This document outlines processes regarding management of rustfmt.

Stabilising an Option

In this Section, we describe how to stabilise an option of the rustfmt's configration.

Conditions

- Is the default value correct ?
- The design and implementation of the option are sound and clean.
- The option is well tested, both in unit tests and, optimally, in real usage.
- There is no open bug about the option that prevents its use.

Steps

Open a pull request that closes the tracking issue. The tracking issue is listed beside the option in `Configurations.md`.

- Update the `Config` enum marking the option as stable.
- Update the the `Configuration.md` file marking the option as stable.
- Update `CHANGELOG.md` marking the option as stable.

After the stabilisation

The option should remain backward-compatible with previous parameters of the option. For instance, if the option is an enum `enum Foo { Alice, Bob }` and the variant `Foo::Bob` is removed/renamed, existing use of the `Foo::Bob` variant should map to the new logic. Breaking changes can be applied under the condition they are version-gated.

Make a Release

0. Update CHANGELOG.md

1. Update Cargo.toml and Cargo.lock

For example, 1.0.0 -> 1.0.1:

```
-version = "1.0.0"
+version = "1.0.1"
```

2. Push the commit to the master branch

E.g., https://github.com/rust-lang/rustfmt/commit/5274b49caa1a7db6ac10c76bf1a3d5710ccef569

3. Create a release tag

```
git tag -s v1.2.3 -m "Release 1.2.3"
```

4. Publish to crates.io

`cargo publish`

5. Create a PR to rust-lang/rust to update the rustfmt submodule

Note that if you are updating `rustc-ap-*` crates, then you need to update **every** submodules in the rust-lang/rust repository that depend on the crates to use the same version of those.

As of 2019/05, there are two such crates: `rls` and `racer` (`racer` depends on `rustc-ap-syntax` and `rls` depends on `racer`, and `rls` is one of submodules of the rust-lang/rust repository).