Minimal Manuals

Manual

Simple and sober manuals inspired by the OG Linux manpages.

---

min-manual	0.1.0	MIT

Maycon F. Melo*

CONTENTS

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

QUICK START

```
#import "@preview/min-manual:0.1.0": 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:0.1.0": manual
#show: manual.with(
  title: none,
  description: none,
  authors: none,
  cmd: none,
  version: none,
  license: none,
  logo: none,
  manual-author: 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 (*required*)

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.

`cmd:` string content

The command or code name used to invoke what is being documented.

`version:` string content

The version of what is being documented. Useful when different versions have different behaviors.

`license:` string content (*required*)

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`.

`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:0.1.0": arg
#arg(
  name,
  types,
  required: false,
  body
)
```

This command offers a convenient way to document the arguments — or parameters, or options, or whatever they are called.

`name` string raw (*required*)

The argument name; can have syntax highlight if set as a raw text, like ```` ```LANG NAME``` ````, where `LANG` is the programming language name.

`types` string array none

The list of types that the argument value can be.

`required:` boolean

Defines if the argument is mandatory.

`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 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,
  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.

`rule:` string

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

`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 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")
)
```

PACKAGE CITATION COMMANDS

```
#import "@preview/min-manual:0.1.0": 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*)

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

user string (*required*)

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

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 recognizes them as mere comments — as is the case with Typst, Rust, and JavaScript. The difderences 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.

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 shorhand to write this same doc-comment:

```
/**
 * #arg(
 *   "arg-name", ("string", "content", "none"),
 *   required: true
 * )[This argument does something]
**/
```

Much smaller, isn't it? To use this syntax, just follow a few rules: block doc-comments `/** **/` must be used if there is an argument description; if there isn't any argument description, inline doc-comments `///` can be used; the arrow `->` must come right after the doc-comment opening, i.e., `/** ->` or `/// ->`; for optional arguments, just don't write `<required>`.

EXTRACT COMMANDS IN DOC-COMMENTS

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

```
/// :function-name: show `typm`
#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",
 *   from-comments
 * )
**/
```

COPYRIGHT

The logo was obtained from Flaticon website.