# Contributing to the Rust Cookbook

The cookbook needs contributors and is intended to be easy to

contribute to. Help is welcome.

\* [Building and testing](#building-and-testing)

\* [Finding what to contribute](#finding-what-to-contribute)

\* [Adding an example](#adding-an-example)

\* [Example guidelines](#example-guidelines)

## Building and testing

To start, clone the cookbook from git and navigate to that directory:

```

git clone https://github.com/rust-lang-nursery/rust-cookbook.git

cd rust-cookbook

```

Cookbook is built with [mdBook], so install that first with Cargo:

```

cargo install --version 0.3.5 mdbook

```

To build and view the cookbook locally, run:

```

mdbook serve

```

Then open `http://localhost:3000` in a web browser to browse the cookbook. Any

changes you make to the cookbook source will be automatically rebuilt and

visible in the browser, so it can be helpful to keep this window open while

editing.

All examples in the cookbook are tested with [skeptic], a tool for

testing arbitrary markdown documentation in a style similar to

rustdoc.

To run the cookbook test suite:

```

cargo test

```

## Linters

The Rust Cookbook comes with link checking and spell checking linters that

run on the continuous integration server. These linters should be run locally

before submitting a pull request to ensure there are no dead links or spelling

errors made.

To install the link checker, review the documentation for [python] to install

python 3.6 and pip3. Installing link-checker once the dependencies are met

is done with pip3.

```

[sudo] pip3 install link-checker==0.1.0

```

Alternatively, add the user install directory (probably `~/.local/bin`) to

your PATH variable and install link-checker for your user.

```

pip3 install --user link-checker==0.1.0

```

Checking the links of the book locally first requires the book to be built

with mdBook. From the root directory of the cookbook, the following commands

run the link checker.

```

mdbook build

link-checker ./book

```

The aspell binary provides spell checking. Apt packages provide installation

on Debian based operating systems.

```

[sudo] apt install aspell -y

```

on Mac:

```

brew install aspell

```

On other Linux distributions you might also need to install the `aspell-en`

package, or similar.

To check the spelling of the Rust Cookbook locally, run the following command

from the root of the Cookbook.

```

./ci/spellcheck.sh

# or, if you're using a different locale

LANG=en\_US.UTF-8 ./ci/spellcheck.sh

```

If the spell checker finds a misspelled word, you have the opportunity to

correct the spelling mistake with the number keys. If the spelling mistake

is erroneous, add the word to the dictionary located in `ci/dictionary.txt`.

Pressing `a` or `l` will not add the word to the custom dictionary.

If there are no errors, it will just print the local Aspell version and exit.

[mdbook]: https://github.com/rust-lang-nursery/mdBook

[python]: https://packaging.python.org/tutorials/installing-packages/#install-pip-setuptools-and-wheel

[skeptic]: https://github.com/brson/rust-skeptic

## Finding what to contribute

This project is intended to be simple to contribute to, and to always

have obvious next work items available. If at any time there is not

something obvious to contribute, that is a bug. Feel free to ask for

additional support at the

[Rust Ecosystem Working Group](https://gitter.im/rust-lang/WG-ecosystem).

The development process for the cookbook is presently oriented around

crates: we decide which crates to represent in the cookbook, then come

up with example use cases to write, then write the examples. And those

are the three basic, recurring types of contributions needed.

The development process for the cookbook today is tied to the [libz

blitz], a broader project to improve the Rust crate ecosystem, and the

cookbook presently represents the crates under consideration there.

The easiest way to find the most immediate work needed for the

cookbook is to follow the "What's next" section at the top of that

thread, which should at all times link to something to contribute to

the cookbook.

Otherwise, look for GitHub issues with the [example] tag. The simplest

way to contribute is to claim one of these examples, and submit a PR

adding it. If you do claim one, please leave a comment saying so, so

others don't accidentally duplicate your work.

If you have an idea for an example for a specific crate, please

suggest it on the relevant [tracking issue].

Please do not submit examples for crates not yet represented in the

cookbook, unless it is part of the libz blitz crate schedule.

Contribution will be open to a broader set of crates in the future.

For more about which crates are represented in the cookbook, see ["a

note about crate representation"][which-crates] in the cookbook.

[example]: https://github.com/rust-lang-nursery/rust-cookbook/issues?q=is%3Aissue+is%3Aopen+label%3Aexample

[tracking issue]: https://github.com/rust-lang-nursery/rust-cookbook/issues?q=is%3Aissue+is%3Aopen+label%3A%22tracking+issue%22

[which-crates]: https://rust-lang-nursery.github.io/rust-cookbook/about.html#a-note-about-crate-representation

[libz blitz]: https://internals.rust-lang.org/t/rust-libz-blitz/5184

## Adding an example

Adding an example involves:

- Deciding which \_section\_ of the book it belongs in

- Deciding which \_categories\_ apply to it

- Adding the example to the section index in intro.md

- Adding the example to the appropriate section markdown file

- Updating badges and hyperlinks as needed

- Writing a useful description of the example

The finished commit will look something like [this one].

[this one]: https://github.com/rust-lang-nursery/rust-cookbook/commit/e698443f2af08d3106d953c68c1977eba3c3526c

Examples are presently organized in three ways:

- Book sections - the cookbook is a book, and is organized like a book

in logical sections, like "basics", "encoding", "concurrency".

- Category tags - each example is tagged with one or more category

tags, like "filesystem", "debugging".

- Crate tags - each example is tagged with one or more crate tags,

indicating which crates are represented in the example. Those that

use no additional crates are simply tagged 'std'.

For more about the organization of the book see ["how to read this

book"] in the cookbook.

Hopefully your example belongs to an obvious section and categories,

but since the cookbook is so new, quite possibly not. Ask on thread.

For most steps you can simply follow the lead of existing examples.

The art comes in writing effective examples.

["how to read this book"]: https://rust-lang-nursery.github.io/rust-cookbook/about.html#how-to-read-this-book

## Example guidelines

Examples in the cookbook have these goals and qualities:

- They can be described by a single sentence that states their utility.

- They can be read and understood by a complete beginner.

- They are standalone examples that can be copied into a learner's

own workspace and compiled and modified for experimentation.

- They demonstrate real tasks, such that experienced developers

may use it as a reference.

- They follow best practices and do not take shortcuts.

- They use consistent error handling.

#### Title

Examples should have a simple single-sentence title that describes

something a typical Rust user typically wants to do.

> ## Generate random numbers with given distribution

#### Description

Describe traits imported and the methods used. Think about what information

supports the use case and might not be obvious to someone new. Keep the

description to 1-4 sentences, avoiding explanations outside the scope of the

code sample.

Use third person narrative of the code execution, taking the opportunity

to link to API documentation. Always use

[active voice](https://www.plainlanguage.gov/guidelines/conversational/use-active-voice/).

Hyperlink all references to APIs, either

on doc.rust-lang.org/std or docs.rs, and style them as `code`. Use

wildcard version specifiers for crate links.

Any requirements to execute the code that are not apparent, such as

passing environment flags, or configuring `Cargo.toml` should be added

after the code sample.

> By default, random numbers are generated with [uniform distribution].

> To generate numbers with other distributions you instantiate a

> distribution, then sample from that distribution using

> [`Distribution::sample`] with help of a random-number

> generator [`rand::Rng`].

>

> The [distributions available are documented here][rand-distributions].

> An example using the [`Normal`] distribution is shown below.

[uniform distribution]: https://en.wikipedia.org/wiki/Uniform\_distribution\_(continuous)

[`Distribution::sample`]: https://docs.rs/rand/\*/rand/distributions/trait.Distribution.html#tymethod.sample

[`rand::Rng`]: https://docs.rs/rand/\*/rand/trait.Rng.html

[rand-distributions]: https://docs.rs/rand/\*/rand/distributions/index.html

[`Normal`]: https://docs.rs/rand/\*/rand/distributions/struct.Normal.html

#### Code

Examples are intended to be read by complete beginners, and copied into

projects for experimentation. They should follow best practices and

not take shortcuts.

The example should have minimal code that doesn't directly support the

description of the example. Keep extra functions and types to a

minimum.

When an example must handle the possibility of errors, follow the error handling

templates in ["A note about error handling"][errors]. Examples always set up

error handling correctly and propagate errors with `?` (not `try!`, `urwrap`, or

`expect`). If there is no need for error handling in the example, prefer `main()`.

Avoid glob imports (`\*`), even for preludes, so that users can see what

traits are called. (Some crates might consider using glob imports for preludes

best practice, making this awkward.)

Examples should be simple and obvious enough that an experienced dev

do not need comments.

Examples should compile without warnings, clippy lint warnings, or panics.

The code should be formatted by rustfmt. Hide all error boilerplate and

parts of the sample that do not accomplish the subject of the example.

Mark examples that depend on external systems with `no\_run` or remove them

if they are not required for the example. Avoid inline comments, preferring

explanation in the description.

> ```rust

> extern crate rand;

>

> use rand::distributions::{Normal, Distribution};

>

> fn main() {

> let mut rng = rand::thread\_rng();

> let normal = Normal::new(2.0, 3.0);

> let v = normal.sample(&mut rng);

> println!("{} is from a N(2, 9) distribution", v)

> }

> ```

Finally, this book is intended to demonstrate the integration

of crates that work well together. We are on the lookout for examples

that feature multiple crates sensibly.

[errors]: https://rust-lang-nursery.github.io/rust-cookbook/about.html#a-note-about-error-handling