Contributing to Subplot

2021-10-21 11:32

Contents

We would love for you to contribute to the Subplot project, in any way or form that is good for you. Here are some suggestions to make it as easy as possible for us to incorporate your contribution. However, you contributing is more important than you following the guidelines below.

Reporting issues, or raising topics for discussion: Please use the GitLab issue tracker. If there's an existing issue for what you want to talk about, please use that, but if not, it's preferable to open a new issue, as it's easier for the Subplot maintainers to merge issues than split them. Add tags if you feel they're relevant, but it's OK to not use tags as well. Issue tags exist primarily to help the maintainers keep a handle on long lists of issues, and the primary responsibility is on the maintainers to add the tags that help them.

Asking for help: Please use the GitLab issue tracker for this as well.

Updating documentation, tests, or code: Please submit a merge request via GitLab, or open an issue with a link to a git repository and branch we can pull from.

Running the test suite: For any and all changes that are meant to change the git repository, the full Subplot test suite must pass. To run it, use the following command:

```
$ ./check -v
```

If you only touch documentation, and don't want to install all the build dependencies, you can skip running the tests yourself.

Code formatting: Rust code must be formatted as if by the `rustfmt` tool. The test suite checks that.

Rust version: We aim for Subplot to be buildable on the stable version of

Debian and the Rust version packaged therein. At this time, that's Rust version 1.34.

Python version: We require Python code to work on Python 3.7 and later, the version in the current stable version of Debian.

Portability: We want Subplot and the test programs it generates to be portable, but haven't yet defined a list of environments we explicitly target, apart from Debian 10 (buster) on 64-bit Intel architecture systems ("amd64" in Debian terminology). We try to avoid breaking compatibility with other environments (other versions of Debian, other Linux distributions, other Unix variants, other operating systems) and architecture, but do not currently have the ability to test that. (Help would be welcome.)

Keep commits small: It's easier for us to review smaller commits, less than about 200 lines of diff, except when each hunk is similar. For example, if you rename a function, a commit that does that and nothing else can be quite large, but is easy to review.

Keep commits integral: Each commit should be a cohesive change, not a mix of unrelated changes. If a commit renames a function, it shouldn't also add a new argument to the function or make other changes unrelated to the rename.

Keep commits in a merge request related: If a commit adds a new feature, it shouldn't also fix a bug in an old feature. If there's a need to re-do the new feature, the bug fix will take longer to get merged.

Tell a story with a merge request: The commits in a merge request should be arranged in a way that make the change as a whole easy to follow: they "tell a story" of how you would make the set of changes the second time, knowing everything you know after doing them the first time.

The goal is not to show the path you originally took, but show how to get there in the best possible way. You should tell the story of flying by plane to get somewhere, not how you explored the world and eventually invented flying machines to travel faster.

1 Definition of done

When a change is made to Subplot, it must meet the following minimum requirements to be considered "done": finished and not requiring further work or attention.

- if review of the change (in an MR or otherwise) has raised any issues, they have been resolved or filed as issues
- the change has been merged to the main branch
- ./check passes after the merge
- all relevant issues have been updated appropriately or closed

2 Debugging subplot

Ideally as a **user** of subplot you shouldn't need this, all error messages should give you sufficient context to chase down your problems. However if you are debugging a change to subplot itself then you may need to know how to do the following:

1. You can change the logging level of subplot by means of the `SUBPLOT_LOG` environment variable. If you set it to `trace` then it will show you absolutely everything traced inside the program. You can learn about the syntax for this variable on the EnvFilter[1] docs on `docs.rs`.
2. You can redirect the logging to a file by means of the `SUBPLOT_LOG_FILE` environment variable. Simply set it to the path you want to store logging to. Subplot will *append* to that file, so it's safe to have it in your environment for multiple runs.
3. To change the logging format, you can set `SUBPLOT_LOG_FORMAT` to one of: `pretty` to have a multiline pretty format, `default` or `oneline` for a basic one-line-per-event log format and `json` for a one-line-per-event JSON log format.

[1] `https://docs.rs/tracing-subscriber/0.2.20/tracing_subscriber/filter/struct.EnvFilter.html`