


Nov 8, 2026

# Switching to Typst, and blogging with it through Tola

typesetting LaTeX typst tola meta

## My history

Over the years I've had to write many documents. Most of them while I was doing my bachelor's, as reports for different assignments, and from that I've had my fair share of late afternoons and nights fixing up broken styles in shared documents, broken figure numbers in MS Word, or even tables of contents that show wrong pages. It was hopeless almost from the start. I tried both Google Docs, which was simple and worked great for simultaneous editing, and Word, which people praised and hated, but was, and mostly is, the *status quo* for most non-academic users.

To be honest, I hated using both, it seemed that I had no control over how the documents looked, and anyone pasted anything without caring and now the styles were all over the place. For me, it seemed there was no way out. I was stuck using the same horrible tool as everybody else.

That's when I thought: "I really can't believe there's nothing better than this. Does everyone just use this? Surely there's something better."

So around 2022-2023 I started searching. And I didn't have to search for long, because I quite quickly found LaTeX (LaTeX), a *typesetting system*, which was the standard in academics, for writing papers and documents in general.

## LaTeX

This tool produces beautiful documents, separating the appearance or styling from the actual content. It generates documents from "script-like" documents, and allows for almost *infinite* control over styling and layout, without blocking the actual writing. It manages bibliographies, figure numbers, and tables of contents with ease, so it's easy to set the style once and forget about it. LaTeX was everything I searched for and wanted.

I was in love.

Still, I think I was so frustrated with previous tools that the love for LaTeX might have clouded my judgment. Like in any other relationship, after that initial period, I started seeing some of the flaws. The TeX Live distribution, which was how I, and most of those who use LaTeX locally, installed it, was really heavy, usually 7–8 GB, because it downloads *all* packages.

Also, the language itself was a bit clunky, especially for a Spanish keyboard, as `\` needs `Alt Gr` + `º`, a clunky combination not present on a US keyboard. For context, let's see an example of some complex LaTeX:

```latex
\documentclass{article}
\usepackage{xcolor}

% A command with 2 arguments, the first one optional
% (and yellow by default)
\newcommand{\highlight}[2][yellow]{\colorbox{#1}{#2}}

% Internal names use @, which is only a "letter"
% between these two lines
\makeatletter
\newcommand{\shout}[1]{\@shout{#1}}
\def\@shout#1{\textbf{\MakeUppercase{#1}}}
\makeatother

% A command that defines commands:
%   the inner arguments need a double #
\newcommand{\defgreeting}[2]{%
  \newcommand{#1}[1]{#2, ##1!}%
}
\defgreeting{\hello}{Hello}

\begin{document}
\section{Introduction}
I write in \LaTeX{} every day. \highlight{This} is yellow,
\highlight[cyan]{this} is cyan, and \shout{this} is loud.
\hello{world}
\end{document}
```

It looks *"reasonable"* at first glance, but creating your own commands is where things start getting specially weird:

- Arguments don't have names, only positions, from `#1` to `#9`. If you need a tenth one, tough luck. And when you read `\colorbox{#1}{#2}` you have to go back up to remember what `#1` was.
- Only *one* argument can be optional, and it has to be the first one, between square brackets. Anything fancier means pulling in another package, like `xparse`.

- There are at least four ways to define a command: `\newcommand`, `\renewcommand`, `\providecommand` and the plain TeX `\def`, each failing (or silently not failing) differently when the command already exists.
- `@` is a letter only sometimes. Packages use it to name their internals, and to touch them from your document you have to wrap everything in `\makeatletter` and `\makeatother`.
- If you define a command inside another command, the inner arguments become `##1`, and one more level deep, `####1`.
- Spaces after a command name are eaten, so `\LaTeX is great` comes out as "LaTeX is great". You need `\LaTeX{}` or `\LaTeX\ ` instead. And line breaks inside definitions count as spaces, which is why you see those `%` at the end of lines everywhere.

And all of that is before getting into `\expandafter`, catcodes, or trying to make sense of an error message that points to a line that has nothing to do with the actual problem.

This started frustrating me, as it seemed the language was powerful, that I could make it do anything, but it was so hard that it didn't make any sense. I spent many hours and days debugging strange error messages and a tool that was not cooperating. Still, it was miles ahead of what I could achieve with Word or similar tools, and as it was standard in academia, I felt confident using it, even though a bit frustrated. I even ported my CV to it.

But I was not satisfied, so I was still searching, even if not as profoundly, for something better. And being in some Rust communities, I eventually heard about a tool that was trying to provide a modern alternative to LaTeX: Typst (**typst**). It promised to fix exactly what I didn't like while still providing results similar or equal to LaTeX. I was intrigued.

## typst

The more I read, the more in love I was. This new language promised beautiful documents, but typeset with a syntax similar to Markdown, while still having a full scripting language behind it. This sold me, so I tried it.

First, I went to install it, and in contrast to LaTeX, to my surprise, it was a single 14 MB executable. So small, I thought. Especially compared to the gigabytes that LaTeX brought with it. But it wasn't because it didn't have packages. Quite the opposite. Let's be clear, **typst** doesn't have the thousands of packages TeX Live pulls in, but it has enough for most tasks, and where a package is missing, the scripting is so easy you don't really miss it.

**typst** even allows for reading `json`, `csv` or any file for building tables or plots dynamically. Everything seems, at least for now, more thought out, and planned. It really seems like it has learned from mistakes of the past. For comparison, here's the **typst** equivalent of the LaTeX example above:

```
// A function with a named argument, yellow by default
#let highlight(body, color: yellow) = box(
  fill: color, inset: (x: 2pt), outset: (y: 2pt), body,
)

// Helpers are just variables, no special characters needed
#let shout(body) = strong(upper(body))

// A function that returns a function
#let greeting(word) = name => [#word, #name!]
#let hello = greeting("Hello")
```

```
12
13 #set heading(numbering: "1.")
14
15 = Introduction
16 I write in Typst every day. #highlight[This] is yellow,
17 #highlight(color: aqua)[this] is cyan, and #shout[this] is loud.
18 #hello[world]
```

The same document, but each of the pain points from before simply isn't there:

- Arguments have names. `color: aqua` says what it is, and the body is just `body`, so there's no going back up to figure out what `#1` meant.
- Any argument can be optional and have a default, as many as you want, in any order. No extra packages needed.
- There is one way to define things: `#let`. Shadowing an existing name is allowed and does exactly what you'd expect.
- There's no `@` trick for internals. Anything you don't want to expose lives in its own file, and you `#import` only what you need.
- Functions are values, so a function that builds functions is just a closure. Nesting doesn't change the syntax, no `##1` in sight.
- Markup and code are clearly separated: `#` jumps into code, and `[...]` jumps back into markup. So "Typst is great" comes out as written, and line breaks in code don't leak into the document.

And when something does go wrong, the error points to the exact span of the problem, usually with a hint on how to fix it. The compiler is also incremental, so the preview updates as you type, instead of after a full rerun (or three, if you have references). Compilation is also orders of magnitude faster than for LaTeX.

I've now been using **typst** for almost 2 years, and I still haven't found something that is uncomfortable to do. I'm still as sold on it as the first day. It provides exactly what I needed without any of the negatives.

Just to finish on this part. As with `beamer` in LaTeX, there are packages to build similar slide decks, and now some universities and conferences are accepting **typst** papers. Mainly in Germany, which, being the home country of the project, is where it has had a bigger spread.

## So typst is great for documents, but, blogging?

If this tool is for creating documents, how come you can build a blog with it?

**typst** has provided for a few versions now the ability to generate HTML versions of the documents, and I have been thinking of reviving my blog using it for some time. The final straw was a post I read on Reddit about someone who had done exactly that, using a WebAssembly version of **typst** run through a bundler. I thought: "If they can, I should be able to as well", so I did. I built a version using the Typst 0.14 CLI together with Astro and some special modules. And it built, the website was great, but I wasn't satisfied with the solution. It was clunky and strange, as it mixed many different unrelated things to work.

A few weeks after I built it, **typst** 0.15 came out, with support for multi-document generation, which meant the possibility of generating multiple HTML documents from a single project. It also included

MathML generation, which was nice, as I was doing it manually before. I looked at it, as it seemed interesting for what I built, but seemed difficult to integrate, so I left it as is.

Finally, a couple of weeks ago, on the Typst Subreddit, I read about a template for blog generation for the *Tola* static site generator, which seemed a lot more promising and less bespoke than whatever I could build myself. So armed with Claude, I decided to move my dead and half ported blog to *Tola*, and here we are.

This is the first post of the new blog built with this stack. Writing this post has been quite nice. This blog also now allows, thanks to **typst**, the download of posts as PDFs.

If I find any inconveniences to this setup or Typst in general, I'll make sure to post here about it, but I think this is the start of some nice place where I will share whatever I'm building, investigating or just thinking every so often.

—
Have a nice day :)