A Brief Introduction to Typst

Sep 22, 2025

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 denumerator 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

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

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.

3.1. Formatting and dates

The title of this document is typeset via

```
#align(center, text(19pt)[
  *A Brief Introduction to Typst*
])

#align(center)[
#datetime.today().display("[month repr:short] [day], [year]")
]
#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.],
[first point],
[another point]
)
```

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. In LaTeX one would have to explicitly specify the first one as a display math environment and the second an equation environment. However, in Typst this can be realized automatically by a nice formatting script. It basically suppresses an equation's numbering if it has no label, which is implemented by an `if` loop. To be specific, the code is

```
#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)
})
```

```
#show math.equation: it => {
  if it.block and not it.has("label") [
    #counter(math.equation).update(v => v - 1)
    #math.equation(it.body, block: true, numbering: none)  #label("")
  ] else {
    it
  }
}
```

The `#label("")` is added to avoid endless recursion: in its absence, an unlabelled equation will be `show`-rule mapped to an equation which *still doesn't have any label*, which should then be processed by this `show` rule again, etc. This work-around looks a bit ugly, and requires the user not to write a stand-alone `@` in the document, because there are too many equations labelled with the empty string.

Of course, one can choose to follow the default of Typst, which is automatically numbering all equations so long as some of them might be labelled. That way, the numbering of an equation will not change upon labelling a new equation, while in our implementation of numbering, only the labelled equations are counted.

3.4. Boxes and theorems

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

Result 3.1. Something important here.

In LaTeX, this is usually achieved via the `tcolorbox` package. But Typst allows formatting through functions, and it so happenes that there is a `block` function with some parameters available, so one can build this box via

```
#let (result-counter, result-box, result, show-result) = make-frame(
  "result",
  "Result",  // The title that will appear
  counter: theorem-counter,  // or inherit from an existing counter if needed
  inherited-levels: 1,
  inherited-from: heading,
  render: (prefix: none, title: "", full-title: auto, body) => [
    #block(
      fill: rgb("#f5f5f5"),  // Light grey background
      stroke: rgb("#cccccc"), // Grey border
      radius: 5pt,           // Rounded corners
      inset: 1.2em,           // Padding inside the box
      width: 100%,           // Full width
      [#strong[#full-title.]#sym.space#body]
    )
  ],
)
#show: show-result
```

This code is the standard customization code of the Theorion package. I'm now using the `simple` theme, there are fancier themes available. Note that because I set the `counter` of my "result" box to be `theorem-counter`, the numbering of the following theorem is 3.2.

Theorem 3.2. *Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.*