Proper citations on a static site: Zotero, Rust, and Typst

Emil Lindfors | February 10, 2026

After finishing my PhD, one habit stuck: Its hard to write a claim without wanting to cite a source.

Blog posts aren't papers. Nobody expects a references section at the bottom. But when I write about aquaculture production costs or sensor network architectures, hand-waving feels wrong. I often want to say "Lindfors et al. (2020) found that..." and have that link to a properly formatted reference that the reader can follow.

The problem is that static site generators don't do citations. Zola, Hugo, Jekyll – they process markdown, not BibTeX. Academic writing tools like LaTeX handle citations natively, but nobody wants to write a blog post in LaTeX. And citation plugins for blogging platforms are either nonexistent or fragile hacks bolted onto systems that were designed for listicles.

So I built a pipeline. It reads from my Zotero library, transforms `@citekey` references in markdown into linked citations, renders a references section in HTML, and generates properly typeset PDFs with Typst. Every piece is open source, file-based, and runs at build time.

The problem, concretely

I want to write markdown like this:

```
Research by @Christiansen2017 has identified diverse narratives
surrounding the greening of Norwegian salmon farming. The industry
faces what @osmundsenFishFarmersRegulators2017 describe as a
"wicked problem" where regulations and production goals create
complex trade-offs.
```

And have it render on the web as properly linked inline citations with a references section at the bottom, complete with journal names, volumes, DOIs, and links. I also want a downloadable PDF that looks like an academic paper, not a printed web page.

Here's what this requires:

1. A citation database (Zotero)
2. A way to resolve `@citekeys` to bibliographic data

3. A template system that renders references in HTML
4. A PDF generator that handles the same content

The pipeline

```
┌──────────────┐    ┌────────────────────┐    ┌─────────────────┐
│ Zotero       │───▶│ zotero-cite        │───▶│ Markdown        │
│ Library      │    │ (Rust CLI)         │    │ + TOML          │
└──────────────┘    └────────────────────┘    │ frontmatter     │
                                              └────────┬────────┘
                                                       │
                         ┌─────────────────────┼─────────────────────┐
                         ▼                     ▼                     ▼
                ┌──────────────┐      ┌──────────────┐      ┌──────────────┐
                │ Zola         │      │ Typst        │      │ Newsletter   │
                │ HTML         │      │ PDF          │      │ Email        │
                └──────────────┘      └──────────────┘      └──────────────┘
```

The markdown file is the single source of truth. Three different renderers consume it for three different outputs.

Step 1: Zotero and Better BibTeX

I keep all my references in Zotero, the open-source reference manager. The key plugin is Better BibTeX (BBT), which assigns stable, human-readable citation keys to every item. Instead of Zotero's auto-generated IDs, I get keys like `osmundsenFishFarmersRegulators2017` or `iversenProductionCostCompetitiveness2020`.

BBT stores these keys in Zotero's SQLite database, which is the interface my tooling reads from. No manual exports, no keeping `.bib` files in sync. The database is the source of truth.

Step 2: zotero-cite

zotero-cite is a Rust CLI I wrote to bridge Zotero and static site generators. It does one thing: scan a markdown file for `@citekey` patterns, look them up in Zotero's database, and transform them into formatted citations.

Install it:

```
cargo install --git https://github.com/emillindfors/zotero-cite
```

It auto-detects your Zotero data directory on Linux, macOS, and Windows (including WSL). Or you can point it at a specific database with `ZOTERO_DATA_DIR`.

What it does

Given a markdown file with `@citekey` references, `zotero-cite process` does two things:

1. Transforms inline citations from `@citekey` to linked text:

- `@Smith2023` becomes a narrative citation: "Smith (2023)" with a link to `#ref-Smith2023`
- `[@Smith2023]` becomes a parenthetical citation: "(Smith, 2023)" with a link

2. Generates a references array in the file's TOML frontmatter:

```toml
[extra]
references = [
  { key = "Smith2023", type = "article", author = "Smith, J. and Johnson, M.",
    title = "IoT Sensor Networks in Modern Aquaculture",
    journal = "Journal of Marine Technology", volume = "45",
    number = "3", pages = "112-128", year = "2023",
    doi = "10.1234/jmt.2023.001" },
]
```

This is the important part. The bibliographic data goes into frontmatter as structured data, not as rendered HTML or hardcoded text. Zola can then pass it to templates, which can render it however they want – HTML for the web, plain text for email, lists for PDFs.

Usage

```bash
# Process a single file
zotero-cite process content/blog/my-post/index.md --output content/blog/my-post/index.md

# Look up a specific citekey
zotero-cite lookup @osmundsenFishFarmersRegulators2017

# List all available citekeys
zotero-cite list
```

I run it as part of my build script. It scans every markdown file for `@` patterns and processes the ones that have citations:

```bash
for file in $(find content -name "*.md" -type f); do
  if grep -q '@[a-zA-Z]' "$file" 2>/dev/null; then
    zotero-cite process "$file" --output "$file"
  fi
done
```

After this step, the markdown files are self-contained. They have both the inline citation text and the full bibliographic data in frontmatter. No external dependencies at build time.

Step 3: Rendering in Zola

Once the references are in frontmatter, Zola's template system takes over.

The citation shortcode

For inline references, I use a minimal shortcode:

```
{# templates/shortcodes/reference.html #}
<a href="#ref-{{ key }}" class="citation">
  [{{ num | default(value=key) }}]
</a>
```

In a post: `{{ reference(key="smith2024", num=1) }}` renders as a clickable `[1]` that jumps to the reference at the bottom.

The references section

The page template checks for `page.extra.references` and renders them automatically:

```
{% if page.extra.references %}
<section class="references">
    <h2>References</h2>
    <ol class="reference-list">
        {% for ref in page.extra.references %}
        <li id="ref-{{ ref.key }}">
            {{ macros::format_reference(ref=ref) }}
        </li>
        {% endfor %}
    </ol>
</section>
{% endif %}
```

Format-aware macros

The `format_reference` macro dispatches on reference type:

```
{% macro format_reference(ref) %}
{% if ref.type == "article" %}
    {{ self::format_article(ref=ref) }}
{% elif ref.type == "book" %}
    {{ self::format_book(ref=ref) }}
{% elif ref.type == "inproceedings" %}
    {{ self::format_inproceedings(ref=ref) }}
{% elif ref.type == "phdthesis" %}
    {{ self::format_phdthesis(ref=ref) }}
{% else %}
```

```
    {{ self::format_generic(ref=ref) }}
{% endif %}
{% endmacro %}
```

Each type macro knows which fields to expect. An article gets journal, volume, issue, pages, and DOI. A book gets publisher and ISBN. A conference paper gets proceedings title. The macros handle missing fields gracefully – not every article has a DOI, not every book has a URL.

Here's the article macro:

```
{% macro format_article(ref) %}
<span class="ref-authors">{{ ref.author }}</span>.
<span class="ref-title">"{{ ref.title }}"</span>.
{% if ref.journal %}<em class="ref-journal">{{ ref.journal }}*{% endif %}
{% if ref.volume %}, vol. {{ ref.volume }}{% endif %}
{% if ref.number %}, no. {{ ref.number }}{% endif %}
{% if ref.pages %}, pp. {{ ref.pages }}{% endif %}
{% if ref.year %}, {{ ref.year }}{% endif %}.
{% if ref.doi %}[doi:{{ ref.doi }}](https://doi.org/{{ ref.doi }}){% endif %}
{% endmacro %}
```

This renders to something like:

> Smith, J. and Johnson, M. “IoT Sensor Networks in Modern Aquaculture”. *Journal of Marine Technology*, vol. 45, no. 3, pp. 112-128, 2023. doi:10.1234/jmt.2023.001

One-click citation export

Every blog post has a “Cite” button in the sidebar that opens a modal with pre-formatted BibTeX and APA citations:

```
@article{lindfors2026_my_post,
  author = {Emil Lindfors},
  title = {My Post Title},
  year = {2026},
  url = {https://lindfors.no/pdf/my-post.pdf},
  note = {Accessed: February 10, 2026}
}
```

This means readers can cite *my* posts the same way I cite others. The whole chain is bidirectional.

From frontmatter to three outputs

Because the bibliographic data lives in TOML frontmatter as structured data, it feeds multiple renderers from a single source:

- **HTML** – Tera macros render the references section with DOI links and semantic markup
- **PDF** – the same frontmatter drives Typst-generated PDFs with properly typeset references
- **Newsletter** – the send pipeline gets a simplified version

The citation processing step runs first in the build, before PDFs are generated or the site is built. After `zotero-cite process` runs, the markdown files are self-contained – they have both the inline citation text and the full bibliographic data. No external dependencies at render time.

Why not just use footnotes?

Footnotes would be simpler. Markdown supports them natively. But they're not the same thing as citations:

1. **Footnotes don't have structured data.** A footnote is just text. A citation has authors, year, journal, DOI – fields I can format differently in different contexts.
2. **Footnotes aren't reusable.** If I cite the same paper in two posts, I retype the reference. With Zotero as the source, I write `@iversenProductionCostCompetitiveness2020` and the full reference appears.
3. **Footnotes can't be exported.** My references section renders as structured HTML with DOI links, semantic classes, and anchor IDs. Footnotes are just paragraphs.
4. **Footnotes don't compose.** The same frontmatter data feeds HTML rendering, PDF generation, and citation export. Footnotes only work in one direction.

Why not Pandoc?

Pandoc has excellent citation support through its `citeproc` filter. It reads `.bib` files, supports CSL styles, and can output to virtually any format. For many people, Pandoc is the right answer.

I didn't use it because:

- **I use Zola, not Pandoc, to build my site.** Introducing Pandoc as a preprocessing step means maintaining two markdown dialects – Zola's (with shortcodes, TOML frontmatter, `+++` delimiters) and Pandoc's (with YAML frontmatter, `---` delimiters). The impedance mismatch creates friction.
- **I wanted the data in frontmatter.** Pandoc renders citations to final output. I want the structured bibliographic data available to my templates so I can render it differently for web, PDF, and email.

- **I wanted to read from Zotero directly.** Pandoc reads `.bib` files. I'd need to export from Zotero to `.bib` and keep it in sync. `zotero-cite` reads from Zotero's database, so my library is always current.

The citation toolchain

Role	Tool
Reference manager	Zotero + Better BibTeX
Citation processing	zotero-cite (Rust CLI)
Template rendering	Tera macros in Zola
Math (web)	KaTeX

What I'd change

If I were starting over:

- **CSL support in zotero-cite.** Right now the formatting is hardcoded in Tera macros. Supporting CSL styles would let me switch between APA, IEEE, Vancouver, etc. without changing templates.
- **Automatic numbering.** Currently I manually assign citation numbers with `{{ reference(key="...", num=1) }}`. The shortcode should auto-number based on order of appearance.
- **Bidirectional links.** Clicking a citation scrolls to the reference, but clicking a reference doesn't scroll back to where it was cited. This is standard in PDFs but tricky in HTML.

These are improvements, not blockers. The system works well enough that I use it for every post that needs references.

The point

Blog posts with citations are better blog posts. When I write "production costs vary significantly across salmon-producing countries", readers shouldn't have to trust me – they should be able to click through to Iversen et al. (2020) and check for themselves.

Academic writing tools make this easy but are terrible for web publishing. Blogging tools are great for the web but ignore citations entirely. Bridging the gap took a small Rust CLI and some Tera macros. The pieces were all there – they just needed connecting.

The code for `zotero-cite` is on GitHub. If you use Zotero and a static site generator, it might save you some time.

This post is part of a series on the infrastructure behind this blog. See also: Site overview, Self-hosted newsletter, Typst PDF generation, Images.