pledge(2)
3. open(2) the file mandoc.db(5)
4. mmap(2)
5. search (not even visible in ktrace, purely userland)
6. access(2) to validate the resulting file name
7. retrieve the record (not even visible in ktrace, purely userland)
8. write(2) the result line

Similarly for semantic searches:

One table for each macro key.

The full format is documented in mandoc.db(5).

Nantes, Maison de L'Administration Noevelle (2016)

The solution for mandoc: summary

- This may sound like "heavily optimized for performance".
- But no, quite to the contrary, it is actually heavily optimized for simplicity and readability of code.
- Performance is merely a by-product of choosing a well-adapted data format and the simplest possible algorithms.

Nantes, Ste. Anne et la Grue Titan Jaune vue de la Cale 3 (2016)

Performance of the new mandoc.db(5)

- Half the database size.
  For example for OpenBSD 6.3
  /usr/share/man/: 4.35 → 2.05 MB
- Double lookup speed.
  For example, for
  “`man -w pledge`”
  on my notebook:
  from disk: 2.7ms → 1.2ms
  from buffer cache: 2.1ms → 0.9ms

Nantes, Jardin des Fonderies,
Rue Louis Joxe (2016)

- Small increase in the database rebuild time.
  For example for /usr/share/man/ on my notebook: 4.1s → 5.1s
  We don’t care that much about rebuild times...
  Doubling it would be unfortunate, but 25% is not an issue.
- Substantially slower database update:
  For example, add one page to /usr/share/man/ on my notebook: 7.5ms → 330ms
  That is 50 times longer (because SQLite efficiently inserts data, while the new makewhatis(8) just reads in the whole file, manipulates the data structures in memory, and writes out the whole file again) — but, honestly, how does less than half a second matter when doing system updates or installing new packages?

Source code

	SQLite	new code	change	
database *.c LOC	205,000	1,570	−99.2%	
database *.c size	7,200kB	44kB	−99.4%	
database *.h LOC	10,700	225	−97.9%	
database *.h size	508kB	12kB	−97.6%	
makewhatis glue LOC	365	125	−65%	now 2350 LOC
apropos glue LOC	510	475	−7%	now 850 LOC

makewhatis(8): Practically no change to the bulk of the code (file system iteration, mdoc(7) node handling, man(7) and *.cat page handling).

apropos(1), man(1): Search and lookup code is now substantially different, but not larger, even though it still does macro specific searches and still supports logical operations like "and", "or", and parentheses.

Both: the new code is much easier to read — constructing command strings of one programming language (SQL) in another programming language (C) is not the kind of code you want to audit.

Plus beau: Nantes, Rond-Point du Pont Willy Brandt (2016)

Immediate benefits of the new mandoc.db(5)

- Deleted SQLite from the OpenBSD base system. (sthen@ 2016 Sep 23)
- 200,000 lines of code less to maintain.
- 15,000 lines of new code less to audit per year.
- Half the database sizes.
- Double lookup speed.
- 300 lines less of very ugly glue code in mandoc itself. (2016 Aug 1)

Costs

- Much slower update time
  (but still below half a second).
- Slightly slower rebuild time
  (5 instead of 4 seconds).
- Had to write slightly below
  2000 lines of new code —
  now two years old,
  needs almost no maintenance.

Nantes, Maison de l'Administration Nouvelle (2016)

What did we learn from the SQLite removal? →

Generic lessons from the SQLite removal

- Do not blindly use standard tools.
- Evaluate specific requirements before selecting tools, do not fall for buzzwords.
- Evaluate integration costs before deciding.
- Try to estimate maintenance cost before committing to a tool.
- Seriously consider using the POSIX C library as your main toolkit:
  It is surprisingly powerful.

  In a long-term project that sees substantial use, the additional effort required for using pure C is sometimes justified by the resulting simplicity, self-containedness, and maintainability, even when performance is *not* the main concern.

Nantes, Notre Dame, vue de la Grue Titan Jaune (2016)

- Quality is multi-dimensional, so even if a tool is excellent in many respects, it may not be good enough.
- Quality does not imply adequacy, so even if something is excellent with respect to **all** its goals and the goals cover all your needs, it may not be good enough.
- "Light" is a relative statement and may not be light enough.

Of course, all this applies to long-term maintenance of public software used by many people — **not** to private, quick and dirty sysadmin scripts.

What about LibreSSL? →

LibreSSL motivation

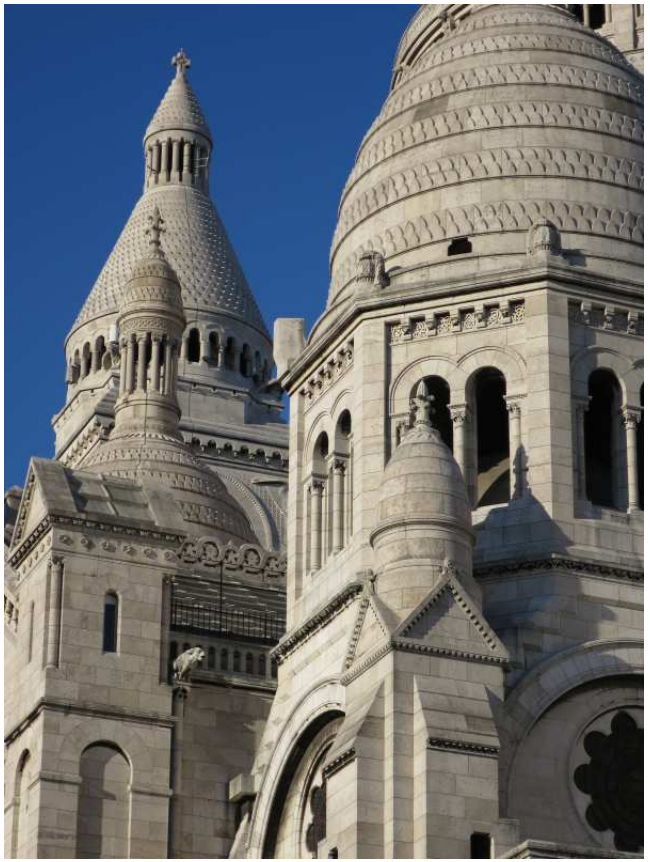
- Forking LibreSSL from OpenSSL was triggered by the CVE-2014-0160 "Heartbleed" vulnerability.
- But the real reason was that inspecting the codebase revealed a general neglect of basic security practices, not just one single vulnerability.
- A second reason for forking was frequent failure of the OpenSSL team to cooperate when patches were sent.
- Once forking was decided, everything happened very quickly, see below.
- Initial focus was on deleting needless code, and on preparing the code for audit; later on code auditing, improving robustness and security, and the new libtls.

April 7	Damien Miller	cherrypicks the heartbleed fix from OpenSSL
April 8	Ted Unangst	"exploit mitigation countermeasures" mail on tech@
April 13	Miod Vallat	imports OpenSSL 1.0.1g
April 13	Theo de Raadt	first Valhalla commit, s3_lib.c rev. 1.22
April 13	Bob Beck	follows within less than 2 hours
April 14	Joel Sing	starts applying KNF to the OpenSSL code
April 15	Ted Unangst	removes FIPS mode support
May 17	Bob Beck	officially announces LibreSSL during BSDCan 2014

What was the task with respect to documentation? →

LibreSSL documentation: the task

- Just like the OpenSSL code was way below OpenBSD quality standards, so was the documentation.
- Incomplete, generally sloppy, and an inferior markup language.
- Just like the code needed reformatting before audit (KNF, style(9)), so did the manual pages.
- But: the code could remain in C, (even though Boring SSL switched to C++) while the manual pages had to change markup language: perlpod(1) → mdoc(7).
- Semi-automatic code reformatting allowed the check "no object change"; no such check was possible for manual pages → manual work.
- Long delays because the manual work required is very substantial.



Paris, Sacré-Cœur (2017)

Which tool was used? →

LibreSSL manual page conversion: the tool

- Tool used: pod2mdoc(1), a small, self-contained C program.
- Started by Kristaps Dzonsons (2014 Mar 20 to Apr 7) —
  by chance, exactly during the two weeks before heartbleed.
- Some steps forward during the next two years, with long pauses.
- Main difficulty: convert presentational to semantic markup, requires heuristics.
- Real work started during g2k14: some code improvements by schwarze@,
  first 83 pages converted by bentley@. (2014 Jul 11 to 19)
- The Ft/Fo/Fa/Fc heuristic formatter for the SYNOPSIS. (2014 Oct 22)
- Use ohash(3) to improve markup,
  convert another 45 pages.
  (2015 Feb 12 to 23)
- Complete conversion during l2k16:
  the last 130 pages. (2016 Nov 2 to 6)
- The tool was already in good shape
  since 2015, almost no more changes
  were needed during l2k16.

Paris, Louvre, Pavillon Flore, vue du Pont Senghor (2017)

Work done by hand during the conversion

- Almost always: missing macros, no markup in the original.
- Often: macros not automatically recognized (phys to sem, Vt, Fn...).
- Occasionally: fix markup that was poor in the original.
- Occasionally: unusually difficult markup, like for callback functions.
- A few technicalities, like removing useless character escapes.
- All required reading of the complete text by a human.

While here, do initial cleanup of content

- Delete inapplicable or useless text.
- Apply wording tweaks.
- Improve SEE ALSO sections.

It is technically desirable to keep content changes separate, but that would be a waste of effort: both cleanups require a human to read the complete text.

After that, linting and semi-automatic checks were done: for example, several functions in libcrypto were documented in more than one manual page.

Paris, Louvre, Cour Marly (2017)

Initial synchronization with OpenSSL

- Systematically work through all OpenSSL manual pages.
- Add Copyright and license to each file. (2016 Nov 10 to Dec 10)
- Bring in bug fixes from OpenSSL.
- Add missing pages from OpenSSL to our tree. (2016 Nov 10 to Dec 11)
- Remove inapplicable stuff. (e.g. 2016 Sep 5 CMS)
- While reading the text again, watch out for various kinds of bugs.
- Reorg overview pages: BIO BN DH DSA EC RSA ssl (2016 Dec 6 to 11)
- Fix d2i* pages. (2016 Dec 24 to 2017 Jan 6)
  All in one page in OpenSSL. Ideally, such pages should say something about the actual format but at least clearly specify what the object is.

Paris, Les Docks, Quai d'Austerlitz (2017)

Maintenance tasks

- At irregular intervals, evaluate all changes in OpenSSL since the last sync.
  (syncs started 2016 Dec 10; 2017 Mar 25, Aug 19; 2018 Feb 12, Mar 29, ...)
- Merge changes that make sense and apply to our code.

Maintenance related to local code changes

- Split tls_init(3). (2017 Jan 25)
- Start merging OpenSSL-1.1 interfaces. (2018 Feb 10, with jsing@ and tb@)
- Start constification. (2018 May 1, with tb@)

Diverse major quality improvements

- Write HISTORY sections. (2018 Mar 20 to 27)
- Systematic checks of openssl(1). (started 2018 Mar 30, incomplete)
- Rewrite ENGINE manuals. (2018 Apr 14)

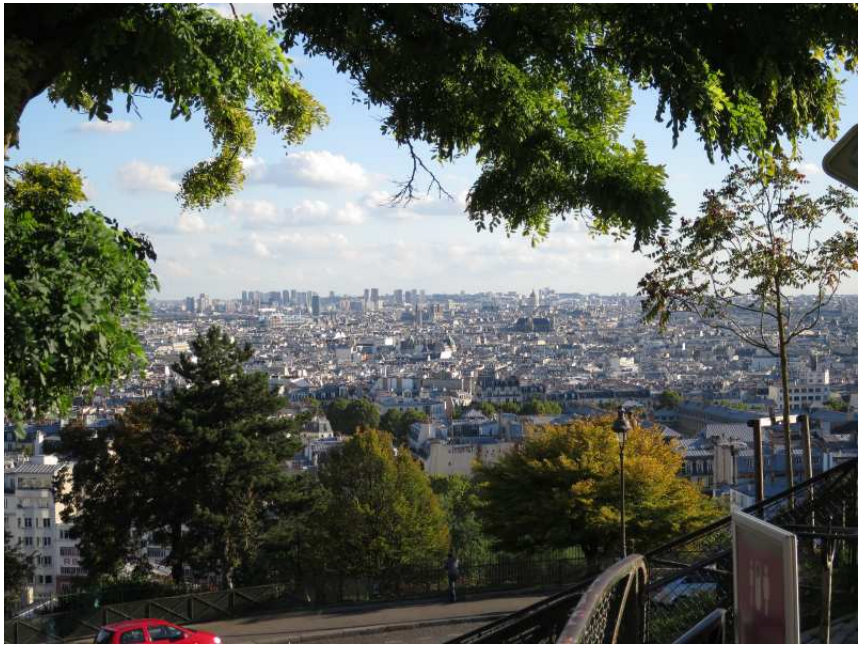
In several cases, reading code in order to document it revealed bugs that got fixed, for example in X509_NAME_add_entry(3). (2018 Apr 4)

Pages written from scratch: the problem

In a few cases, missing pages were written as soon as the lack was noticed; earliest example: BN_set_negative(3). (2016 Nov 5)

Not in general, both because too many pages are missing, so it would have derailed and blocked the rest of the work, and because in some cases, functions are intentionally undocumented, and identifying these cases is non-trivial.

Some classes of functions that could be safely documented without the risk of accidentally exposing internals:



Paris, vue du Montmartre vers la cité (2017)

Pages written from scratch: progress

- Functions referenced elsewhere in the manuals:
  e.g. 11 new pages in libssl (2016 Dec 6 to 10)
  X509_STORE_load_locations(3) (2017 Jan 6)
  OPENSSL_sk_new(3) and STACK_OF(3) (2018 Mar 1)
- ASN1 and X509 constructor manuals. (2016 Dec 12 to 2017 Jan 4)
  Public objects always require at least constructor documentation.
  Explain not just what the constructor technically does, but what the meaning of the constructed objects is, with refs to STANDARDS etc.
- Functions analogous to documented functions,
  e.g. SSL_set_tmp_ecdh(3). (2017 Aug 12)
- Public functions with semantics that substantially differs from OpenSSL,
  e.g. ASN1_STRING_TABLE_add(3). (2017 Aug 20)
- Pages where OpenSSL manuals are seriously misleading,
  e.g. X509_check_private_key(3). (2017 Aug 20)
- Only one systematic effort so far to get a particularly important sub-library completely documemted — but even that is still incomplete: two new pages and two new functions in existing pages in BN documentation. (2017 Jan 25 to 30)

The current state of affairs: what is still missing

Much better than OpenSSL (all improvements merged, large numbers of additional bugs fixed, many substantial content improvements, many new pages) — but:

- Many public functions lack manual pages in LibreSSL. Write them from scratch or add comments saying why they are intentionally undocumented.
  Many months of full-time work (at least)...
- Almost all existing pages need basic copy-editing, they are in general wordy, imprecise, and incomplete. Fixing them usually requires comparing the text to the code, but the code is often contorted, so reading it is time-consuming.
  Many months of full-time work (at least)...
- Complete the first pass through the openssl(1) manual page. Lack of motivation due to low code quality, it's basically a quick and dirty testing tool, and it is of considerable size. Probably about a week of full-time work, maybe more...
- Routine syncs with OpenSSL. These will become a serious problem in the near future when OpenSSL changes their license and becomes non-free (Apache 2).
- And of course, the usual ongoing maintenance when code is added, deleted, or changed in the future, like for any other code and documentation.

Paris, Sacré-Cœur (2017)

Lessons learnt about LibreSSL and API design (1)

- Use standard POSIX functions. If you want, provide fallback implementations for defective operating systems. If you can't, accept the truism that running defective operating systems implies limited functionality. Never design your API to cater for the worst possible system you can find. That makes everybody suffer from idiosyncrasy and bloat. For example, the BIO abomination was originally designed to deal with shortcomings of Microsoft Windows.
- Avoid wrappers around POSIX functions. Exception: in application code (but not in a library), a wrapper around malloc(3) that calls err(3) on failure is OK and can make the main code more readable and less prone to errors.
  Bad example: OPENSSL_malloc(3), CRYPTO_malloc(3) — both exist!
- Follow C library semantics a much much as possible, lest you cause misunderstandings, bugs, and the need for confusing warnings in the documentation. For example, ASN1_STRING_cmp(3) ought to do lexicographical ordering like strcmp(3), but it does not. As a last resort, if you must be different for some reason, use a clearly different name.

Paris, Notre Dame (2016)

Lessons learnt about LibreSSL and API design (2)

- Minimize the number of public API functions. With too many functions, some functions will likely remain undocumented for lack of time to write the text. With too many functions, users will lose their way and fail to find the the function they need. It is harder to keep a large number of functions consistent than a small number. Particularly bad example: PEM_read_bio_PrivateKey(3).
- In particular, avoid families of nearly identical functions differing only in one minor aspect. They are almost impossible to document properly, it is very hard to avoid that the documentation becomes both vague and repetitive.
  Just design your API in some different way.
  Bad offenders: see CRYPTO_set_ex_data(3), BIO_set_ex_data(3).
- In particular, if the prototypes agree with each other, make them one function, with the behaviour controlled by one of the arguments.
  Bad example: DES_ofb64_encrypt(3) and its companions.
- Be wary, even when doing so, you can still create absurdly large APIs; for a particularly bad example, see EVP_EncryptInit(3). For complex tasks, avoiding such failure is not trivial and requires careful and disciplined design.
- Never create families of functions differing only in one type and one component of their name. Never generate function names from preprocessor macros.
  Such functions are almost impossible to document.
  Particularly bad examples: STACK_OF(3), lh_new(3).

Lessons learnt about LibreSSL and API design (3)

- Minimize the number of objects in object oriented code. Each object needs at least one manual page, usually more than one if it is non-trivial. Constructor manuals usually add a lot of volume with relatively little useful content.
- Avoid representing the same logical entity on two different API levels. For example, ASN.1 objects ought to be either represented as NIDs troughout (with utility functions to retrieve names etc.) or as objects (structs) throughout. OpenSSL provides both, also resulting in duplicate sets of accessors and in duplication in several other interfaces.
  See OBJ_nid2obj(3) for the ugly consequences.
- Avoid redundant interfaces. For example, if your objects have child objects, either provide copying accessors only and require users to *always* free the retrieved copies after use. Or provide accessors only that return references and require users to copy them themselves when needed. Providing both causes confusion, invites bugs, and bloats the interface and the documentation.
- Avoid cryptic naming conventions like get0, get1, add0, add1. They require users to learn additional rules and make understanding harder than necessary. If you feel tempted, your interface is getting too complicated.
- Be consistent whether copies are deep or shallow; it’s a major source of confusion and bugs. OpenSSL is inconsistent and rarely even documents it.

Lessons learnt about LibreSSL and API design (4)

- Never provide public accessor that can break invariants, like ASN1_STRING_length_set(3). They are a sure sign of failed interface design.
- Think twice before using callbacks; they make interfaces and documentation significantly more complex and massively obstruct call tree analysis during auditing. If you must use them, typedef the public callback prototypes. A SYNOPSIS becomes inscrutable when callback prototypes appear verbatim as function arguments or return values. Bad offender: BIO_meth_get_read(3).
- Avoid object flags radically changing the behaviour of the object, for example ASN1_OBJECT_FLAG_DYNAMIC or BN_FLG_CONSTTIME. They cause surprising behaviour, invite bugs, and complicate the documentation substantially. Look at ASN1_OBJECT_free(3) and BN_set_flags(3) for particularly bad examples.
- Never define types that are essentially typedefs for "`void *`". They are confusing and merely cause a false sense of type safety. If you really must sacrifice type safety, use "`void *`" directly. Bad counter-example: ASN1_VALUE.

Novi Beograd, Zapadna kapija (West gate) during EuroBSDCon 2016

Lessons learnt about LibreSSL and API design (5)

- Avoid functions that show radically different behaviour depending on input arguments. For example, a function ought to either fill in provided storage or allocate new storage, but not decide itself based on whether it was given a NULL pointer. A function ought to either operate on NUL-terminated strings or on fixed-length char buffers, but not decide itself based on whether it was given a length of -1. Radical behaviour changes cause confusion, invite bugs, and force wordy and unwieldy documentation.
  Particularly bad offender: ASN1_item_d2i(3).
- Good naming is vital for comprehensibility. Absolutely avoid wrong names; they confuse users, invite bugs, and make correct documentation sound wrong.
  For example, the type ASN1_STRING_TABLE is really a *table entry*, not a complete *table*.
  For example, the type ASN1_TYPE does not contain a type at all, but a value of arbitrary type, so it should be called ASN1_VALUE_ANY or similar.
  For example, X509_check_private_key(3) compares the *public* key components only.
- Limit function arguments to reasonable numbers.

Rural countryside: Cambridge, Kings College (2016)

Do these lessons ever end? →

Lessons learnt about LibreSSL and API design (6)

- Get syntax and semantics right when first adding a function.
  Changing syntax or semantics in a later release not only necessitates change of application programs but also complicates documentation. Adding a function constitutes a huge responsibility and is not to be done lightly.
- Keep logging and error reporting as simple as possible.
  I'm not aware of any other subject area so prone to overengineering.
  ERR(3), ERR_get_error(3), ERR_error_string(3), ERR_print_errors(3), ERR_GET_LIB(3), ERR_put_error(3), ERR_set_mark(3), ERR_load_strings(3), ERR_load_crypto_strings(3), ...
- Keep configuration and initialization simple.
  They are also quite prone to feature creep and spaghetti code. See OPENSSL_config(3) and the functions mentioned there for a bad example.

John Brocke, Qol/Voice, 1987

(Calgary, Glenbow Museum, 2015)

What about manual pages on the web? →

Manual pages on the web

About ten important improvements were implemented here in 2016–2018 alone.

Structure and internals

- C code structure cleanup in the HTML formatter, print_otag() reorg. (2017 Jan 16)
- HTML output line break logic. (2017 Jan 18)
- HTML element, attribute, and CSS cleanup. (started 2017 Jan 19)
- Replacement of hard-coded “`style=`” attributes with CSS where possible, for example for “`.Bl -compact`”. (2017 July 14)
- Don’t print an embedded style sheet if an external one is referenced. (2018 May 1)

Features

- The HTML <title> element now shows the name and section number of the manual page. (2017 Mar 13)
- When using the search form, redirect to concise URIs of the form:
  http://man.openbsd.org/[manpath/][arch/]name[.sec]
  The optional parts are omitted whenever possible.

Manual pages on the web: features

- catman(8) and mandocd(8) (2017 Feb 4, with Michael Stapelberg)
  Both manpages.debian.org and Arch Linux now use the mandoc formatter for their official online manuals.
- Deep linking into manual pages: (2017 Mar 15)
  To almost the same places as the less(1) :t tags on the terminal.
  Implemented with "`id=`" attributes.

  Results in concise, human-readable URIs like:
  https://man.openbsd.org/mmap.2#MAP_STACK

  Dotted underline in HTML+CSS output, hover to cut and paste the URI.

  In man(7), only for .SH, .SS, and .UR due to lack of semantic information.
- Tooltips show semantic function of marked-up content: (2017 Mar 13)

  That is useful because it may occasionally help understanding the text, because it definitely helps to develop the ability of using apropos(1) semantic search efficiently, and because slowly becoming familiar with the macro keys also helps to lower the entry barrier for users who consider sending patches.

  Implemented with "`title=`" attributes for now, likely to be improved.

Markdown output format

Avoid that people have to maintain two copies of documentation and allow using mdoc(7) even when a project policy requires markdown.

New output mode implemented in just two weeks, part-time. (2017 Mar 3 to 17)

How simple it has become to implement a new mandoc output mode

1600	lines of C code grand total
110	head matter: license, includes, protos, flag defines
140	mdoc macro dispatch table
40	main function: header, main loop over mdoc nodes, footer (straightforward)
60	node driver (incl. text line and roff request formatting)
20	markdown stack handler (for blockquote (>), code blocks (tab), lists)
60	spacing and outflags handling
70	input escape character handling
190	output character escaping
850	mdoc node handlers (mostly straightforward)

Cime de Caron 3193m, Vanoise, France (2017)

Markdown output format — the most difficult parts

Markdown output character escaping:
Totally horrific due to context sensitivity, 190 lines, 12% of the code.

Markdown block nesting:
Somewhat complicated due to context sensitivity, interacts with output character escaping, touches about a dozen places in the code, about 90 lines of code (6%).

Horizontal spacing in the output:
Somewhat difficult in many output formatters; about 160 lines of code (10%).

The bulk of the code (50%) is
straightforward mdoc node handling.

All difficult parts
(except horizontal spacing, which is always somewhat tricky)
are due to quirks of the markdown language,
none are intrinsic difficulties
of writing a mandoc(1) output module.

Is there a way? Stockholm, Järva Krog (during EuroBSDCon 2015)

Markdown: how a markup language should not be designed

Lack of expressiveness:
Goal: easy writing like in plain-text email; yet e.g. no syntax for definition lists.

Context sensitivity:
Almost every token can take different meanings depending on where it appears.

Ambiguity:
Enclosing in asterisks/underscores: long_var_name, **bold***italic***bold**

Mixup of semantic and presentational markup:
No way to switch off filling without <code> tags.
Could be improved, but HTML output from markdown is now fixed by tradition and people's CSS.

Lack of independence:
Allows and requires embedded HTML, but with crippling restrictions.
In unfilled text: no char refs, no flow-level elements, no native formatting.
In indented text: no paragraph breaks, no block-level HTML.

Syntax inspired by Whitespace:
Two trailing blanks mean a line break.

Lack of both standardization and extensibility:
Bad because it lacks features, so everybody adds their own, incompatible ones.

See my essay on undeadly.org for more details.

Do not use markdown:

Use mdoc(7) to maintain your source documents and mandoc(1) to convert them when needed.

Mandoc in ports: how it began

Initially, all manuals in ports
were formatted with groff(1)
at package build time
(USE_GROFF Makefile variable).

Ports were switched to
install source manuals instead
after checking that the manuals
worked well with mandoc(1).

Paris, Institut de France, Quai de Conti (2017)

Three reasons to get rid of USE_GROFF

1. Installing source manuals allows using semantic searching with apropos(1)
   — though so far, that mostly applies to mdoc(7) manuals and doesn't make
   much of a difference for man(7) manuals.
2. Avoiding dependencies simplifies optimization of bulk builds for speed.
3. Getting rid of USE_GROFF altogether
   would take one complication out of the ports build infrastructure.

Mandoc in ports: how it evolved

	USE_GROFF	all ports	
Oct 20, 2010	3080	5750	54%
BSDCan 2011	2850	6150	46%
BSDCan 2012	2516	6967	36%
BSDCan 2013	2180	7790	28%
BSDCan 2014	1210	8260	15%
BSDCan 2015	215	8569	2.5%
BSDCan 2016	199	8858	2.2%
BSDCan 2017	76	8898	0.85%
BSDCan 2018	28	8930	0.3%

Obstacle: Nantes, Boulevard Maurice Bertin, vue vers le Pont Éric Tabarty (2016)

- Progress in 2011–2014 mostly by straightforward implementation of missing low-level features.
- State in early 2015: USE_GROFF remained in about 250 ports.
- First list of reasons for all remaining USE_GROFFs drafted by naddy@ before p2k15.
- Explicit effort to reduce the list duing p2k15 only fixed 22 out of 250 because the top remaining reasons (ta 60, ti 50, \h 30) were almost unfixable.

Which was the latest leap forward? →

Parser unification

Originally, mandoc(1) only had mdoc(7) and man(7) parsers, no roff(7) handling.

Partial roff(7) request handling was added in 2010,
but purely in the form of a preprocessor. (see my BSDCan 2011 talk)

But several roff(7) requests operate in a way similar to macros
and require syntax tree nodes for representation:
producing output (.br .mc .sp) or changing formatter state (.ce .ft .ll .po .rj .ta .ti)

Extensive reorganization

- Unified types: enum roff_type, struct roff_node, roff_meta, roff_man
  (2015 Apr 2 to 18)
- Unified node handling library in roff.c. (2015 Apr 19)
- Unified way to use the ohash library. (2015 Oct 13)
- Seperate the validation phase from parsing. (2015 Oct 20)
- Unified token IDs: enum roff_tok, roff_name[]. (2017 Apr 24)
- Use ohash for all request and macro tables. (2017 Apr 29)

Syntax tree nodes can now be generated on the roff(7) level.

Framework for terminal and HTML output from these roff nodes. (2017 May 4)

New low-level roff(7) features

Generating nodes on the roff(7) level allowed implementing many new roff(7) requests and escape sequences, (ab)used by man(7) pages in ports.

Requests moved from the man(7) parser to the roff(7) parser:

.br .ft (2017 May 4) .ll .sp (2017 May 5)

New requests changing state visible to the formatters:

.ta (2017 May 7) .ti (2017 May 8) .ce (2017 June 6) .rj .po (2017 June 14)

New requests changing preprocessor state only:

.ec .eo (2017 June 3) .rn (2017 June 6) .als (2017 June 14) .am (2017 June 18)

\n auto-increment and .nr step size (2018 Apr 9)

New requests and escapes producing output:

.mc (2017 June 4) \h (2017 June 1 and 14) \l (2017 June 2) \p (2017 June 13)

New state inspectors:

\n[an−margin] (2017 June 13) .if d conditional (2017 June 14)

New man(7) macros:

.DT (2017 May 7) .MT .ME (bentley@ 2017 June 25)

Improvements for tbl(7)

Rather tricky improvements in the tbl(7) formatter.

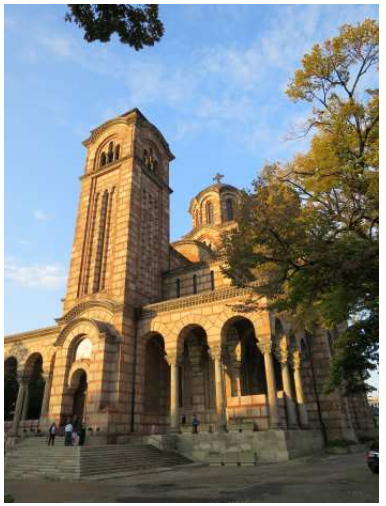
- Filling text inside table columns. (2017 June 7 and 12)
- Table option "allbox". (2017 June 12)
- Layout specifier "w" (column width). (2017 June 8)
- Specification of column spacing in the table layout. (2017 June 27)
- Various improvements regarding data contained in unspecified output cells and regarding horizontal and vertical lines. (2017 June 16)

Paris, Louvre, La Pyramide (2017)

Benefit for ports

Once the framework was in place, we reduced the number of OpenBSD ports that still USE_GROFF from over 200 to just 25 in just two months:

date	ports	main new feature (ab)used in these ports
2017 May 7/8	22	tabulator settings: .ta .DT
2017 May 8	42	temporary indent: .ti
2017 June 1/2	42	horizontal spacing: \h \l
2017 June 3	7	escape control: .ec .eo
2017 June 4	4	margin notes: .mc
2017 June 4	3	centering: .ce
2017 June 4	2	remove number register: .rn
2017 June 12	8	tbl(7) improvements
2017 June 13	6	\n[an−margin], \p, .if d
2017 June 14–29	21	various



Beograd, Crkva Svetog Marka (2016)

Non-English manual pages

- Let pkg_add(1) run makewhatis(8) in /usr/local/man/*lang*/. (2017 May 15)
- Paragraph in the porting guide:
  standard directories, always UTF-8, use iconv(1) if needed. (2017 June 1)

Cooperation across communities

Coordinating with other implementations of the languages and utilities in question:

groff = GNU troff

- ASCII output of special characters (“`tty-char.tmac`”):
  focus on meaning rather than graphical shape. (2017 Aug 22)
- Rewrite groff_mdoc(7) .Lk macro. (2017 Apr 10 to 13 and 2018 Jan 12)
- 3 minor features, 2 formatting improvements, 7 bugfixes, 4 string table updates,
  5 documentation improvements, 16 build system fixes:
  grand total about 40 contributions in 2011–2018.

man−db = GPLv2+ manual page viewer

- For compatibility with the
  man(1) implementation of man−db,
  interpret names containing slashes
  as absolute or relative file names,
  even without the “`-l`” option.
  (2018 Apr 19)

Beograd, Narodna Skupshtina (2016)

But that’s more or less all that was done, right? →

All those small things: infrastructure

Keeping up with newly developed security features:

pledge(2) for man(1), mandoc(1), apropos(1), makewhatis(8). (2016 Nov 15)
pledge(2) for man.cgi(8). (semarie@ 2017 Feb 22)

Moving ahead with regression testing:

New portable version of the mandoc regression suite,
including more than 1000 of the existing test cases. (2017 Feb 17)
Run it iteratively rather than recursively in portable mode. (2017 July 18)

Further improving diagnostic functionalities:

Most prone to overengineering, feature creep, and code sprawl;
consider groff, OpenSSL, SQLite, errno(2), even mandoc(1) itself.
Integration of mdoclint(1). (2017 April 27 to July 3, with wiz@)
Message levels "`-W style`" (2017 May 16 to July 6) and "`-W base`". (2017 June 24)

Getting rid of MLINKS.

Fewer files in the distribution tarballs, simpler Makefiles. (jmc@ 2016 March 30)

Constant bugfixing and usability improvements.

Portable releases:

1.14.1 (2017 Feb 21), 1.14.3 (2017 Aug 5)

All those small things: features

Further improving search:

- Support more than one tag entry for the same search term, tag leading presentational macros in .It, and some more improvements to ctags(1)-style internal searching with less(1) :t. (2016 Nov 8)
- Enable full makewhatis(8) by default in OpenBSD. (2017 Apr 15)

Improving eqn(7):

- Lexer: complete rewrite — simpler and fixing several bugs. (2017 June 26)
- Parser: recognize well-known function names. (2017 June 21)
- Parser: quoted words are not parsed. (2017 June 21)
- Parser: better font selection. (2017 June 21)
- Parser: implement operator precedence. (2017 July 5)
- HTML formatter: use <mi>, <mn>, <mo> in MathML. (2017 June 22)
- Terminal formatter: output disambiguation with parentheses. (2017 July 7)
- Terminal formatter: better horizontal spacing. (2017 Aug 23)
- Use in erf(3) and lgamma(3). (2017 Aug 26)

Occasional performance enhancements:

- For example substantially smaller PostScript output. (espie@ 2017 Nov 1)

Done...

project	announced	completed	presented
install manual page sources	BSDCan 2011	June 23, 2011	BSDCan 2014
implement −mdoc −Tman	BSDCan 2011	Nov 19, 2012	BSDCan 2014
apropos(1), makewhatis(8)	BSDCan 2011	April 14, 2014	BSDCan 2014
replace man.cgi(8)	BSDCan 2011	July 12, 2014	EuroBSD 2014
pod2mdoc(1) for LibreSSL	BSDCan 2011	Nov 6, 2016	BSDCan 2018
implement −man −Tmdoc	BSDCan 2011	abandoned	
integrate preconv(1)	BSDCan 2014	Oct 30, 2014	BSDCan 2015
unify parsers, better roff(7)	BSDCan 2014	in progress	BSDCan 2018
docbook2mdoc(1) for man(7)	BSDCan 2014	stalled	
pod2mdoc(1) for Perl manuals	BSDCan 2014	not yet started	
default to −Tlocale	EuroBSD 2014	Dec 2, 2014	BSDCan 2015
replace man(1)	EuroBSD 2014	Dec 14, 2014	BSDCan 2015
use less(1) tags	BSDCan 2015	July 17, 2015	EuroBSD 2015
delete most MLINKS	BSDCan 2015	March 30, 2016	BSDCan 2018
use texi2mdoc(1) in practice	BSDCan 2015	not yet started	
better less(1) tags	EuroBSD 2015	mostly done	BSDCan 2018

Which tasks are still open? →

Future directions(1)

You might think mandoc is finished, but there is still surprisingly much to do:

Structure — privilege separation:

Do parsing and formatting in different processes.
Stricter pledge(2). Use unveil(2).

Parsers — continue unification:

Represent escape sequences as AST nodes and improve width measurements.
Retain more roff(7) requests as AST nodes, and document AST invariants.
Provide an mdoc(7) output mode (normalization).

Parsers — mdoc(7):

Better support pages that apply to multiple, but not to all architectures.
Disentangle filling and font control for displays.

Parsers — man(7):

Add minimal heuristics
for better linking and markup.

Parsers — tbl(7):

Support macros inside tables.

Parsers — roff(7):

A few requests that are abused in
practice are still unimplemented.

La Grande Casse 3855m, Vanoise, France

Future directions (2)

Formatters — HTML:

Use more fitting HTML elements for some macros, improve CSS style, avoid title= attributes for tooltips, futher reduce style= attributes, solve the problem of duplicate anchors, solve the remaining HTML syntax violations.

Formatters — PostScript and PDF:

Better font support; espie@ started work, but i ran out of time to support him.

Foreign formats — perlpod(1):

Use pod2mdoc(1) for Perl manual pages.

Foreign formats— texinfo(5):

Use texi2mdoc(1) in practice.

Foreign formats — man(7):

Support man(7) to mdoc(7)
semi-automatic migrations
with doclifter(1) and docbook2mdoc(1).

Manual pages:

Write the missing LibreSSL manuals
and clean up the existing ones.
That’s a huge problem due to the volume.

La Daille 1800m, Val d’Isére, Vanoise, France

For many minor issues, see the mandoc TODO list.

Adoption of mandoc

Default formatter and viewer
- OpenBSD (schwarze@), Alpine Linux (Sabogal), Void Linux (leah@)

Default formatter and search tool, but weaker viewer
- FreeBSD (bapt@)

Default formatter, but using weaker view and search tools
- NetBSD (christos@), illumos (yuripv@)

Included by default, but outdated and not used by default
- Dragonfly, Minix 3

Official ports and packages
- Debian (stapelberg@), Ubuntu, Gentoo, pkgsrc (wiz@)

Unofficial ports and packages
- Arch, Slackware, Crux (juef@), MacPorts, MacOS Homebrew

Mandoc for official online web pages

man.openbsd.org, manpages.debian.org, and manpage links on wiki.archlinux.org

Conclusions about API design

- Writing documentation is an excellent way to understand the quality of an API.
- Bad API design can make documentation almost impossible.
- Typical problems: too many functions, cryptic conventions, misleading names, wrappers, redundancy, inconsistent and surprising semantics, excessively complicated logging and initialization, incompatible API changes in the past and resulting compatibility hacks.
- About the worst API design error ever:
  function names autogenerated with macros.
  Completely impossible to properly document.
- Callbacks are very hard to document.
  Avoid them as much as possible.
  If you cannot avoid them, typedef the prototypes.
- Gratuitious use of typedef.
  Never “`typedef struct foo FOO`”.
  Never “`typedef struct foo *foo_p`”.
  Never “`typedef int64_t myint`”.
  Such typedefs seriously obfuscate documentation.

Promote confusion: Beograd, Ulica Svetogorska (2016)

More conclusions... →

Conclusions related to documentation tools

- Mandoc has been the standard BSD toolkit for manuals on the command line since about BSDCan 2015.
- Now, it is also becoming the standard formatter for the web.
  It has extensive support for semantic markup and hyperlinking.
- Mandoc now covers well above 99% of ports due to improved roff(7) support.
- Never try to write markdown documentation by hand. Generate it from mdoc(7).

Conclusions for any kind of free software project

- Do not prematurely introduce dependencies, not even on widely-available, high quality libraries. Evaluate requirements, integration costs, and maintenance costs first. Seriously consider using the POSIX C library as your main toolkit.
- Worrying about performance is vastly overvalued. Good performance is a by-product of pursuing *other* goals, mainly simplicity. Simply choose well-adapted data structures and the most straighforward algorithms, and you usually get performance that is more than good enough, plus excellent maintainability.
- The success of a long-term software project depends on a good balance of features vs. maintenance. In case of doubt, attention to maintenance details (code quality and cleanup, usability details, diagnostics, bugfixes, regression testing, ...) matters more than adding ever more bells and whistles.

Thanks!

These slides only list new contributions since BSDCan 2015.
For complete acknowledgements since the start of the mandoc project,
see my EuroBSDCon 2015 slides.

- Anthony Bentley (OpenBSD) for about ten code contributions and dozens of bug reports, useful suggestions, and discussions.
- Christian Weisgerber (OpenBSD) for lots of work on the USE_GROFF removal and several bug reports and suggestions.
- Michael Stapelberg (Debian) for designing and writing most of mandocd(8) and catman(8) and for more than a dozen patches, bug reports, and suggestions.
- Marc Espie (OpenBSD) for implementing smaller PostScript output and many useful suggestions and discussions.
- Baptiste Daroussin (FreeBSD) for writing soelim(1), for makewhatis(8) performance testing, and for some bug reports and suggestions.
- Jason McIntyre (OpenBSD) for countless useful discussions, suggestions, and bug reports, repeated testing, and lots of copy-editing in the LibreSSL manuals.
- Thomas Klausner (NetBSD) for extensive cooperation on the mdoclint(1) integration and for several bug reports and useful suggestions.
- Joel Sing (OpenBSD) for lots of feedback regarding LibreSSL manual pages.

Thanks!

- Kristaps Dzonsons (bsd.lv) for implementing Mac OS X sandbox_init(3) support and for useful discussions.
- Sebastien Marie (OpenBSD) for contributing code and ideas to pledge(2) support and some useful discussions.
- Jonathan Gray and Theo Bühler (OpenBSD) for a bug fix patch and more than twenty bug reports and suggestions each, many based on afl(1) findings.
- Alexander Bluhm (OpenBSD) for several significant improvements to the mandoc regression suite.
- Todd Miller (OpenBSD) for important help with process group handling, for checking many patches, and for useful discussions and suggestions.
- John Gardner for extensive suggestions regarding HTML output.
- Carsten Kunze (Heirloom troff), Werner Lemberg, Bertrand Garrigues, Branden Robinson, and Peter Schaffter (GNU troff) for help getting patches committed to groff(1) and for useful discussions.
- Theo de Raadt (OpenBSD) for several bug reports, useful discussions, and suggestions, and for checking several patches.
- Florian Obser, Jérémie Courrèges-Anglas, Martin Natano, Philipp Guenther (OpenBSD), Ed Maste (FreeBSD), and Peter Bray for patches, bug reports, and useful discussions.

- Michael McConville (OpenBSD), Kamil Rytarowski (NetBSD), Andreas Voegele, Fabian Raetz, Max Fillinger, and Tiago Silva for patches.
- The OpenCSW.org team for access to the Solaris test cluster.
- Anton Lindqvist (OpenBSD) for suggesting a new feature and multiple bug reports.
- Reyk Floeter, Paul Irofti (OpenBSD), Vsevolod Stakhov (FreeBSD), Colin Watson (Debian), Jean-Yves Migeon, Mike Williams, Nate Bargmann, and Thomas Guettler for suggesting new features.
- T. J. Townsend, Ted Unangst (OpenBSD), Christos Zoulas, Sevan Janiyan (NetBSD), Yuri Pankov (illumos), Leah Neukirchen (Void), Svyatoslav Mishyn (Crux), Jan Stary, and Markus Waldeck for multiple bug reports and useful suggestions.
- Kurt Jaeger (FreeBSD) for reporting multiple missing features.
- Antoine Jacoutot (OpenBSD) for suggesting multiple usuability improvements.
- Stuart Henderson (OpenBSD) for bug reports and for checking multiple ports patches.
- Aaron M. Ucko, Bdale Garbee (Debian), Daniel Sabogal (Alpine), Daniel Levai, and Rafael Neves for suggesting portability improvements.

- Brad Smith, Daniel Dickman, David Coppa, Dmitrij Czarkoff, Igor Sobrado, Ken Westerback, Martijn van Duren, Mike Belopuhov, Tim van der Molen (OpenBSD), Takeshi Nakayama (NetBSD), Alexander Kuleshov, Andy Bradford, Gabriel Guzman, George Brown, Gonzalo Tornaria, Jeremy Mates, Jerome Ibanes, Jesper Wallin, Lorenzo Beretta, Mark Patruck, Maxim Belooussov, Michal Mazurek, Michael Reed, Pavan Maddamsetti, Peter Bui, Raf Czlonka, Reiner Herrmann, Sean Levy, Serguey Parkhomovsky, Shane Kerr, Steffen Nurpmeso, Tony Sim, and Wolfgang Mueller for bug reports.
- Mark Kettenis, Otto Moerbeek, Tobias Stoeckmann, and Tom Cosgrove for checking patches in the base system.
- Jasper Lievisse Adriaanse, Klemens Nanni, Markus Friedl, Masahiko Yasuoka, Matthias Kilian, Pierre-Emmanuel Andre, Rafael Sadowski, Sebastian Reitenbach, Todd Fries (OpenBSD), Greg Steuck, and Jung Lee for checking patches in the ports tree.
- Alexander Hall, Andrew Fresh, Brent Cook, Doug Hogan, Kent Spillner, Nicholas Marriott, Peter Hessler, Stefan Sperling, Vadim Zhukov (OpenBSD), Abhinav Upadhyay, Joerg Sonnenberger (NetBSD), Dag-Erling Smørgrav (FreeBSD), Benny Lofgren, David Dahlberg, and Laura Morales for useful discussions.
- James Turner (OpenBSD) and Ulrich Spörlein (FreeBSD) for testing.