



# Memorias de p\*\*\*\* madre

Introducción a Typst

Luis Daniel Casais

@rajayonin

GUL UC3M

Noviembre 2025

# Transparencias



[github.com/rajayonin/typst-intro](https://github.com/rajayonin/typst-intro)

# **Introducción**


Typst es como  $\text{\LaTeX}$ , pero no da asco.

— Yo

## Como L<sup>A</sup>T<sub>E</sub>X...

- Genera documentos (PDFs) a partir de archivos de texto plano
- Defines reglas → programable
- Extremadamente útil para  $e - c \cdot u^a = c_i \sqrt{o} + \frac{n}{e^s}$
- Numeración y referencias automáticas
- Paquetes (librerías) para hacer mil cosas

## ...pero no da asco

- *Blazingly fast* 
- Sintaxis simple e intuitiva
- Lenguaje de *scripting* moderno
- Herramientas modernas

# Usar Typst

- **Online:** [typst.app](https://typst.app)
- **Terminal:** Typst CLI
- **IDEs:** Tinymist LSP
  - VS Code (y sus 800 *forks*): Tinymist Typst
  - IntelliJ: Typst Support
  - Neovim: [typst-preview.nvim](#)
  - GNU Emacs: [typst-preview.el](#)
  - Zed: [Typst](#)

¡A instalar!

*How to Typst*

# Modos sintácticos

**Código** ( `#` ): Permite usar directamente el lenguaje de *scripting*.

```
#let foo(n) = { n + 1 }
```

Tengo `#foo(68)` Labubus.

**Markup** ( `[..]` ): Modo por defecto, *WYSIWYG*.

Typst es *\_blazingly fast\_!* 

```
let cool-os = [*Linux!* 
```



# Modos sintácticos

**Mates** ( `$..$` ): Permite expresar ecuaciones matemáticas.

```
$ x = (-b plus.minus sqrt(b^2 - 4a c)) / (2a) $
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Los distintos modos son intercalables:

```
El peso de tus madres se calcula con $lim_(x -> 0) 1/x$  
#let peso(_madre) = { sym.infinity }  
#for i in range(2) [  
  Tu madre #i pesa #peso(i) kg.  
]
```

# Formato de texto

- Comentarios con `//` o `/* */`
- Caracteres especiales ( `\\` , `#` , `$` , `_` , `*` ) se escapan con `\\`
- **Negrita:** `*...*` (`strong`)
- *Cursiva:* `_..._` (`emph`)
- Monoespaciado : ``...`` o `````` (bloques de código) (`raw-`  
Subrayado: `#underline[...]`
- `= Capítulo` , `== Sección` , `=== Subsección` (`heading`)
- Notas a pie de página: `#footnote[...]`
- Links: `#link("<url>")[...]` , ) o `https://link.com`
- Comillas «inteligentes» (dependientes del idioma): `"` , `'`
- Símbolos: `~` (`nsbp`), `---` / `--` (raya/semiraya), `sym emoji`

Todo esto son «funciones elementales».

`_caca_`  $\equiv$  `#emph[caca]`  $\equiv$  `#emph([caca])`

# Saltos

Los saltos de línea son automáticos. Un salto de línea en el archivo fuente no rompe una línea.

- Para saltar una línea, se usa `/` (o `#linebreak()` )
- Para saltar un párrafo se deja una línea en blanco (o `#parbreak()` )
- Para saltar una página se usa `#pagebreak()`

Esta línea no  
la rompe ni Dios.

Esta línea no la rompe ni Dios.

Esta la rompo yo\  
porque me da la gana.

Esta la rompo yo  
porque me da la gana.

# Listas

Enumeradas (`enum`) o no enumeradas (`list`).

- + Primero
- + Segundo

1. Primero
2. Segundo

- Uno
- Otro

- Uno
- Otro

Y se pueden anidar:

- + Primero
  - Primero uno
  - Luego otro
- + Segundo

1. Primero
  - Primero uno
  - Luego otro
2. Segundo

# Imágenes

Soporta PNG, JPG, GIF, SVG<sup>1</sup>, PDF<sup>2</sup>, y *raw bytes*.

```
#figure(  
  image("img0.png", width: 80%),  
  caption: [...]  
) <fig:img0> // label
```

- Colocación: parámetro `placement`
  - `none` : exactamente aquí (*default*)
  - `auto` : arriba o abajo de la página
- Para hacerlas *inline*, usa `#box` en lugar de `#figure` .
- Puedes escalarlas con `scale` , e.g. `scale(50%, image(...))`

---

<sup>1</sup>Excepto `foreignObject`, ver [typst/typst#1421](https://typst.app/docs/#/1421).

<sup>2</sup>A partir de la versión 0.14.

## Generación de diagramas

Se recomienda trabajar con SVGs.

- [draw.io](#): La vieja confiable
- [Excalidraw](#): Un rollito más *cool*
- [PlantUML](#): Lenguaje declarativo para UML
- [CeTZ/fletcher](#): Puro Typst (para *tryhards*)



### Advertencia

Si los SVG dan problemas<sup>1</sup>, exportar como PDF, importar en [Inkscape](#), y re-exportar a SVG.

---

<sup>1</sup>Ver [draw.io FAQ](#).

# Tablas

Echadle un vistazo a la [guía de tablas](#).

También podéis usar un [generador de tablas](#).

```
#figure(  
  table(  
    columns: 2, // o `(1fr, 1fr)`  
    align: horizon, // o `auto` o `(left, right)`  
    [Fila 0, Columna 0], [Fila 0, Columna 1],  
    [Fila 1, Columna 0], [Fila 1, Columna 1],  
  ),  
  caption: [...]  
) <tab:example>
```

Veremos más adelante cómo hacer magia con esto...



# Subfiguras

```
#import "@preview/subpar:0.2.2"
```

```
#subpar.grid(  
  caption: [...]  
  columns: 2,  
  [ #figure(caption: [...], ..) <subfig0>],  
  [ #figure(caption: [...], ..) <subfig1>],  
) <fig0>
```

# Ecuaciones

Sintaxis *similar* a  $\text{\LaTeX}^1$ , pero con menos `\`.

- Echadle un vistazo al [capítulo en Typst Examples Book](#)
- Hay herramientas interactivas como [Typerino](#) para crearlas

TL;DR:

- $x^{2y}$  con `x^(2y)`,  $x_{2y}$  con `x_(2y)`
- texto con `"texto"`,  $xy$  con `x y`, *caca* con `italic("caca")`
- $\frac{2a}{b}$  con `(2a)/b`,  $(a + b)$  con `(a + b)`
- $\rightarrow$  con `->`,  $\leq$  con `<=`,  $\neq$  con `!=`
- $\forall$  con `forall` ó `\forall`,  $\alpha$  con `alpha`,  $\Omega$  con `Omega`,  $\mathbb{N}$  con `NN`  
(ver [sym](#))

---

<sup>1</sup>Conversor de fórmulas de LaTeX a Typst.

# Referencias

Primero se crea una marca (`label`) con `<id>` .

Para referenciarla, basta con usar `@id`

- Dependiendo de lo que hayas marcado (sección, figura, ecuación, etc.), se pondrá un texto automáticamente
- Puedes especificar el texto (suplemento) con `@id[...]`

```
=== Introducción <intro>
```

```
En la @intro...
```

```
$ pi = e = 3 $ <real>
```

```
El @real[Teorema]...
```

## 6.9.1. Introducción

En la Sección 6.9.1...

$$\pi = e = 3 \quad (1)$$

El Teorema 1...

# Referencias

## Bibliografía

Las bibliografías se gestionan con [Bib<sub>La</sub>T<sub>E</sub>X](#) o [Hayagriva](#).

- Entradas en un archivo `.bib` / `.yaml`, con un ID y atributos<sup>1</sup>
- Es recomendable añadir el [DOI](#) siempre que se pueda
- Se referencian con `@entry-id` o, con suplemento, `@entry-id[suplemento]` (`cite`)

Se imprime con:

```
#bibliography("references.bib", style: "ieee")
```

---

<sup>1</sup>Más información sobre Bib<sub>La</sub>T<sub>E</sub>X/Bib<sub>La</sub>T<sub>E</sub>X en la [documentación de CiteDrive](#).

# Scripting

Formado por *expresiones* que se evalúan.

## Variables

- Se declaran con `let`<sup>1</sup>
- Se pueden redeclarar y redefinir

## Bloques

Existen bloques de código ( `{..}` ) y bloques de contenido ( `[..]` )

- Generan un nuevo *scope*<sup>2</sup>
- Ambos se pueden anidar y usar indistintamente

```
#let foo = [Pepe#{42 * 10}] // [Pepe420]
```

---

<sup>1</sup>No es necesario definirlas en la declaración, e.g. `let foo`

<sup>2</sup>Siempre puedes acceder al *scope* padre, pero el hijo sobrescribe

## Tipos

Todos tienen *constructor*, e.g. `str(1)` y *métodos*, e.g. `"a".len()`.

- `content` : Todo lo que se ve en el documento
- `str` : `"E"`, `int` : `5`, `float` : `6.7`, `bool` : `true`, `none`, `auto`
- `array` : `(1, "patata")`, `dictionary` : `(caca: "culo")`
- `fraction` : `1fr`, `length` : `1pt` / `1cm` / `1em`

## Operadores

- Asignación: `=`, `+=`, `-=`, `*=`, `/=`
- Aritméticos: `+`, `-`, `*`, `/`
- Relacionales: `==`, `!=`, `<`, `>`, `<=`, `>=`
- Lógicos: `and`, `or`, `not`; y pertenencia: `in`, `not in`

Para el resto de operaciones, ver módulo `calc`.

## Desestructurado

Esto funciona en todos lados, funciones, etc.

```
let (x, y) = (0, 1)           // x = 0, y = 1
```

```
let (_, y, _) = (0, 1, 2)    // y = 1
```

```
let (a, .., b) = (0, 1, 2, 3) // a = 0, b = 3
```

```
let (x, ..rest) = (0, 1, 2)   // x = 0, rest = (1, 2)
```

## Control de flujo

`if`, `else if`, `else` – la expresión debe evaluar a bool

```
if a < 0 [ negativo ]  
else if a == 0 [ zero ]  
else { calc.sqrt(a) }
```

## Bucles

`while` condition {...}

`for` – «ranged for»:

- `for` value `in` array-or-str {...}
- `for` (key, value) `in` dict {...}

También existen `break` y `continue` .

**Ejemplo** – generar tablas a partir de CSVs:

```
#table(  
  columns: 2,  
  ..for (... , NIA, nota) in csv("alumnos.csv") {  
    (NIA, nota)  
  }  
)
```



## Funciones

Son **puras**, es decir, los parámetros de entrada se pasan por copia.

- Nombradas: `let foo(x) = { .. }`
- *Inline (lambdas)*: `(x) => { .. }`

Parámetros **posicionales** (requeridos) o **nombrados** (opcionales)

```
let foo(a, b: 1) = { a + b }  
foo(1, b: 2)
```

- Parámetros variádicos ( `arguments` ):  
`let foo(x, ..args) = { .. }`
- Retornan todo el bloque, a no ser que se use `return`
- `.with(..)` devuelve una función con parámetros preaplicados
- Parámetros `content` se pueden sacar fuera, e.g. `foo(b: 2)[a]`

## Otras funciones

Os recomiendo echarles un vistazo a estas:

- `v` (espaciado **v**ertical) y `h` (espaciado **h**orizontal)
- `box`
- `grid`
- `align`

Mirad documentación de `array` , `dictionary` .

# Módulos

Puedes separar tus proyectos en archivos `.typ` .

- `#include "file.typ"` inserta el contenido del archivo
- `#import "file.typ"` importa el módulo `file`
  - `#import "file.typ" as foo` : renombra el módulo
  - `#import "file.typ": foo, bar` : importa variables/funciones específicas
  - `#import "file.typ": *` : lo importa todo

Importante tener en cuenta los *path*:

- Son relativos al archivo donde se usan
- `/` (*root*) es el **directorio del archivo que compilamos**
  - Se puede cambiar *root* en la configuración ( `--root` )

**Estilado**

`show` reemplaza elementos, mientras que

`set` sobrescribe parámetros por defecto

## Reglas **set**

Ponen parámetros por defecto en funciones *builtin*.

```
#set text(lang: "es")  
  
#text(["caca"])
```

```
#text(  
  lang: "es", ["caca"]  
)
```

Sólo afectan dentro del *scope* actual:

```
#if epañol {  
  set text(lang: "es")  
} else {  
  set text(lang: "en")  
}  
  
#text["caca"] // no fufa
```

```
#set text(  
  lang: if epañol { "es" }  
       else { "en" }  
)  
  
#text["caca"] // ahora sí
```

# Reglas **show**

Reemplazan (modifican) elementos.

```
#show /* selector */: /* predicado */
```

Los selectores pueden ser:

- Funciones *builtin* → reemplazan los elementos
  - Se puede filtrar por parámetros con `.where()`
- Texto ( `"caca"` ), `regex` , `label` ( `<test>` )
- El resto del documento ( `show: ...` )

Los predicados pueden ser:

- Expresión
- Función anónima `it => { }`
- Regla **set**

## Ejemplos

```
#show "Rust": [Rust 🚀 ]
```

```
#show heading: set text(red)
```

```
#show raw.where(block: false): set raw(lang: "typ")
```

```
#show header: it => block[  
  \~  
  #emph(it.body)  
  #counter(heading).display(it.numbering)  
  \~  
]
```



# **Plantillas y paquetes**

# Otros paquetes útiles

Disponibles en [Typst Universe](#) (400+).

- Bloques de código: `codly`
- Gráficas: `lilaq`
- Algoritmos: `lovelace`
- Teoremas bonicos: `theorion`
- Presentaciones: `touying` , `gentle-clues` , `pinit`
- Glosario: `glosarium`
- Gantt: `timeley`
- Documentación: `tidy`
- Ingeniería de verdad: `physica` , `zero` , `quick-maths`
- Requisitos de *software*: `srs`

# Plantillas para memorias

## Prácticas/trabajos

`guluc3m/report-template-typst`

## TFGs/TFMs

`guluc3m/uc3m-thesis-ieee-typst`

# Más información

- Documentación de Typst
- Typst Examples Book
- Typst Forums
- Typst Discord server
- Guía para usuarios de  $\text{\LaTeX}$
- Typst To  $\text{\TeX}$
- $\text{\MiTeX}$  ( $\text{\LaTeX} \rightarrow \text{\Typst}$ )
- ~~L. D. Casais — Memorias de TFG en  $\text{\LaTeX}$  (2025)~~

A stylized, high-contrast illustration of a person's head and shoulders in profile, facing right. The person's mouth is wide open in a shout, and their hair is depicted with sharp, flame-like spikes. The entire figure is rendered in white against a solid gray background. Overlaid on the person's face is the text '¡Ánimo!' in a bold, black, serif font.

**¡Ánimo!**