Lint levels

In `rustc`, lints are divided into five *levels*:

1. allow
2. warn
3. force-warn
4. deny
5. forbid

Each lint has a default level (explained in the lint listing later in this chapter), and the compiler has a default warning level. First, let's explain what these levels mean, and then we'll talk about configuration.

allow

These lints exist, but by default, do nothing. For example, consider this source:

```
pub fn foo() {}
```

Compiling this file produces no warnings:

```
$ rustc lib.rs --crate-type=lib
$
```

But this code violates the `missing_docs` lint.

These lints exist mostly to be manually turned on via configuration, as we'll talk about later in this section.

warn

The 'warn' lint level will produce a warning if you violate the lint. For example, this code runs afoul of the `unused_variables` lint:

```
pub fn foo() {
    let x = 5;
}
```

This will produce this warning:

```
$ rustc lib.rs --crate-type=lib
warning: unused variable: `x`
 --> lib.rs:2:9
  |
2 |     let x = 5;
  |         ^
  |
  = note: `#[warn(unused_variables)]` on by default
  = note: to avoid this warning, consider using `_x` instead
```

force-warn

'force-warn' is a special lint level. It's the same as 'warn' in that a lint at this level will produce a warning, but unlike the 'warn' level, the 'force-warn' level cannot be overridden. If a lint is set to 'force-warn', it is guaranteed to warn: no more, no less. This is true even if the overall lint level is capped via cap-lints.

deny

A 'deny' lint produces an error if you violate it. For example, this code runs into the `exceeding_bitshifts` lint.

```
fn main() {
    100u8 << 10;
}
```

```
$ rustc main.rs
error: bitshift exceeds the type's number of bits
 --> main.rs:2:13
  |
2 |     100u8 << 10;
  |     ^^^^^^^^^^^
  |
  = note: `#[deny(exceeding_bitshifts)]` on by default
```

What's the difference between an error from a lint and a regular old error? Lints are configurable via levels, so in a similar way to 'allow' lints, warnings that are 'deny' by default let you allow them. Similarly, you may wish to set up a lint that is `warn` by default to produce an error instead. This lint level gives you that.

forbid

'forbid' is a special lint level that fills the same role for 'deny' that 'force-warn' does for 'warn'. It's the same as 'deny' in that a lint at this level will produce an error, but unlike the 'deny' level, the 'forbid' level can not be overridden to be anything lower than an error. However, lint levels may still be capped with `--cap-lints` (see below) so `rustc --cap- lints warn` will make lints set to 'forbid' just warn.

Configuring warning levels

Remember our `missing_docs` example from the 'allow' lint level?

```
$ cat lib.rs
pub fn foo() {}
$ rustc lib.rs --crate-type=lib
$
```

We can configure this lint to operate at a higher level, both with compiler flags, as well as with an attribute in the source code.

You can also "cap" lints so that the compiler can choose to ignore certain lint levels. We'll talk about that last.

Via compiler flag

The `-A`, `-W`, `--force-warn` `-D`, and `-F` flags let you turn one or more lints into allowed, warning, force-warn, deny, or forbid levels, like this:

```
$ rustc lib.rs --crate-type=lib -W missing-docs
warning: missing documentation for crate
 --> lib.rs:1:1
  |
1 | pub fn foo() {}
  | ^^^^^^^^^^^^
  |
  = note: requested on the command line with `-W missing-docs`

warning: missing documentation for a function
 --> lib.rs:1:1
  |
1 | pub fn foo() {}
  | ^^^^^^^^^^^^

$ rustc lib.rs --crate-type=lib -D missing-docs
error: missing documentation for crate
 --> lib.rs:1:1
  |
1 | pub fn foo() {}
  | ^^^^^^^^^^^^
  |
  = note: requested on the command line with `-D missing-docs`

error: missing documentation for a function
 --> lib.rs:1:1
  |
1 | pub fn foo() {}
  | ^^^^^^^^^^^^

error: aborting due to 2 previous errors
```

You can also pass each flag more than once for changing multiple lints:

```
$ rustc lib.rs --crate-type=lib -D missing-docs -D unused-variables
```

And of course, you can mix these five flags together:

```
$ rustc lib.rs --crate-type=lib -D missing-docs -A unused-variables
```

The order of these command line arguments is taken into account. The following allows the `unused-variables` lint, because it is the last argument for that lint:

```
$ rustc lib.rs --crate-type=lib -D unused-variables -A unused-variables
```

You can make use of this behavior by overriding the level of one specific lint out of a group of lints. The following example denies all the lints in the `unused` group, but explicitly allows the `unused-variables` lint in that group (forbid still trumps everything regardless of ordering):

```
$ rustc lib.rs --crate-type=lib -D unused -A unused-variables
```

Since `force-warn` and `forbid` cannot be overridden, setting one of them will prevent any later level for the same lint from taking effect.

Via an attribute

You can also modify the lint level with a crate-wide attribute:

```
$ cat lib.rs
#![warn(missing_docs)]

pub fn foo() {}
$ rustc lib.rs --crate-type=lib
warning: missing documentation for crate
 --> lib.rs:1:1
  |
1 | / #![warn(missing_docs)]
2 | |
3 | | pub fn foo() {}
  | |_______________^
  |
note: lint level defined here
 --> lib.rs:1:9
  |
1 | #![warn(missing_docs)]
  |         ^^^^^^^^^^^^

warning: missing documentation for a function
 --> lib.rs:3:1
  |
3 | pub fn foo() {}
  | ^^^^^^^^^^^^
```

`warn`, `allow`, `deny`, and `forbid` all work this way. There is no way to set a lint to `force-warn` using an attribute.

You can also pass in multiple lints per attribute:

```
#![warn(missing_docs, unused_variables)]

pub fn foo() {}
```

And use multiple attributes together:

```
#![warn(missing_docs)]
#![deny(unused_variables)]

pub fn foo() {}
```

Capping lints

`rustc` supports a flag, `--cap-lints LEVEL` that sets the "lint cap level." This is the maximum level for all lints. So for example, if we take our code sample from the "deny" lint level above:

```
fn main() {
    100u8 << 10;
}
```

And we compile it, capping lints to warn:

```
$ rustc lib.rs --cap-lints warn
warning: bitshift exceeds the type's number of bits
 --> lib.rs:2:5
  |
2 |     100u8 << 10;
  |     ^^^^^^^^^^^
  |
  = note: `#[warn(exceeding_bitshifts)]` on by default

warning: this expression will panic at run-time
 --> lib.rs:2:5
  |
2 |     100u8 << 10;
  |     ^^^^^^^^^^^ attempt to shift left with overflow
```

It now only warns, rather than errors. We can go further and allow all lints:

```
$ rustc lib.rs --cap-lints allow
$
```

This feature is used heavily by Cargo; it will pass `--cap-lints allow` when compiling your dependencies, so that if they have any warnings, they do not pollute the output of your build.