# Changelog Update

If you want to help with updating the changelog, you're in the right place.

## When to update

Typos and other small fixes/additions are *always* welcome.

Special care needs to be taken when it comes to updating the changelog for a new Rust release. For that purpose, the changelog is ideally updated during the week before an upcoming stable release. You can find the release dates on the Rust Forge.

Most of the time we only need to update the changelog for minor Rust releases. It's been very rare that Clippy changes were included in a patch release.

## Changelog update walkthrough

### 1. Finding the relevant Clippy commits

Each Rust release ships with its own version of Clippy. The Clippy subtree can be found in the `tools` directory of the Rust repository.

Depending on the current time and what exactly you want to update, the following bullet points might be helpful:

- When writing the release notes for the **upcoming stable release** you need to check out the Clippy commit of the current Rust `beta` branch. Link
- When writing the release notes for the **upcoming beta release**, you need to check out the Clippy commit of the current Rust `master`. Link
- When writing the (forgotten) release notes for a **past stable release**, you need to check out the Rust release tag of the stable release. Link

Usually you want to write the changelog of the **upcoming stable release**. Make sure though, that `beta` was already branched in the Rust repository.

To find the commit hash, issue the following command when in a `rust-lang/rust` checkout:

```
git log --oneline -- src/tools/clippy/ | grep -o "Merge commit '[a-f0-9]*' into .*" | head -
```

### 2. Fetching the PRs between those commits

Once you've got the correct commit range, run

```
util/fetch_prs_between.sh commit1 commit2 > changes.txt
```

and open that file in your editor of choice.

When updating the changelog it's also a good idea to make sure that `commit1` is already correct in the current changelog.

### 3. Authoring the final changelog

The above script should have dumped all the relevant PRs to the file you specified. It should have filtered out most of the irrelevant PRs already, but it's a good idea to do a manual cleanup pass where you look for more irrelevant PRs. If you're not sure about some PRs, just leave them in for the review and ask for feedback.

With the PRs filtered, you can start to take each PR and move the `changelog:` content to `CHANGELOG.md`. Adapt the wording as you see fit but try to keep it somewhat coherent.

The order should roughly be:

1. New lints
2. Moves or deprecations of lints
3. Changes that expand what code existing lints cover
4. False positive fixes
5. Suggestion fixes/improvements
6. ICE fixes
7. Documentation improvements
8. Others

As section headers, we use:

```
### New Lints
### Moves and Deprecations
### Enhancements
### False Positive Fixes
### Suggestion Fixes/Improvements
### ICE Fixes
### Documentation Improvements
### Others
```

Please also be sure to update the Beta/Unreleased sections at the top with the relevant commit ranges.