

zprint

zprint is a library and command line tool providing a variety of pretty printing capabilities for both Clojure code and Clojure/EDN structures. It can meet almost anyone's needs. As such, it supports a number of major source code formatting approaches.

clojars [zprint "1.2.9"]

Quickstart

- Latest pre-compiled binaries for macOS and Linux are here on GitHub
- Run the same code in babashka 
- Library to use in the REPL or embed in your project: clojars [zprint "1.2.9"]

Overview

zprint does far more than just properly indent code. **Before:**

```
1 (defn change-start-column [new-start-column style-vec [inline-comment-
2   start-column spaces-before :as comment-vec]] (if (zero? inline-
3   comment-index)
4   style-vec (let [delta-spaces (- new-start-column start-column) new-
5   spaces
6   (+ spaces-before delta-spaces) previous-element-index (dec
7   inline-comment-index) [s c e :as previous-element] (nth style-vec
8   previous-element-index) new-previous-element (cond (= e :indent) [(
9   str "\n"
10  (blanks new-spaces)) c e] (= e :whitespace) [(str (blanks new-spaces)
11  )
12  c e 26] :else nil)] (assoc style-vec previous-element-index
13  new-previous-element))))
```

After:

```
1 (defn change-start-column
2   [new-start-column style-vec
3    [inline-comment-index start-column spaces-before :as comment-vec]]
4   (if (zero? inline-comment-index)
5       style-vec
6       (let [delta-spaces (- new-start-column start-column)
7             new-spaces (+ spaces-before delta-spaces)
8             previous-element-index (dec inline-comment-index)
```

```

9      [s c e :as previous-element] (nth style-vec previous-element-
      index)
10     new-previous-element
11     (cond (= e :indent) [(str "\n" (blanks new-spaces)) c e]
12           (= e :whitespace) [(str (blanks new-spaces)) c e 26]
13           :else nil])
14     (assoc style-vec previous-element-index new-previous-element))))

```

Recent Additions!

- A new style, `:sort-require`, will sort the requires in an `ns` macro, as suggested by how-to-ns. This will interoperate with `:ns-justify` well, but be sure and put the `:sort-require` to the left of (i.e., before) the `:ns-justify`. This will sort the elements of the `:refer` vector as well.
- `:ns-justify` has been modified to support explicit parameters for the variance in the `:require`, `:require-macros` and `:import` sections of the `ns` macro. They are `:require-max-variance`, `:require-macros-max-variance` and `:import-max-variance` and may be used in a style map. For example `{:style-call :ns-justify :require-max-variance 1000}` will try as hard as possible to justify the `:require` list in an `ns` macro.
- A new pre-compiled binary is available for macOS running on Apple Silicon. While the macOS Intel binary runs fine on Apple Silicon, the Apple Silicon binary runs considerably (up to 3x) faster! Download `zprintma-1.2.9` from the release to get the Apple Silicon version.
- You can now run `zprint` as a `babashka` task or use `bbin`. It starts very quickly and runs faster than the `uberjar` on even very large files. If using a task, you don't need to install a new version, just edit `bb.edn`. See the simple details [here](#)
- Important updates and fixes for comment wrapping changes first available in 1.2.6. Avoid 1.2.6, use 1.2.7 or later.
- Comment wrapping has been considerably altered. When working on the stability fixes for 1.2.5, the largest remaining problem was comment wrapping causing changes to the formatting in subsequent runs. In addition, the comment wrapping has been very simplistic since its inception, leaving wrapped comments looking pretty bad. There is a new capability called `{:comment {:smart-wrap? true}}` which will now word wrap comments cleanly. It will also repair most of the problems that the simplistic wrapping produced in the past. It is now the default, in no small part to repair the problems of the past. If you are working to minimize changes when running `zprint`, I would recommend running it once over your code before you disable it, as will clean up most of the problems that were added by `zprint` in the past. You can disable it with `{:comment {:smart-wrap? false}}`. You can also configure it to minimize the amount of word wrapping it does, while still allowing it to do much better than

the previous default by using `{:style :minimal-smart-wrap}`. You need to have `{:comment {:smart-wrap? true}}` to use `:minimal-smart-wrap`. Smart wrap works hard to not wrap things like numbered or bulleted lists. If you have a case where it wraps something that it shouldn't, please submit an issue. It is likely that it can be fixed with a configuration change. See the reference manual for more details on how to configure smart wrap.

- You can now specify some keys to come last in a map as well as some keys to appear first in a sorted map. The `{:map {:key-order [...]}}` configuration places all of the keys prior to the distinguished key `:|` at the front of the map, and all of the keys after the `:|` key at the end of the map.
- All changes

See zprint:

- **classic zprint** – ignores whitespace in function definitions and formats code with a variety of heuristics to look as good as hand-formatted code (*see examples*)
- **respect blank lines** – similar to classic zprint, but blank lines inside of function definitions are retained, while code is otherwise formatted to look beautiful (*see examples*)
- **indent only** – very different from classic zprint – no code ever changes lines, it is only correctly indented on whatever line it was already on (*see examples*)

In addition, zprint is very handy **to use at the REPL**.

Use zprint:

- to format whole files
- while using an editor
- at the REPL
- with a team
- with different formatting for different projects
- to format a range of lines in a file
- to format a babashka script
- to correct indentation but not otherwise reformat a file
- and have it run even faster
- from inside a Clojure(script) program

Get zprint:

- a standalone binary for macOS *starts in <50 ms*
- a standalone binary for Linux *starts in <50 ms*
- a VS Code extension for zprint
- using babashka
- an uberjar for any Java enabled platform *starts in several seconds*
- an accelerated uberjar for any Java enabled platform *starts in about 1s*
- a library to use at the REPL
- other ways to access zprint

Get something other than the default formatting:

Without learning how to configure zprint:

Maybe one of the existing “styles” will meet your needs. All you have to do is put `{:style ...}` on the command line or as the third argument to a zprint call. For example, `{:style :community}` or `{:style :respect-bl}`.

Some commonly used styles:

- Format using “community” standards
- Respect blank lines
- Indent Only
- Respect all newlines
- Detect and format hiccup vectors
- Justify all pairs
- Backtranslate `quote`, `deref`, `var`, `unquote` in structures
- Detect keywords in vectors, if found respect newlines
- Sort dependencies in `project.clj`
- Support “How to ns”

Learn how to alter zprint’s formatting behavior:

- How do I change zprint’s behavior?

I want to change...

- how user defined functions are formatted

-
- the indentation in lists
 - the configuration to track the “community” standard
 - how blank lines in source are handled
 - how map keys are formatted
 - the colors used for formatting source
 - how the second element of a pair is indented
 - how comments are handled
 - how blank lines are handled at the top level
 - how vectors are formatted based on their content
 - how constants are defined when formatting constant pairs
 - the options map by defining functions to format based on content
 - anything else...

Usage

clojars [zprint "1.2.9"]

 babashka compatible

Clojure 1.9, 1.10, 1.10.3, 1.11.1, 1.12.0-alpha8:

Leiningen (via Clojars)

clojars [zprint "1.2.9"]

Clojurescript:

zprint has been tested in each of the following environments:

- figwheel-main 0.2.16 (Clojurescript 1.11.4)
- shadow-cljs 2.18.0
- [planck](#) 2.27.0 (Clojurescript 1.11.60)

It requires [tools.reader](#) at least 1.0.5, which all of the environments above contain.

Clojure 1.8:

The last zprint release built with Clojure 1.8 was [zprint “0.4.15”].

In addition to the zprint dependency, you also need to include the following library when using Clojure 1.8:

```
1 [clojure-future-spec "1.9.0-alpha17"]
```

The zprint Reference

- Entire reference document
- What does zprint do?
- Features
- The zprint API
- Configuration
 - Configuration uses an options map
 - Where to put an options map
 - **Simplified Configuration** – using `:style`
 - * Respect blank lines
 - * Indent Only
 - * Format using “community” standards
 - * Respect all newlines
 - * Detect and format hiccup vectors
 - * Justify all pairs
 - * Backtranslate `quote`, `deref`, `var`, `unquote` in structures
 - * Detect keywords in vectors, if found respect newlines
 - * Sort dependencies in `project.clj`
 - * Support “How to ns”
 - * Add newlines between pairs in `let` binding vectors
 - * Add newlines between `cond`, `assoc` pairs
 - * Add newlines between extend clauses
 - * Add newlines between map pairs
 - * Prefer hangs and improve performance for deeply nested code and data
 - Options map format
 - * Option Validation
 - * What is Configurable
 - Generalized Capabilities
 - Syntax Coloring
 - Function Classification for Pretty Printing
 - * Changing or Adding Function Classifications

-
- ★ Replacing functions with reader-macros
 - ★ Controlling single and multi-line output
 - ★ A note about two-up printing
 - ★ A note on justifying two-up printing
 - Formatting large or deep collections
 - Widely Used Configuration Parameters
 - **Configurable Elements**
 - ★ :agent
 - ★ :array
 - ★ :atom
 - ★ :binding
 - ★ :comment
 - ★ :delay
 - ★ :extend
 - ★ :fn
 - ★ :future
 - ★ :list
 - ★ :map
 - ★ :object
 - ★ :pair
 - ★ :pair-fn
 - ★ :promise
 - ★ :reader-cond
 - ★ :record
 - ★ :set
 - ★ :spec
 - ★ :style
 - ★ :style-map
 - ★ :tab
 - ★ :vector
 - ★ :vector-fn

Testing and Development

Information on testing and development can be found [here](#).

Note: Changed the default branch to [main](#).

Contributors

A number of folks have contributed to `zprint`, not all of whom show up on GitHub because I have integrated the code or suggestions manually. **Thanks for all of the great contributions!**

- Exposing `sci.core` in babashka: @borkdude
- Tests running in babashka: @borkdude
- Additional colors and color-map entries: @RingMan
- Updated `rewrite-cljs` dependency to 0.4.5 @rundis/
- Readme updates: @mathiasn, @Quezion, @vemv, @arichiardi, @bhurlow, @kommen.
- `--url` and `--url-only`: @coltnz
- Use UTF-8 locale to build the native image: @mynomoto
- Suggestion/encouragement to implement `:respect-bl`: @griffis
- Thread safety suggestions: @fazzone
- `:option-fn` and `:fn-format` for enhanced vector formatting: @milankinen
- Fixed missing require in `spec.cljc`: @Quezion
- Corrected readme: @griffis
- Fixed nested reader conditional: @rgould1
- Clarified and added useful example for clj usage: @bherrmann7
- Sublime text plugin instructions: @ekinnear
- Use body indentation for the `ns` macro: @pesterhazy
- Suggested fix for international chars and graalVM native image: @huahaiy

Thanks to everyone who has contributed fixes as well as everyone who has reported an issue. I really appreciate all of the help making `zprint` better for everybody!

Acknowledgements

At the core of `zprint` is the `rewrite-clj` library originally created by Yannick Scherer, ported to Clojurescript by Magnus Rundberget, and recently merged into a single, supported, documented, and updated library by Lee Read. This is a great library! I would not have attempted `zprint` if `rewrite-clj` didn't exist to build upon.

Additionally, allowing options maps containing functions to be read from files safely is made possible by `sci`, the Small Clojure Interpreter by Michael Borkent (@borkdude). This is a very well designed and implemented addition to Clojure that required almost no effort to integrate into `zprint`.

License

Copyright © 2016-2023 Kim Kinnear

Distributed under the MIT License. See the file LICENSE for details.