What is rustdoc?

The standard Rust distribution ships with a tool called `rustdoc`. Its job is to generate documentation for Rust projects. On a fundamental level, Rustdoc takes as an argument either a crate root or a Markdown file, and produces HTML, CSS, and JavaScript.

Basic usage

Let's give it a try! Create a new project with Cargo:

```
$ cargo new docs --lib
$ cd docs
```

In `src/lib.rs`, Cargo has generated some sample code. Delete it and replace it with this:

```
/// foo is a function
fn foo() {}
```

Let's run `rustdoc` on our code. To do so, we can call it with the path to our crate root like this:

```
$ rustdoc src/lib.rs
```

This will create a new directory, `doc`, with a website inside! In our case, the main page is located in `doc/lib/index.html`. If you open that up in a web browser, you will see a page with a search bar, and "Crate lib" at the top, with no contents.

Configuring rustdoc

There are two problems with this: first, why does it think that our package is named "lib"? Second, why does it not have any contents?

The first problem is due to `rustdoc` trying to be helpful; like `rustc`, it assumes that our crate's name is the name of the file for the crate root. To fix this, we can pass in a command-line flag:

```
$ rustdoc src/lib.rs --crate-name docs
```

Now, `doc/docs/index.html` will be generated, and the page says "Crate docs."

For the second issue, it is because our function `foo` is not public; `rustdoc` defaults to generating documentation for only public functions. If we change our code...

```
/// foo is a function
pub fn foo() {}
```

... and then re-run `rustdoc`:

```
$ rustdoc src/lib.rs --crate-name docs
```

We now have some generated documentation. Open up `doc/docs/index.html` and check it out! It should show a link to the `foo` function's page, which is located at `doc/docs/fn.foo.html`. On that page, you'll see the "foo is a function" we put inside the documentation comment in our crate.

Using rustdoc with Cargo

Cargo also has integration with `rustdoc` to make it easier to generate docs. Instead of the `rustdoc` command, we could have done this:

```
$ cargo doc
```

Internally, this calls out to `rustdoc` like this:

```
$ rustdoc --crate-name docs src/lib.rs -o <path>/docs/target/doc -L
dependency=<path>/docs/target/debug/deps
```

You can see this with `cargo doc --verbose`.

It generates the correct `--crate-name` for us, as well as pointing to `src/lib.rs`. But what about those other arguments? - `-o` controls the *o*utput of our docs. Instead of a top-level `doc` directory, notice that Cargo puts generated documentation under `target`. That is the idiomatic place for generated files in Cargo projects. - `-L` flag helps rustdoc find the dependencies your code relies on. If our project used dependencies, we would get documentation for them as well!

Outer and inner documentation

The `///` syntax is used to document the item present after it. That's why it is called an outer documentation. There is another syntax: `//!`, which is used to document the item it is present inside. It is called an inner documentation. It is often used when documenting the entire crate, because nothing comes before it: it is the root of the crate. So in order to document an entire crate, you need to use `//!` syntax. For example:

```
//! This is my first rust crate
```

When used in the crate root, it documents the item it is inside, which is the crate itself.

For more information about the `//!` syntax, see the Book.

Using standalone Markdown files

`rustdoc` can also generate HTML from standalone Markdown files. Let' s give it a try: create a `README.md` file with these contents:

```
# Docs

This is a project to test out `rustdoc`.
```

````
[Here is a link!](https://www.rust-lang.org)

## Example

```rust
fn foo() -> i32 {
    1 + 1
}
```
````

And call `rustdoc` on it:

```
$ rustdoc README.md
```

You will find an HTML file in `docs/doc/README.html` generated from its Markdown contents.

Cargo currently does not understand standalone Markdown files, unfortunately.

Summary

This covers the simplest use-cases of `rustdoc`. The rest of this book will explain all of the options that `rustdoc` has, and how to use them.