The `mantys` template

MANuals for **TY**p**S**t

v1.0.0 2024-12-05 MIT

Helpers to build manuals for Typst packages and templates.

Jonas Neugebauer

GitHub: @jneug

mantys is a Typst template to help package and template authors to write manuals. It provides functionality for consistent formatting of commands, variables and source code examples. The template automatically creates a table of contents and a command index for easy reference and navigation.

For even easier manual creation, mantys works well with Tidy, the Typst docstring parser.

The main idea and design was inspired by the LaTeX package CNLTX by Clemens Niederberger.

Table of Contents

Part I

About

Mantys is a Typst package to help package and template authors write manuals. The idea is that, as many Typst users are switching over from TEX, they are used to the way packages provide a PDF manual for reference. Though in a modern ecosystem there are other ways to write documentation (like mdBook[1] or AsciiDoc[2]), having a manual in PDF format might still be beneficial since many users of Typst will generate PDFs as their main output.

This manual is a complete reference of all of mantys features. The source file of this document is a great example of the things mantys can do. Other than that, refer to the README file in the GitHub repository and the source code for mantys.

I.0.1 Acknowledgements

Mantys was inspired by the fantastic LATEX package CNLTX[3] by Clemens Niederberger[4].

Thanks to @tingerrr and … for contributing to this package and giving feedback.

Thanks to @Mc-Zen for developing Mc-Zen/tidy[5].

[1] https://rust-lang.github.io/mdBook/
[2] https://asciidoc.org
[3] https://ctan.org/pkg/cnltx
[4] clemens@cnltx.de
[5] https://github.com/Mc-Zen/tidy

Part II

Usage

Initialize a new manual using `typst init`:

```
typst init "@preview/mantys" docs
```

> We suggest to initialize the template inside a `docs` subdirectory to keep your manual separated from your packages source files.

If you prefer to manually setup your manual, create a `.typ` file and import mantys at the top:

```
#import "@preview/mantys:1.0.0": *
```

II.0.1 Initializing the template

After importing mantys the template is initialized by applying a show rule with the `#mantys` command.

`#mantys` requires some information to setup the template with an initial title page. Most of the information can be read directly from the `typst.toml` of your package:

```
#show: mantys.with(
  ..toml("typst.toml"),
  ...
)
```

`#mantys(⟨theme⟩: "themes.default")`

Argument

`⟨title-page⟩: auto` — `auto` | `function`

A function that renders a titlepage for the manual. Refer to `#titlepage` for details.

Argument

`⟨examples-scope⟩: (:)` — `dictionary`

Default scope for code examples. The examples scope is a `dictionary` with two keys: `scope` and `imports`. The `scope` is passed to `#eval` for evaluation. `imports` maps module names to a set of imports that should be prepended to example code as a preamble.

Schema:

```
(
 ⟨scope⟩ dictionary
 ⟨imports⟩ dictionary
)
```

For example, if your package is named `my-pkg` and you want to import everything from your package into every examples scope, you can add the following `examples-scope`:

```
examples-scope: (
  scope: (
    pkg: my-pkg
  ),
  imports: (
    pkg: "*"
  )
)
```

For further details refer to #example.

All other arguments will be passed to #titlepage.

All uppercase occurrences of ⟨name⟩ will be highlighted as a packagename. For example MANTYS will appear as mantys.

II.1 The mantys document

```
(
  ⟨title⟩ content
  ⟨subtitle⟩: none content
  ⟨urls⟩: none array of url
  ⟨date⟩: none date
  ⟨abstract⟩: none content
  ⟨package⟩: (:) package
  ⟨template⟩: none template
  ⟨show-index⟩: true boolean
  ⟨show-outline⟩: true boolean
  ⟨show-urls-in-footnotes⟩: true boolean
  ⟨examples-scope⟩: none examples-scope
  ⟨assets⟩: () array of (
      ⟨id⟩ string
      ⟨src⟩ string
      ⟨dest⟩ string
    )
  ⟨git⟩: none (
      ⟨branch⟩: "main" string
      ⟨hash⟩ string
    )
)
```

II.1.0.a Schema for package information

```
(
  ⟨name⟩ string
  ⟨version⟩ version
  ⟨entrypoint⟩ string
  ⟨authors⟩: () author
  ⟨license⟩ string
  ⟨description⟩ string
  ⟨homepage⟩: none url
  ⟨repository⟩: none url
  ⟨keywords⟩: none array of string
  ⟨categories⟩: none array of one of ("components", "visualization" … )
  ⟨disciplines⟩: none array of one of ("agriculture", "anthropology" … )
  ⟨compiler⟩: none version
  ⟨exclude⟩: none array of string
)
```

II.1.0.b Schema for template information

```
(
  ⟨path⟩ string
  ⟨entrypoint⟩ string
  ⟨thumbnail⟩: none string
)
```

II.1.0.c Schema for author information

```
(
  ⟨name⟩ string
  ⟨email⟩: none string
  ⟨github⟩: none string
  ⟨urls⟩: none array of url
  ⟨affiliation⟩: none string
)
```

II.1.1 Accessing document data

There are two methods to access information from the mantys `document`:

1. Using commands from the `document` module or
2. using `#mantys-init` instead of `#mantys`.

II.1.1.a Using the `document` module

II.1.1.b Custom initialization

Instead of using `#mantys` in a `#show` rule, you can initialize mantys using `#mantys-init` directly (`#mantys` essentially is a shortcut for using `#mantys-init`).

`#mantys-init` → `array`

Calling this function will return a tuple with two elements:

[0] The mantys `document`.

[1] The mantys function to be used in a `#show` rule.

```
#let (doc, mantys) = mantys-init(..toml("../typst.toml"))

#show: mantys

This is the manual for #doc.package.name version
#str(doc.package.version).
```

Part III
Documenting commands

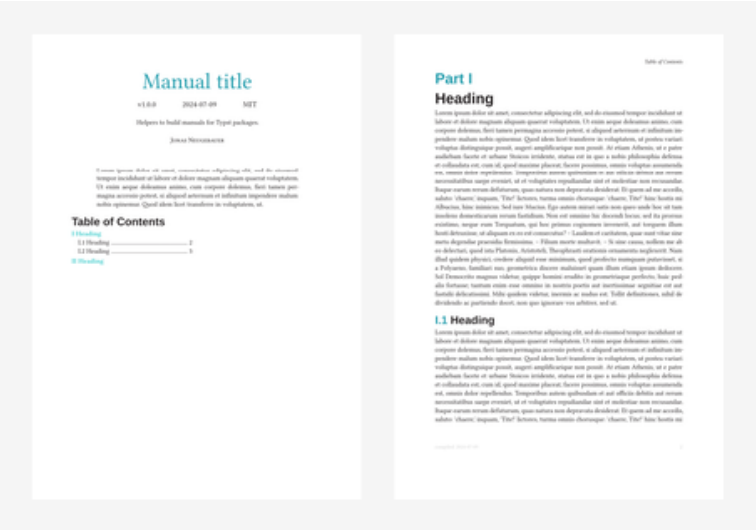
Part IV
Customizing the template

IV.1 Themes

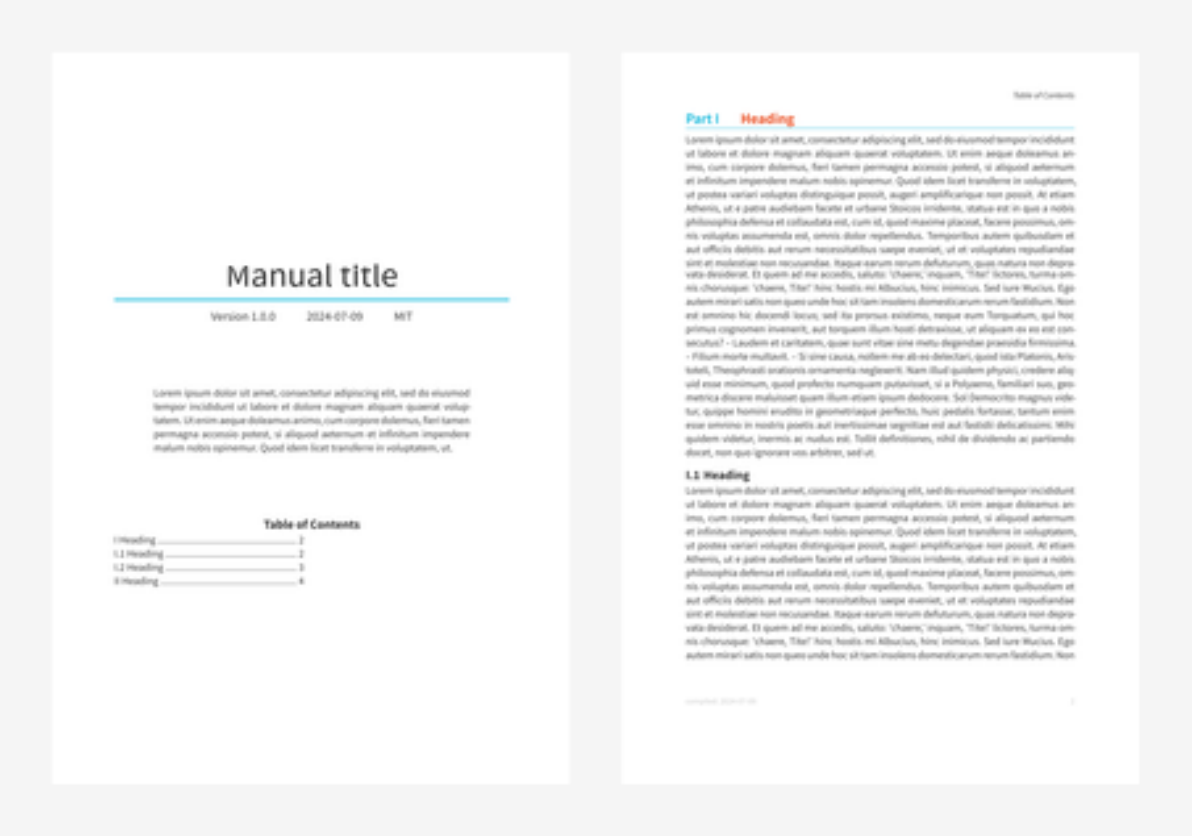
IV.1.1 Using themes

IV.1.2 Bundled themes

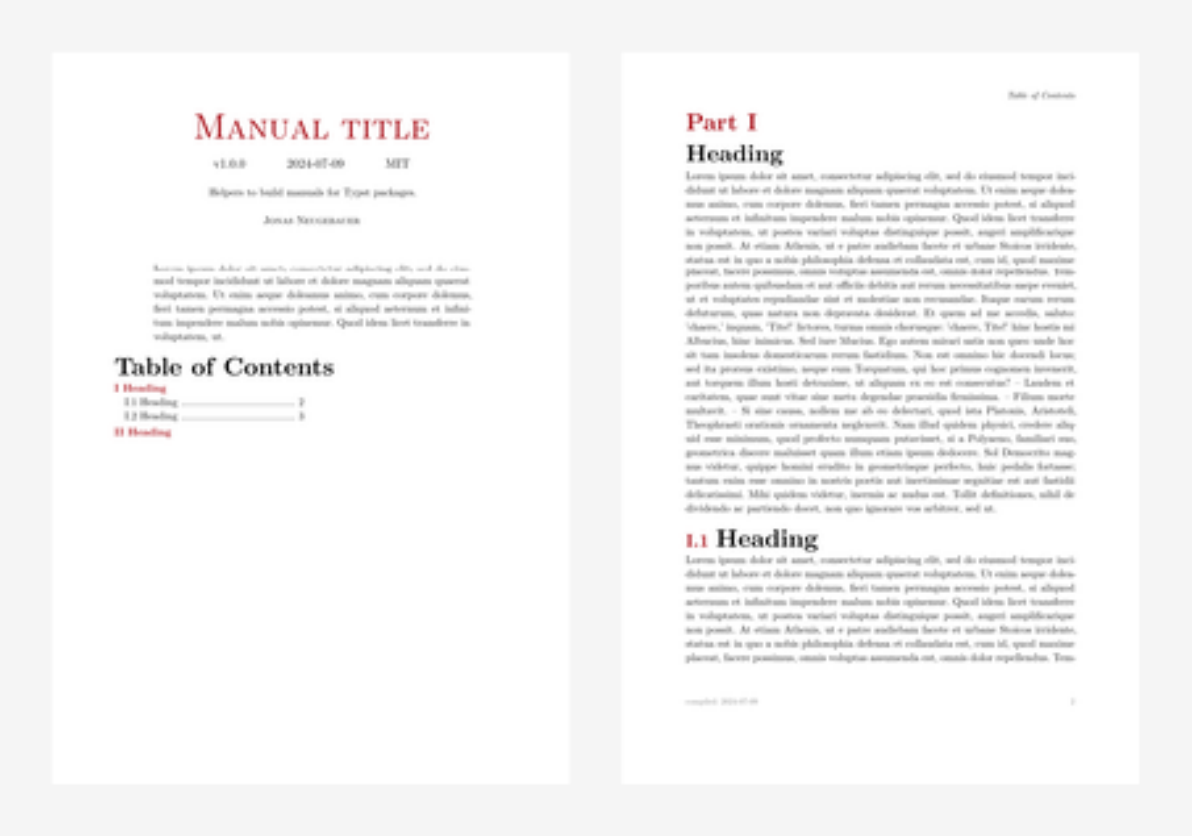
IV.1.2.a Typst theme



IV.1.2.b Modern theme



IV.1.2.c CNLTX theme



IV.1.3 Creating a custom theme

IV.2 The index

mantys adds an index of all commands and custom types to the end of the manual. You can modify this index in several ways.

IV.2.1 Adding entries to the index

Using `#idx` you can add new entries to the index. Entries may be categorized by `⟨kind⟩`. Commands have `⟨kind⟩: "cmd"` set and custom types `⟨kind⟩: "type"`. You may add arbitrary new types. If your package handles colors, you may want to add a “color” category like this:

```
1  idx("red", kind: "color")
```

IV.2.2 Showing index entries by category

The default index can be disabled by passing `⟨show-index⟩: false` to `#mantys`.

To manually show an index in the manual, use `#make-index`

```
#make-index(⟨kind⟩: auto)
```

Shows an index of the specified ⟨kind⟩.

```
#for c in (red, green, yellow, blue) {
  idx(c.to-hex(), kind:"color", display:box(inset:2pt,baseline:3pt,fill:c,
text(white, c.to-hex())))
}

#block(height:8em, columns(2)[
  #make-index(
    kind:"color",
    entry-format: (term, pages) => [#term (#pages.join(", "))\ ],
    grouping: it => it.term.at(1)
  )
])
```

0

#0074d9 (11)

2

#2ecc40 (11)

f

#ff4136 (11)

#ffdc00 (11)

(
 ⟨term⟩ `string`
 ⟨kind⟩ `string`
 ⟨main⟩ `boolean`
 ⟨display⟩ `content`
)

IV.3 Examples

Part V
Available commands

V.1 Utilities

```
#utils.add-preamble          #utils.get-text-color       #utils.rawi
#utils.build-preamble        #utils.place-reference      #utils.split-cmd-name
#utils.get-text              #utils.rawc
```

#utils.add-preamble(⟨code⟩, ⟨imports⟩) → string

Adds a preamble for cutoms imports to ⟨code⟩.

#utils.build-preamble(⟨imports⟩) → string

Creates a preamble to attach to code before evaluating.

Argument

⟨imports⟩ dictionary

Module name - imports pairs, like (mantys: "*").

#utils.get-text[it] → string

Extract text from content.

Argument

⟨it⟩ content

A content item.

#utils.get-text-color(⟨clr⟩, ⟨light⟩: white, ⟨dark⟩: black) → color

Returns a light or dark color, depending on the provided ⟨clr⟩.

```
#utils.get-text-color(red)
─────────────────────────────
luma(100%)
```

Argument

⟨clr⟩ color | gradient

Paint to get the text color for.

Argument

⟨light⟩: white color

Color to use, if ⟨clr⟩ is a dark color.

Argument
⟨dark⟩: black color

Color to use, if ⟨clr⟩ is a light color.

#utils.place-reference(⟨label⟩, ⟨kind⟩, ⟨supplement⟩, ⟨numbering⟩: "1") → content

Places a hidden #figure in the document, that can be referenced via the usual @label-name syntax.

Argument
⟨label⟩ label

Label to reference.

Argument
⟨kind⟩ string

Kind for the reference to properly step counters.

Argument
⟨supplement⟩ string

Supplement to show when referencing.

Argument
⟨numbering⟩: "1" string

Numbering schema to use.

#utils.rawc(⟨color⟩, ⟨lang⟩: none)[code] → content

Shows ⟨code⟩ as inline #raw text (with ⟨block⟩: false) and with the given ⟨color⟩. This supports no language argument, since ⟨code⟩ will have a uniform color.

- #utils.rawc(purple, "some inline code") → some inline code

Argument
⟨color⟩ color

Color for the raw text.

Argument
⟨code⟩ content

String content to be displayed as raw.

#utils.rawi(⟨code⟩, ⟨lang⟩: none) → content

Displays ⟨code⟩ as inline #raw code (with ⟨inline⟩: true).

- #utils.rawi("my-code") → my-code

Argument

⟨code⟩ `string` | `content`

The content to show as inline raw.

Argument

⟨lang⟩: none `string`

Optional language for highlighting.

#utils.split-cmd-name(⟨name⟩) → `dictionary`

Splits a string into a dictionary with the command name and module (if present). A string of the form `"cmd:utils.split-cmd-name"` will be split into `(name: "split-cmd-name", module: "utils")`
Note, that the prefix `cmd:` is removed.

Argument

⟨name⟩ `string`

The command optionally with module and `cmd:` prefix.

V.2 API

#alert	#default	#sarg
#arg	#dtypes	#shortex
#args	#example	#show-git-clone
#argument	#frame	#show-import
#barg	#is-custom-type	#sourcecode
#builtin	#lambda	#typeref
#carg	#meta	#value
#choices	#module	#var
#cmd	#name	#variable
#cmdref	#package	
#command	#property	

#alert(⟨color⟩: blue, ⟨width⟩: 100%, ⟨size⟩: .88em, ..⟨style⟩)[body] → `content`

An alert box to highlight some content.

```
#alert(color:purple, width:4cm)[#lorem(10)]
```

> Lorem ipsum dolor sit amet, consectetur adip-iscing elit, sed do.

Argument

⟨color⟩: blue `color`

Color of the alert.

Argument

⟨width⟩: 100% `length` | `ratio`

Width of the alert box.

Argument

⟨size⟩: .88em `length`

Size of the text.

Argument

..⟨style⟩ `any`

Style arguments to be passed to #block.

Argument

⟨body⟩ `content`

Content of the alert.

#arg(..⟨args⟩, ⟨_value⟩: values.value) → `content`

Shows an argument, either positional or named. The argument name is highlighted with #meta and the value with #value.

- #arg[name] → ⟨name⟩
- #arg("name") → ⟨name⟩
- #arg(name: "value") → ⟨name⟩: "value"
- #arg("name", 5.2) → ⟨name⟩: 5.2

Argument

..⟨args⟩ `any`

Either an argument name (`string`) or a (name: value) pair either as a named argument or as exactly two positional arguments.

#args(..⟨args⟩) → `array`

Creates a list of arguments from a set of positional and/or named arguments.

`string`s and named arguments are passed to #arg, while `content` arguments are passed to #barg. The result is to be unpacked as arguments to #cmd.

```
#cmd( "conditional-show", ..args(hide: false, [body]) )
```

```
#conditional-show(⟨hide⟩: false)[body]
```

Argument

`..⟨args⟩` `any`

Either an argument name (`string`) or a (`name`: `value`) pair either as a named argument or as exactly two positional arguments.

```
#argument(
  ⟨name⟩,
  ⟨is-sink⟩: false,
  ⟨types⟩: none,
  ⟨choices⟩: none,
  ⟨default⟩: "__none__",
  ⟨title⟩: "Argument",
  ⟨_value⟩: values.value
)[body] → content
```

Displays information for a command argument. See the argument list below for an example.

```
#argument("category", default:"utilities")[
  #lorem(10)
]

#argument("category", choices: ("a", "b", "c"), default:"d")[
  #lorem(10)
]

#argument("style-args", title:"Style Arguments",
    is-sink:true, types:(length, ratio))[
  #lorem(10)
]
```

Argument

⟨category⟩: "utilities" `string`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Argument

⟨category⟩: "d" `string`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Style Arguments

..⟨style-args⟩ `length` | `ratio`

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Argument

⟨name⟩ `string`

Name of the argument.

Argument

⟨is-sink⟩: false `boolean`

If this is a veriadic argument .

Argument

⟨types⟩: none `array` | `none`

Array of types to be passed to #dtypes.

Argument

⟨choices⟩: none `array`

Optional array of valid values for this argument.

Argument
`⟨default⟩: "__none__"` `any`

Optional default value for this argument. Will be automatically included in `⟨choices⟩`, if it is missing. To allow `none` as a default value, the default is `"__none__"`.

Argument
`⟨title⟩: "Argument"` `string` | `none`

Sets the title of the argument box.

Argument
`⟨body⟩` `content`

Description of the argument.

`#barg(⟨name⟩)` → `content`

Shows a body argument.

> Body arguments are positional arguments that can be given as a separat content block at the end of a command.

- `#barg[body]` → `[body]`

Argument
`⟨name⟩` `string`

Name of the argument.

`#builtin(⟨name⟩, ⟨module⟩: none)` → `content`

Displays a built-in Typst function with a link to the documentation.

- `#builtin[context]` → `#context`
- `#builtin(module:"math")[clamp]` → `#math.clamp`

Argument
`⟨name⟩` `string` | `content`

Name of the function (eg. `raw`).

Argument
`⟨module⟩: none` `string`

Optional module name.

`#carg(⟨name⟩)` → `content`

Shows a “code” argument.

> “Code” are blocks og Typst code wrapped in braces: `{ ... }`. They are not an actual argument, but evaluate to some other type.

- `#carg[code]` → `{code}`

Argument

`⟨name⟩` `string`

Name of the argument.

`#choices(⟨default⟩: "__none__", ⟨sep⟩: sym.bar.v, ..⟨values⟩)` → `content`

Shows a list of choices possible for an argument.

If `⟨default⟩` is set to something else than `"__none__"`, the value is highlighted as the default choice. If `⟨default⟩` is already present in `⟨values⟩` the value is highlighted at its current position. Otherwise `⟨default⟩` is added as the first choice in the list.

```
#cmd(
  ⟨name⟩,
  ⟨module⟩: none,
  ⟨ret⟩: none,
  ⟨index⟩: true,
  ⟨unpack⟩: false,
  ..⟨args⟩
) → content
```

Renders the command `⟨name⟩` with arguments and adds an entry with `⟨kind⟩: "cmd"` to the index.

`⟨args⟩` is a collection of positional arguments created with `#arg`, `#barg` and `#sarg` (or `#args`).

All positional arguments will be rendered first, then named arguments and all body arguments will be added after the closing paranthesis. The relative order of each argument type is kept as given.

```
- #cmd("cmd", arg[name], sarg[args], barg[body])
- #cmd("cmd", ..args("name", [body]), sarg[args], module:"mod")
- #cmd("clamp", arg[value], arg[min], arg[max], module:"math", ret:int,
unpack:true)
```

- `#cmd(⟨name⟩, ..⟨args⟩)[body]`
- `#mod.cmd(⟨name⟩, ..⟨args⟩)[body]`
- `#math.clamp(`
  `⟨value⟩,`
  `⟨min⟩,`
  `⟨max⟩`
  `) →` `integer`

Argument

`⟨name⟩` `string`

Name of the command.

Argument

`⟨module⟩: none` `string`

Name of a module, the command belongs to.

Argument

`⟨ret⟩: none` `any`

Returned type.

Argument

`⟨index⟩: true` `boolean`

Whether to add an index entry.

Argument

`⟨unpack⟩: false` `boolean`

If `true`, the arguments are shown in separate lines.

Argument

`..⟨args⟩` `any`

Arguments for the command, created with individual argument commands (`#arg`, `#barg`, `#sarg`) or `#args`.

`#cmdref(⟨name⟩, ⟨module⟩: none)`

Creates a reference to the command ⟨name⟩. This is equivalent to using @cmd:name.

- #cmdref("builtin") → #builtin
- @cmd:builtin → #builtin

#command(⟨name⟩, ⟨label⟩: auto, ..⟨args⟩)[body] → **content**

Displays information of a command by formatting the name, description and arguments. See this command description for an example.

The command is formated with #cmd and an index entry is set, that is marked as the “main” index entry for this command.

Argument

⟨name⟩ — string

Name of the command.

Argument

⟨label⟩: auto — string | auto | none

Custom label for the command.

Argument

..⟨args⟩ — content

List of arguments created with the argument functions (#arg, #barg, #sarg) or #args.

Argument

⟨body⟩ — content

A description for the command.

#default(⟨value⟩, ⟨parse-str⟩: true) → **content**

Highlights the default value of a set of #choices.

- #default("default-value") → default-value
- #default(true) → true

Argument

⟨value⟩ — any

The value to highlight.

#dtypes(..⟨types⟩, ⟨link⟩: true, ⟨sep⟩: box(inset: (left: 1pt, right: 1pt), sym.bar.v))

Creates a list of datatypes.

```
#example(
  (side-by-side): false,
  (scope): (:),
  (imports): (:),
  (use-examples-scope): true,
  (mode): "markup",
  (breakable): false,
  ..(args)
)[example-code]
```

Show an example by evaluating the given `#raw` code with Typst and showing the source and result in a `#frame`.

See Section IV.3 for more information on sourcecode and examples.

Argument

`(side-by-side): false` `boolean`

Shows the source and example in two columns instead of the result beneath the source.

Argument

`(scope): (:)` `dictionary`

A scope to pass to `#eval`.

Argument

`(use-examples-scope): true` `boolean`

Set to `false` to not use the gloabl examples scope.

Argument

`(mode): "markup"` `string`

The evaulation mode: `"markup"`|`"code"`|`"math"`

Argument

`(breakable): false` `boolean`

If the frame may brake over multiple pages.

Argument

`(example-code)` `content`

A `#raw` block of Typst code.

Argument

`..(args)` `content`

An optional second positional argument that overwrites the evaluation result. This can be used to show the result of a sourcecode, that can not evaulated directly.

#frame(..⟨args⟩) → **content**

Create a frame around some content.

Uses showybox and can take any arguments the #showybox command can take.

```
#frame(title:"Some lorem text")[#lorem(10)]
```

Some lorem text

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

..args (any): Arguments for Showybox.

#is-custom-type(⟨name⟩)

⚠ **requires-context**

#lambda(..⟨args⟩, ⟨ret⟩: none) → **content**

Create a lambda function argument. Lambda arguments may be used as an argument value with arg. To show a lambda function with an argument sink, prefix the type with two dots.

- `#lambda(int, str)` → `(integer, string)→none`
- `#lambda("ratio", "length")` → `(ratio, length)→none`
- `#lambda("int", int, ret:bool)` → `(integer, integer)→boolean`
- `#lambda("int", int, ret:(int,str))` → `(integer, integer)→(integer, string)`
- `#lambda("int", int, ret:(name: str))` → `(integer, integer)→(name:string)`

Argument

`..⟨args⟩` — `type | string`

Argument types of the function parameters.

Argument

`⟨ret⟩: none` — `type`

Type of the returned value.

#meta(⟨name⟩, ⟨l⟩: sym.angle.l, ⟨r⟩: sym.angle.r) → **content**

Highlight an argument name.

`#meta[variable]` → ⟨variable⟩

Argument

⟨name⟩ `string` | `content`

Name of the argument.

Argument

⟨l⟩: sym.angle.l `string` | `content` | `symbol`

Prefix to ⟨name⟩.

Argument

⟨r⟩: sym.angle.r `string` | `content` | `symbol`

Suffix to ⟨name⟩.

#module(⟨name⟩)

Show a module name.

- #module("util") → util

Argument

⟨name⟩ `string`

Name of the module.

#name(⟨name⟩, ⟨last⟩: none) → `content`

Highlight human names (with first- and lastnames).

- #name("Jonas Neugebauer") → Jonas Neugebauer
- #name("J.", last:"Neugebauer") → J. Neugebauer

Argument

⟨name⟩ `string`

First or full name.

Argument

⟨last⟩: none `string`

Optional last name.

#package(⟨name⟩)

Show a package name.

- #package("codelst") → codelst

Argument

⟨name⟩ `string`

Name of the package.

#property(..⟨args⟩)

Shows a command propertie (annotation). This should be used in the `[body]` of `#command` to annotate a function with some special meaning.

Properties are provided as named arguments to the `#property` function.

The following properties are currently known to mantys:

requires-context `boolean` Requires a function to be used inside `#context`.

> ⚠ `requires-context`

since `version` | `string` Marks this function as available since a given package version.

> i since version **1.0.0**

until `version` | `string` Marks this function as available until a given package version.

> i until version **0.1.4**

deprecated `boolean` | `version` | `string` Marks this function as deprecated. If set to a version, the function is supposed to stay availalbel until the given version.

> ⚠ this function is `deprecated` and will be removed in version **1.0.1**

see `array` **of** `string` | `label` Adds references to other commands or websites.

> ↗ see `#mantys`, https://github.vom/jneug/typst-mantys

todo `string` | `content` Adds a todo note to the function.

> ✅ TODO:
> - Add documentation.
> - Add ⟨foo⟩ paramter.

Other named properties will be shown as is:

```
module: utilities
```

`#sarg((name)) → content`

Shows an argument sink.

- `#sarg[args] → ..⟨args⟩`

Argument

⟨name⟩ `string`

Name of the argument.

`#shortex((sep): [ #sym.arrow.r ], (mode): "markup", (scope): (:))[code] → content`

Show a “short example” by showing ⟨code⟩ and the evaluation of ⟨code⟩ separated by ⟨sep⟩. This can be used for quick one-line examples as seen in `#name` and other command docs in this manual.

```
- #shortex(`#name("Jonas Neugebauer")`)
- #shortex(`#meta("arg-name")`, sep: ": ")
```

- `#name("Jonas Neugebauer")` → Jonas Neugebauer
- `#meta("arg-name")`: ⟨arg-name⟩

Argument

⟨code⟩ `content`

The `#raw` code example to show.

Argument

⟨sep⟩: `[ #sym.arrow.r ]` `content`

The separator between ⟨code⟩ and its evaluated result.

Argument

⟨mode⟩: `"markup"` `string`

One of `"markup"`|`"code"`|`"math"`

Argument

⟨scope⟩: `(:)` `dictionary`

A scope argument similar to `examples-scope`.

#show-git-clone(⟨repository⟩: auto, ⟨out⟩: auto)

Shows an import statement for this package. The name and version from the document are used.

```
#show-git-clone()
#show-git-clone(repository: "typst/packages", out:"preview/mantys/1.0.0")
```

```
git clone https://github.com/jneug/typst-mantys mantys/1.0.0
```

```
git clone https://github.com/typst/packages mantys/1.0.0
```

Argument

⟨repository⟩: auto `string` | `auto`

Custom package repository to show.

#show-import(⟨repository⟩: "@preview", ⟨imports⟩: "*", ⟨mode⟩: "markup")

Shows an import statement for this package. The name and version from the document are used.

```
#show-import()
#show-import(repository: "@local", imports: "mantys", mode:"code")
```

```
#import "@preview/mantys:1.0.0": *
```

```
import "@local/mantys:1.0.0": mantys
```

Argument

`⟨repository⟩: "@preview"` `string`

Custom package repository to show.

Argument

`⟨imports⟩: "*"` `string` | `none`

What to import from the package. Use `none` to just import the package into the global scope.

Argument

`⟨mode⟩: "markup"` `string`

One of `"markup"`|`"code"`. Will show the import in markup or code mode.

`#sourcecode((title): none, (file): none, ..(args))[code]` → `content`

Shows sourcecode in a `#frame`.

> Uses codelst[6] to render the code.

See Section IV.3 for more information on sourcecode and examples.

[6] https://typst.app/universe/package/codelst

```
#sourcecode(title:"Example", file:"sourcecode-example.typ")[```
#let module-name = "sourcecode-example"
```]
```

---

Example — *sourcecode-example.typ*

```
1  #let module-name = "sourcecode-example"
```

Argument

⟨title⟩: none — string

A title to show on top of the frame.

Argument

⟨file⟩: none — string

A filename to show in the title of the frame.

Argument

..⟨args⟩ — any

Argumente für #codelst.sourcecode

Argument

⟨code⟩ — content

A #raw block of Typst code.

#typeref(⟨name⟩)

Creates a reference to the custom type ⟨name⟩. This is equivalent to using @type:name.

#value(⟨value⟩, ⟨parse-str⟩: false) → **content**

Shows ⟨value⟩ as content.

- #value("string") → "string"
- #value([string]) → [string]
- #value(true) → true
- #value(1.0) → 1.0
- #value(3em) → 3em
- #value(50%) → 50%
- #value(left) → left
- #value((a: 1, b: 2)) → (a: 1, b: 2)

Argument

⟨value⟩ — any
```

Value to show.

#var(⟨name⟩, ⟨module⟩: none, ⟨index⟩: true) → content

Shows the variable ⟨name⟩ and adds an entry to the index.

- #var[colors] → #colors

Argument

⟨name⟩ — string | content

Name of the variable.

Argument

⟨module⟩: none — string

Optional name of a module, the function is in.

Argument

⟨index⟩: true — boolean

Set to false to not add this location to the index.

#variable(⟨name⟩, ⟨types⟩: none, ⟨value⟩: none, ⟨label⟩: auto)[body] → content

Displays information for a variable defintion.

```
#variable("primary", types:("color",), value:green)[
  Primary color.
]
```

#primary: rgb("#2ecc40") — color

Primary color.

Argument

⟨name⟩ — string

Name of the variable.

Argument

⟨types⟩: none — array

Array of types to be passed to #dtypes.

Argument

⟨value⟩: none — any

Default value.

Argument

⟨body⟩ `content`

Description of the variable.

#cmd- `content`

Same as #cmd, but does not create an index entry.

#var- `content`

Same as #var, but does not create an index entry.

#codesnippet `content`

Shows some #raw code in a #frame, but without line numbers or other enhancements.

> By default mantys wrapps any #raw code in the manual in this command.

```typc
let a = "some content"
[Content: #a]
```

```
1 let a = "some content"
2 [Content: #a]
```

#side-by-side `content`

Same as #example, but with ⟨side-by-side⟩: true set.

Part VI

Index

#

A

B

C

D

E

F

G

I

L

M

N

P

R

S

T

V