Clippy

A collection of lints to catch common mistakes and improve your Rust code.

There are over 500 lints included in this crate!

Lints are divided into categories, each with a default lint level. You can choose how much Clippy is supposed to ~~annoy~~ help you by changing the lint level by category.

Category	Description	Default level
`clippy::all`	all lints that are on by default (correctness, suspicious, style, complexity, perf)	**warn/deny**
`clippy::correctness`	code that is outright wrong or useless	**deny**
`clippy::suspicious`	code that is most likely wrong or useless	**warn**
`clippy::style`	code that should be written in a more idiomatic way	**warn**
`clippy::complexity`	code that does something simple but in a complex way	**warn**
`clippy::perf`	code that can be written to run faster	**warn**
`clippy::pedantic`	lints which are rather strict or have occasional false positives	allow
`clippy::nursery`	new lints that are still under development	allow
`clippy::cargo`	lints for the cargo manifest	allow

More to come, please file an issue if you have ideas!

The lint list also contains "restriction lints", which are for things which are usually not considered "bad", but may be useful to turn on in specific cases. These should be used very selectively, if at all.

Table of contents:

Usage

Below are instructions on how to use Clippy as a cargo subcommand, in projects that do not use cargo, or in Travis CI.

As a cargo subcommand (`cargo clippy`)

One way to use Clippy is by installing Clippy through rustup as a cargo subcommand.

Step 1: Install Rustup

You can install Rustup on supported platforms. This will help us install Clippy and its dependencies.

If you already have Rustup installed, update to ensure you have the latest Rustup and compiler:

```
rustup update
```

Step 2: Install Clippy

Once you have rustup and the latest stable release (at least Rust 1.29) installed, run the following command:

```
rustup component add clippy
```

If it says that it can't find the `clippy` component, please run `rustup self update`.

Step 3: Run Clippy

Now you can run Clippy by invoking the following command:

```
cargo clippy
```

Automatically applying Clippy suggestions

Clippy can automatically apply some lint suggestions, just like the compiler.

```
cargo clippy --fix
```

Workspaces

All the usual workspace options should work with Clippy. For example the following command will run Clippy on the `example` crate:

```
cargo clippy -p example
```

As with `cargo check`, this includes dependencies that are members of the workspace, like path dependencies. If you want to run Clippy **only** on the given crate, use the `--no-deps` option like this:

```
cargo clippy -p example -- --no-deps
```

Using `clippy-driver`

Clippy can also be used in projects that do not use cargo. To do so, run `clippy-driver` with the same arguments you use for `rustc`. For example:

```
clippy-driver --edition 2018 -Cpanic=abort foo.rs
```

Note that `clippy-driver` is designed for running Clippy only and should not be used as a general replacement for `rustc`. `clippy-driver` may produce artifacts that are not optimized as expected, for example.

Travis CI

You can add Clippy to Travis CI in the same way you use it locally:

```
language: rust
rust:
  - stable
  - beta
before_script:
  - rustup component add clippy
script:
  - cargo clippy
  # if you want the build job to fail when encountering warnings, use
  - cargo clippy -- -D warnings
  # in order to also check tests and non-default crate features, use
  - cargo clippy --all-targets --all-features -- -D warnings
  - cargo test
  # etc.
```

Note that adding `-D warnings` will cause your build to fail if **any** warnings are found in your code. That includes warnings found by rustc (e.g. `dead_code`, etc.). If you want to avoid this and only cause an error for Clippy warnings, use `#![deny(clippy::all)]` in your code or `-D clippy::all` on the command line. (You can swap `clippy::all` with the specific lint category you are targeting.)

Configuration

Some lints can be configured in a TOML file named `clippy.toml` or `.clippy.toml`. It contains a basic `variable = value` mapping e.g.

```
avoid-breaking-exported-api = false
blacklisted-names = ["toto", "tata", "titi"]
cognitive-complexity-threshold = 30
```

See the list of lints for more information about which lints can be configured and the meaning of the variables.

Note that configuration changes will not apply for code that has already been compiled and cached under `./target/`; for example, adding a new string to `doc-valid-idents` may still result in Clippy flagging that string. To be sure that any configuration changes are applied, you may want to run `cargo clean` and re-compile your crate from scratch.

To deactivate the "for further information visit *lint-link*" message you can define the `CLIPPY_DISABLE_DOCS_LINKS` environment variable.

Allowing/denying lints

You can add options to your code to `allow` / `warn` / `deny` Clippy lints:

- the whole set of `Warn` lints using the `clippy` lint group (`#![deny(clippy::all)]`). Note that `rustc` has additional lint groups.
- all lints using both the `clippy` and `clippy::pedantic` lint groups (`#![deny(clippy::all)]`, `#![deny(clippy::pedantic)]`). Note that `clippy::pedantic` contains some very aggressive lints

prone to false positives.

- only some lints (`#![deny(clippy::single_match, clippy::box_vec)]` , etc.)
- `allow` / `warn` / `deny` can be limited to a single function or module using `#[allow(...)]` , etc.

Note: `allow` means to suppress the lint for your code. With `warn` the lint will only emit a warning, while with `deny` the lint will emit an error, when triggering for your code. An error causes clippy to exit with an error code, so is useful in scripts like CI/CD.

If you do not want to include your lint levels in your code, you can globally enable/disable lints by passing extra flags to Clippy during the run:

To allow `lint_name` , run

```
cargo clippy -- -A clippy::lint_name
```

And to warn on `lint_name` , run

```
cargo clippy -- -W clippy::lint_name
```

This also works with lint groups. For example, you can run Clippy with warnings for all lints enabled:

```
cargo clippy -- -W clippy::pedantic
```

If you care only about a single lint, you can allow all others and then explicitly warn on the lint(s) you are interested in:

```
cargo clippy -- -A clippy::all -W clippy::useless_format -W clippy::...
```

Specifying the minimum supported Rust version

Projects that intend to support old versions of Rust can disable lints pertaining to newer features by specifying the minimum supported Rust version (MSRV) in the clippy configuration file.

```
msrv = "1.30.0"
```

The MSRV can also be specified as an inner attribute, like below.

```
#![feature(custom_inner_attributes)]
#![clippy::msrv = "1.30.0"]

fn main() {
  ...
}
```

You can also omit the patch version when specifying the MSRV, so `msrv = 1.30` is equivalent to `msrv = 1.30.0` .

Note: `custom_inner_attributes` is an unstable feature, so it has to be enabled explicitly.

Lints that recognize this configuration option can be found here

Contributing

If you want to contribute to Clippy, you can find more information in CONTRIBUTING.md.

License