A Brief Introduction to Typst

Apr 6, 2026

Typst is an open-source markup format. It has a free web app that charges for some Pro features such as git integration and Zotero connection, but its compiler, CLI and VS Code extension are all free.

This is a brief introduction to Typst and its ecosystem (the "Typst Universe"), from the perspective of someone already familiar with Markdown and LaTeX.

1. Why another markup format?

Typst is a markup language *integrated with scripting*. This allows the user to easily write scripts that are tailored to their specific needs, so that they depend less on heavy, external packages.

This gives rise to another advantage of Typst: lightweight. Built with Rust, its binaries are typically less than `20MB` in size, and it compiles to `PDF` blazingly fast (try the web app https://typst.app and see for yourself). A full-fledged technical paper is compiled and previewed in real time like a plain markdown file.

2. Markup in Typst

The markup syntax of Typst inherits neither LaTeX nor Markdown. The official tutorial about basic markup explains it in detail. Instead of `#`, Typst uses `=` to indicate sections, because the latter is reserved for "functions":

```
#<some-function>(<arguments>)
```

For example, figures are placed in the document by the `figure` function. Functions, rather than the markup syntax, is what makes Typst special.

2.1. Math

The math syntax of Typst, however, deserves an explicit mention. It removes much of the clutter in the usual LaTeX math syntax *out of the box*, including the backward dash `\` before symbols, the `\left`, `\right` declarations before brackets in order for them to stretch. The fraction syntax is especially elegant: for simple fractions where both the nominator and the denominator have only one term, a simple forward slash `/` suffices to make a fraction. But if you actually want the forward slash for, e.g. in-line math, you can either escape this character: `a\/b` $\mapsto a/b$ or, more elegantly, use the built-in symbol `slash`: `a slash b` $\mapsto a/b$. For example, the following Typst code

```
$ sum_(n=1)^(infinity) alpha^n / n! = "e"^(alpha) $
```

produces the formula

$$\sum_{n=0}^{\infty} \frac{\alpha^n}{n!} = \mathrm{e}^{\alpha}. \tag{2.1}$$

When formulas get long, these little differences accumulate and dramatically improve the user's editing experience.

Also, rather than separately defined "in-line" and "display" math environments, not to mention "equation", "align" and friends, Typst has a unified environment for math, which is the single dollar sign. Put spaces around your formula to make it "display". Add line breaks (`\` instead of `\\`) and alignment symbols (still the `&` symbol) for multi-line math. All with single dollar signs.

2.2. Reference

The label-reference syntax in Typst is `<label>` and `@<label>[<supplement>]`, where the `@ ...` syntax is shorthand for the `ref` function. Typst automatically finds the `kind` of the referenced element and attaches the appropriate text prefix, i.e. it produces "Equation (3.1)" instead of the bare number "3.1" when you reference equation 3.1. As a special case, passing an empty `supplement` gives back the bare number.

3. Scripting in Typst

Examples of Typst scripts are collected here. I don't yet have the ability to do a top-down explanation on this, but Typst's scripting language is quite intuitive. Anyone with a bit of Python fluency should be able to learn it quickly: `values` have `fields` and `methods`, just like Python objects. There are three different "modes" in Typst: text, math, and code. See the documentation for detail.

The two most important scripting keywords of Typst are `set` and `show`. The `set` keyword sets the default values for some of the parameters of a function for all future uses of that function, while the `show` keyword redefines a function that is already defined. In Typst, all styling elements (headings, code blocks, formulas, etc) are functions, so whenever you tweak the styling, you are effectively redefining a function or setting its arguments.

3.1. Formatting and dates

The title of this document is typeset via

```
#set document(title: [A Brief Introduction to Typst])
#show title: it => {align(center,it)}
// We see explicitly the re-defining of the function `title` here.
// The symbol => is an automatic concatenation of = and >.
#title()

#align(center)[
#datetime.today().display("[month repr:short] [day padding:none], [year]")
]
// Here, the second argument of `align`
// is given in a Typst idiom: contents in square brackets
// are automatically recognized as the `body` argument.
#block(height: 0.5em)
```

3.2. Custom lists

To customize a numbered list environment, in LaTeX one needs the `enumitem` package. But in Typst it's very simple: to make a list numbered with "*Step n*", we write

```
#enum(numbering: n => emph[Step #n.],
[#lorem(50)],
// Fifty placeholder words; `#lorem` is the dummy text function
[another point]
)
```

which produces:

Step 1. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Step 2. another point

Further, we can wrap this in a function,

```
#let steps(..args) = {
// The `..args` is a "sink" for an array of
// arguments you don't know how many there are.
  enum(
    numbering: n => emph[Step #n.],
    ..args
  )
}
// and call it like this:
// #steps([#lorem(50)],[another point])
```

so that it can be re-used throughout the document.

3.3. Equation numbering

Equation numbering in this document is automatic. Consider the following equations.

$$\text{First equation: } x = 1$$

$$\text{Second equation: } x^2 = 1 \tag{3.1}$$

The first equation is not numbered while the second is numbered and inherits the section number "3". This inheritance is implemented as follows:

```
#show heading.where(level: 1): it => {
  counter(math.equation).update(0)
  it
}
#set math.equation(numbering: (..nums) => {
  let section = counter(heading).get().first()
  numbering("(1.1)", section, ..nums)
})
```

and the first equation is explicitly declared to have no numbering:

```
#math.equation(block: true, numbering: none)[$ "First equation:" x=1 $]
```

3.4. Boxes and theorems

I like to put important parts of text into boxes, like the following one:

Result 3.1. Something *really* important...

This is easily realized with the ctheorems package and the following code.

```
#import "@preview/ctheorems:1.1.3": *
#show: thmrules
#let result = thmbox(
                    "theorem", // counter name
                    "Result", // head display name
                    base_level:1, // inherit level from heading
                    separator:[*.* ], // separator between head and body
                    fill: rgb("#eeecec")
                   )
```

3.5. Three-line tables

The official table guide of `typst` already provides many examples regarding tables. An interesting example missing from there is the so-called 3-line table, where only the top and bottom are stroked, and the first row has a thin bottom line to separate it from the table entries.

This can be very easily realized in `typst`:

```
// Only 1pt top and bottom, plus a 0.5pt separator between headers and entries
// The key: top can override bottom, except for
// 1st-row top (nobody's bottom) and last-row bottom (nobody's top)
#set table(stroke: (_, y) => (
  top: if y > 1 { 0pt } else if y == 0 { 1pt } else if y == 1 { 0.5pt },
  bottom: 1pt,
))
```