My workshop about Typst for the fhLUG in Campus Hagenberg

Tim Peko (TimerErTim)

On June 8th, 2026, I had the wonderful opportunity to lead a Typst workshop for the fh Linux User Group at Campus Hagenberg, made possible by Daniel Knittl-Frank's excellent organization. It was a fantastic experience, with enthusiastic and engaged participants. In this blog post, you'll find highlights from the workshop as well as access to the full recording (German).

Table of Contents

If you're primarily interested in the workshop recording (German language only), you'll find the video at **https://www.youtube.com/watch?v=GS978MRbqqo**

1 Quick Background

fhLUG is the **Linux User Group FH Hagenberg**. They host talks, workshops or simple get togethers roughly once a month to discuss Linux and topics from the open software world.

Daniel Knittl-Frank is the main organizer and head of the group. His support and hassle-free organization allowed me to focus on the content of the workshop. Thank you, Daniel!

2 The Workshop

Figure 1: Me after the workshop and snagging a fitting shirt.

Before jumping into action, we started with a high level overview of document writing and why **Typst** stands out. Did you know that you can categorize tools in two categories?

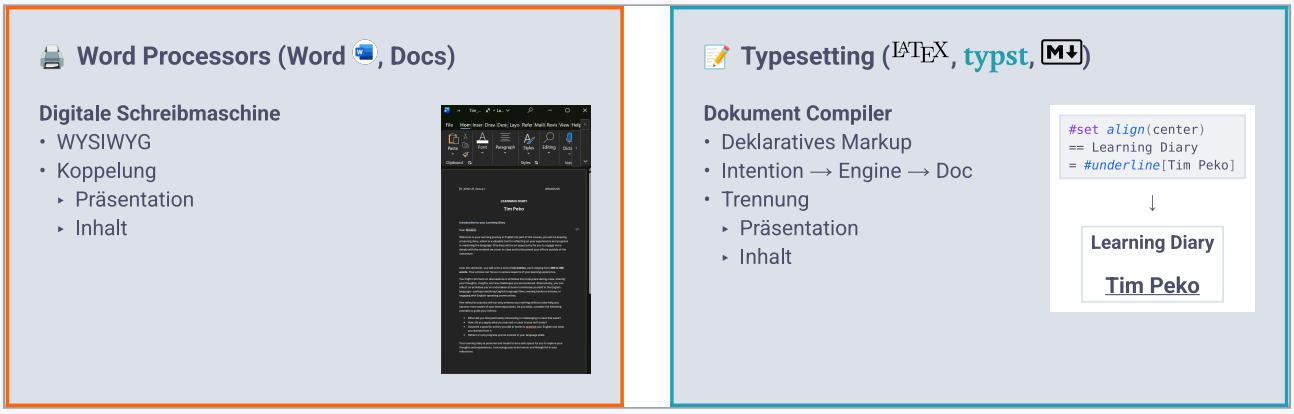

Figure 2: Two types of categories: Word Processors and Typesetting Systems.

The greatest advantage of typesetting is the separation between content and presentation. This allows for a lot of flexibility and reusability. It is literally what powers this blog website in its dark and light theme, as well as making blog posts available for download in a PDF optimized format.

Typst enables us to write documents programmatically. For example, let's generate the first 4 cubic numbers:

```
#for i in range(1, 4 + 1)
[
  $#i^3 = #calc.pow(i, 3)$\
]
```
Typst

→

$1^3 = 1$
$2^3 = 8$
$3^3 = 27$
$4^3 = 64$

Not a single cubic number written by hand

See that?! We have the power of a full turing complete programming language at our fingertips.

2.1 About Typst

The **Typst GitHub Repository** has about **54.1k** stars and **441** Contributors, which is no small achievement at all. Development was initiated by **Martin Haug** and **Laurenz Mädje** in 2019. After 4 years of development they founded an organization in **Berlin**.

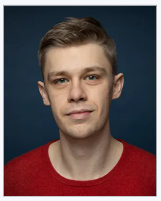

Figure 3: Martin Haug

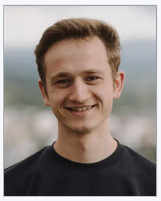

Figure 4: Laurenz Mädje

> There are two ways to make something flexible: Have a knob for everything or have a few knobs that you can combine in many ways. Typst is designed with the second way in mind.
>
> –Typst `README.md`

The above quote perfectly captures the philosophy of Typst and lays the foundation for its expressive power.

2.2 Deep Dive

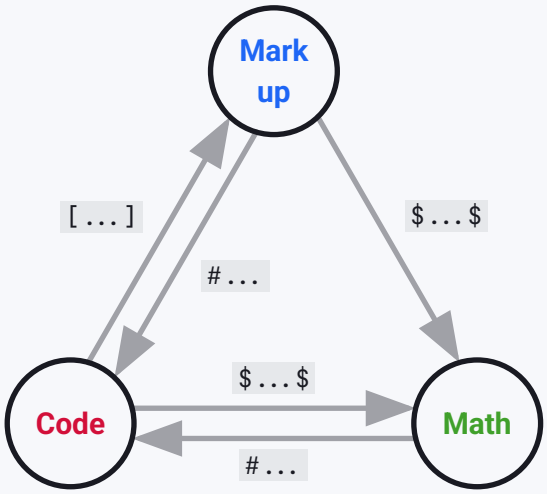

Figure 5: The three modes of Typst: *Markup*, *Math*, and *Code*. Arrows indicate syntax to switch between modes.

```
== A Title
*Bold* and _italic_ content.
#rect(stroke: red, "Trapped")
$ E(X) = sum_(i=1)^n x_i P(X = x_i) $
```
Typst

↓

A Title

Bold and *italic* content.

Trapped

$$E(X) = \sum_{i=1}^{n} x_i P(X = x_i)$$

Listing 1: The markdown related *Markup* mode, switching to *Code* mode for `rect` drawing and *Math* mode for the equation.

The official **Typst Documentation** is a great resource to learn more about the syntax and features of Typst. You can directly jump into **the tutorial** for a step-by-step introduction to Typst's features.

Arguably the most powerful styling tools are the `#set`- and `#show`-Rules. You can already tell that they are invoked in *Code* mode.

2.2.1 `#set`-Rules

`#set`-Rules are used to set default parameters for the following functions.

> Have you mentioned functions?
>
> Turns out, everything you see in Typst is a function. `= Topic` is really just syntactic sugar for an invocation of the `#heading("Topic")` function. `*Bold*` is just `#strong("Bold")`. This holds true for all of Typst and is the second foundation for its flexibility.
>
> 
>

Let's say we wanted to change the text font for the rest of this document. I will just quickly insert `#set text(font: "Source Code Pro")`...

`Ah, there it is. Perfect! Neat huh?`

`Revert back with #set text(font: "Roboto")`

> Mind the scope!
>
> `#set` (and `#show`) -Rules are effective for the whole scope they are defined in. If you have a block

```typst
#[
  #set text(fill: red)
  Red Text
]

Text normal
```

↓

Red Text

Text normal

you can observe the red text `#set`-Rule only effects the *Markup* block. Had we left out the block:

```typst
#set text(fill: red)
Red Text

Text normal
```

↓

Red Text

Text normal

you can observe the red text `#set`-Rule now effects the whole content.

2.2.2 `#show`-Rules

Whereas we saw `#set`-Rules to set default parameters for the following functions, `#show`-Rules are used to transform elements. For the rest of this document, I want “Typst” to show up in blue and in italic.

```typst
#show "Typst": set text(fill: blue)
#show "Typst": it => [_#(it)_]
```

Let’s also try out putting the headings inside a green rectangle:

```typst
#show heading: it => rect(stroke: green, it)
```

If you continue on reading this blog post, you should notice all our intended `#show`-Rules are in place and working as expected.

2.2.3 Packages

This serves only as demonstration purpose about the power of the already promising package ecosystem: We will use the “conch” package for emulating a whole terminal right in our document:

```
timerertim@conch

timerertim@conch:~$ ls
test.txt
timerertim@conch:~$ cat test.txt
Show Line!
Ignore Line!
timerertim@conch:~$ cat test.txt | grep Show
Show Line!
timerertim@conch:~$
```

Very nice, looking like a real terminal! Can you believe that all of that is just 19 lines of code?

```typst
#import "@preview/conch:0.1.0": terminal-block, system

#terminal-block(
  width: 100%,
  user: "timerertim",
  system: system(
    files: (
      "test.txt": (
        "Show Line!",
        "Ignore Line!"
      ).join("\n")
    )
  ),
  ```
  ls
  cat test.txt
  cat test.txt | grep Show
  ```
)
```

3 Conclusion

Thanks for reading this far! I hope you enjoyed the very brief overview of *Typst* and the incredibly steep deep dive. You can find a few resources below in order to dive deeper into *Typst*:

- ***Typst* Documentation**
- ***Typst* Tutorial**
- ***Typst* Forum**
- ***Typst* Universe** (Package Registry)
- ***Typst* GitHub Repository**