Minimal Manuals
Manual

Simple and sober manuals inspired by the OG Linux manpages.

CONTENTS

---

[*]https://www.github.com/mayconfmelo

QUICK START

```
#import "@preview/min-manual:0.1.1": manual
#show: manual.with(
  title: "Package Name",
  description: "Short description, no longer than two lines.",
  authors: "Author <mailto:author@email.com>",
  cmd: "pkg-name",
  version: "0.4.2",
  license: "MIT",
  logo: image("assets/logo.png")
)
```

DESCRIPTION

Generate modern manuals, without loosing the simplicity and looks of old manuals. This package draws inspiration from the Linux manpages, as they look in terminal emulators until today, and adapts it to the contemporary formatting possibilities.

The package is designed to universally document any type of program or code, including Typst packages and templates. It allows to create documentation separated in dedicated files or extract it from the source code itself through doc-comments.

This manual will be updated only when new versions break or modify something; otherwise, it will be valid to all newer versions starting by the one documented here.

OPTIONS

Those are the full list of options available and its default values:

```
#import "@preview/min-manual:0.1.1": manual
#show: manual.with(
  title: none,
  description: none,
  authors: none,
  package: none,
  cmd: none,
  version: none,
  license: none,
  logo: none,
  manual-author: none,
  from-comments: none,
  toc: false,
  paper: "a4",
  lang: "en",
  justify: true,
```

```
    line-space: 0.65em,
    par-margin: 1.2em,
    margin: 1in,
    font: "Arial",
    font-size: 12pt
)
```

Seems like an awful lot to start with, but let's just break down all this to understand it better, shall we?

`title:` `string` `content`   (*required*)

   The longer, descriptive and more readable name of what is been documented.

`description:` `string` `content`

   A short description of what is being documented; generally two lines long or less.

`authors:` `string` `array`

   The author or authors of what is being documented — not the manual. When more than one author, is an array of strings, in format (`"NAME <URL>"`, `"NAME <URL>"`), where `<URL>` is optional.

`package:` `string`

   Set `cmd` and `version` using Typst package notation: `CMD:VERSION`. Takes precedence over the `cmd` and `version` arguments, and shadows them if they are set too.

`cmd:` `string`

   The command or code name used to invoke what is being documented. Set only if `package` argument is not used.

`version:` `string`

   The version of what is being documented. Useful when different versions have different behaviors. Set only if `package` is not used.

`license:` `string` `content`

   The license of what is being documented — not the manual.

`logo:` `image`

   The logo image of what is being documented.

`manual-author:` `string`

The author of the manual — not what is being documented. If not set, fallback to the first (or only) one of `authors`.

`from-comments:` `string`

The source code from which the doc-comments will be extracted; when set, activates the doc-comments gathering and ignores the content of the current file. To obtain a external source code as string, just `#read` it.

`toc:` `boolean`

Defines whether the manual will have a table of contents or not.

`paper:` `string`

Defines the page paper type — and its size therefore.

`lang:` `string`

Defines the language of the text.

`justify:` `boolean`

Defines if the text will have justified aliment.

`line-space:` `length`

Defines the space between lines in the document.

`par-margin:` `length`

Defines the margin space after each paragraph. Set it the same as `line-space` to remove get paragraphs without additional space in between.

`margin:` `length`

Defines the document margins.

`font:` `string` `array`

Defines the font families used for the text: a principal font and its fallback.

`font-size:` `length`

Defines the size of the text in the document.

ARGUMENT COMMAND

```
#import "@preview/min-manual:0.1.1": arg
#arg(
  name,
  body
)
```

This command offers a convenient way to document the arguments — or parameters, or options, or whatever they are called; and even structures can be easily explained.

title `string`   (*required*)

Defines the argument `NAME`, `TYPE`s, and if it is `REQUIRED` using the following syntaxes:

$$\text{"NAME <- TYPE | TYPE | TYPE <REQUIRED>"}$$

$$\text{"NAME -> TYPE | TYPE | TYPE"}$$

The name can be ` ```LANG NAME``` ` to get syntax highlight. The `<-` arrow indicates that `NAME` receives any of the value `TYPE`s, and `->` indicates that `NAME` returns one of the value `TYPE`s. A special `nothing` type is used when nothing is received and/or returned. For optional arguments, just don't write the `<REQUIRED>`.

body `content`

A brief description of what the argument does.

EXTRACT COMMAND

Allows the extraction of code structures, usually functions, directly from the source code to illustrate and enrich the documentation examples. It works like this: it reads a source code file and seeks a specific structure within code; and when finding it, prints it in the document. It is important to note that, for this command to work, it must have access to the source code that contains the structure in question as a string, this can be done by `#read` the source code file.

```
#extract(
  name: none,
  namespace: none,
  rule: none,
  model: "(?s)#?let\s+<name>\((.*)\)\s*=",
  lang: "typm",
  body
)
```

`name:` `string`   (*required*)

> The name of the structure that will be extracted from the code. If more than one are found, matches the last one.

`namespace:` `string`

> The module namespace used before the structure name itself, like in `namespace.name`.

`rule:` `string`

> Used only when extracting Typst functions: Determines the rule used to present the Typst functions extracted. If `none`, it shows the syntax `#let NAME(ARGS)`; if `"show"`, shows the syntax `#show: NAME.with(ARGS)`; and if `"set"`, shows the syntax `#set NAME(ARGS)`.

`model:` `string`

> A string containing a regex pattern, used to capture and extract the code structure; by default, captures the Typst function structure. The string can have a special `<name>` pattern that will be replaced by the `#extract(name)` argument to get the structure with this name.

`lang:` `string`

> The programming language used for the syntax highlight.

`body` `string`   (*required*)

> The source code from which the structure will be extracted; usually obtained by `#read` the source code file.

As an example of documentation for files in other languages, to extract a Rust function called `main` from a file called `src/lib.rs` just use:

```
#extract(
  name: "main",
  model: "\s*fn\s*<name>\(.*\)",
  lang: "rust",
  read("src/lib.rs")
)
```

The `#extract` command is in early development and is currently an experimental feature, therefore it may or may not present errors, usability problems, and unexpected behaviors.

PACKAGE CITATION COMMANDS

```
#import "@preview/min-manual:0.1.1": univ, pip, crate, gh, pkg
#univ(name)
#pip(name)
#crate(name)
#gh(name, user)
#pkg(name, url)
```

These are small helper commands that simplifies the citation of any type of external package, crate, or library using its repository URL. The `#univ` command is used to Typst Universe packages, the `#pip` to Pip/Pypi Python modules, the `#crate` to Rust crates, the `#gh` to GitHub repositories, and `#pkg` to any other repositories in general.

`name` `string` (*required*)

The name of the package, or library, or crate, or anything else, as it appears in the package repository, e.g.: just the `babel` of `https://ctan.org/pkg/babel`.

`url` `string` (*required*)

Used only by `#pkg`: The package repository URL without package name path, e.g.: just the `https://ctan.org/pkg/` of `https://ctan.org/pkg/babel`.

`user` `string` (*required*)

Used only by `#gh`: The GitHub user, as it appears in GitHub repositories, e.g.: just the `typst` of `https://github.com/typst/packages`.

DOC-COMMENTS

Doc-comments provide a way to embed documentation within the code itself: they are special comments in the source code that make it possible to generate complete documentation without creating and maintaining another file just for this.

```
/// This is an inline doc-comment!

/**
 * This is a block doc-comment!
 * Use as many lines as you want.
 * The * at the start of each line is optional.
**/
```

For doc-comments to work properly, it is necessary that the programming language of the source code recognize them as mere comments — as is the case with Typst,

Rust, and JavaScript, for example. The differences between comments and doc-comments are minimal, making them discrete in the code:

Doc-Comments	Comments
///	//
/** **/	/* */

Within these doc-comments it is possible to write Typst code, and also to use the *min-manual* commands and features without import them. If you need to perform advanced tasks and queries from within doc-comments, all the content of the source code itself is available in `#from-comments` variable inside doc-comments.

The doc-comments are in early development and are currently an experimental feature, therefore it may or may not present errors, usability problems, and unexpected behaviors.

ARGUMENT COMMANDS IN DOC-COMMENTS

To better fit in the code, the `#arg` command can be invoked on doc-comments by using a special syntax:

```
#let function-name(
  arg-name,
  /** <- string | content | none <required>
    * This argument does something. **/
) = {...}
```

This is a very useful shorthand to write this same doc-comment:

```
/**
 * #arg("arg-name <- string | content | none <required>")[
 *   This argument does something
 * ]
**/
```

Much cleaner, isn't it? To use this syntax, just follow a few rules: the doc-comment syntax must be

```
/** <- TYPES \n BODY **/
```

or else

```
/// <- TYPES
```

And the arrow <- must always come right after the doc-comment opening, i.e., /** <- or else /// <-.

EXTRACT COMMANDS IN DOC-COMMENTS

The `#extract` command can also be invoked in doc-comments using a special syntax:

```
/// :function-name: show `typm` "model-regex"
#let function-name(
  arg-name,
) = {...}
```

This short syntax is equivalent to this much more cumbersome doc-comment:

```
/**
 * #extract(
 *   name: "function-name",
 *   rule: "show",
 *   lang: "typm",
 *   model: "model-regex",
 *   from-comments
 * )
**/
```

In this syntax, only `:NAME:` is mandatory, while `RULE`, `` `LANG` ``, and `"MODEL"` will fallback to the defaults when omitted. Can be a `:NAMESPACE.NAME:` syntax when `lang` is Typst-related.

COPYRIGHT

Copyright © 2025 Maycon F. Melo.
This manual is licensed under MIT terms and rights.
The manual source code is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

The logo was obtained from Flaticon website.