# Basics for hacking on Clippy

This document explains the basics for hacking on Clippy. Besides others, this includes how to build and test Clippy. For a more in depth description on the codebase take a look at Adding Lints or Common Tools.

## Get the Code

First, make sure you have checked out the latest version of Clippy. If this is your first time working on Clippy, create a fork of the repository and clone it afterwards with the following command:

```
git clone git@github.com:<your-username>/rust-clippy
```

If you've already cloned Clippy in the past, update it to the latest version:

```
# If the upstream remote has not been added yet
git remote add upstream https://github.com/rust-lang/rust-clippy
# upstream has to be the remote of the rust-lang/rust-clippy repo
git fetch upstream
# make sure that you are on the master branch
git checkout master
# rebase your master branch on the upstream master
git rebase upstream/master
# push to the master branch of your fork
git push
```

## Building and Testing

You can build and test Clippy like every other Rust project:

```
cargo build  # builds Clippy
cargo test   # tests Clippy
```

Since Clippy's test suite is pretty big, there are some commands that only run a subset of Clippy's tests:

```
# only run UI tests
cargo uitest
# only run UI tests starting with `test_`
TESTNAME="test_" cargo uitest
# only run dogfood tests
cargo test --test dogfood
```

If the output of a UI test differs from the expected output, you can update the reference file with:

```
cargo dev bless
```

For example, this is necessary, if you fix a typo in an error message of a lint or if you modify a test file to add a test case.

*Note:* This command may update more files than you intended. In that case only commit the files you wanted to update.

## `cargo dev`

Clippy has some dev tools to make working on Clippy more convenient. These tools can be accessed through the `cargo dev` command. Available tools are listed below. To get more information about these commands, just call them with `--help`.

```
# formats the whole Clippy codebase and all tests
cargo dev fmt
# register or update lint names/groups/...
cargo dev update_lints
# create a new lint and register it
cargo dev new_lint
# automatically formatting all code before each commit
cargo dev setup git-hook
# (experimental) Setup Clippy to work with IntelliJ-Rust
cargo dev setup intellij
```

More about intellij command usage and reasons here

## lintcheck

`cargo lintcheck` will build and run clippy on a fixed set of crates and generate a log of the results.
You can `git diff` the updated log against its previous version and see what impact your lint made on a small set of crates.
If you add a new lint, please audit the resulting warnings and make sure there are no false positives and that the suggestions are valid.

Refer to the tools README for more details.

## PR

We follow a rustc no merge-commit policy. See https://rustc-dev-guide.rust-lang.org/contributing.html#opening-a-pr.

## Common Abbreviations

| Abbreviation | Meaning |
|---|---|
| UB | Undefined Behavior |

| | |
|---|---|
| FP | False Positive |
| FN | False Negative |
| ICE | Internal Compiler Error |
| AST | Abstract Syntax Tree |
| MIR | Mid-Level Intermediate Representation |
| HIR | High-Level Intermediate Representation |
| TCX | Type context |

This is a concise list of abbreviations that can come up during Clippy development. An extensive general list can be found in the rustc-dev-guide glossary. Always feel free to ask if an abbreviation or meaning is unclear to you.

## Install from source

If you are hacking on Clippy and want to install it from source, do the following:

First, take note of the toolchain override in `/rust-toolchain`. We will use this override to install Clippy into the right toolchain.

> *Tip: You can view the active toolchain for the current directory with `rustup show active-toolchain`.*

From the Clippy project root, run the following command to build the Clippy binaries and copy them into the toolchain directory. This will override the currently installed Clippy component.

```
cargo build --release --bin cargo-clippy --bin clippy-driver -Zunstable-options --
out-dir "$(rustc --print=sysroot)/bin"
```

Now you may run `cargo clippy` in any project, using the toolchain where you just installed Clippy.

```
cd my-project
cargo +nightly-2021-07-01 clippy
```

...or `clippy-driver`

```
clippy-driver +nightly-2021-07-01 <filename>
```

If you need to restore the default Clippy installation, run the following (from the Clippy project root).

```
rustup component remove clippy
rustup component add clippy
```

> ***DO NOT** install using `cargo install --path . --force` since this will overwrite rustup proxies. That is, `~/.cargo/bin/cargo-clippy` and `~/.cargo/bin/clippy-driver` should be hard or soft links to `~/.cargo/bin/rustup`. You can repair these by running `rustup update`.*