Tests

`rustc` has a built-in facility for building and running tests for a crate. More information about writing and running tests may be found in the Testing Chapter of the Rust Programming Language book.

Tests are written as free functions with the `#[test]` attribute. For example:

```rust
#[test]
fn it_works() {
    assert_eq!(2 + 2, 4);
}
```

Tests "pass" if they return without an error. They "fail" if they panic, or return a `Result` with an error.

By passing the `--test` option to `rustc`, the compiler will build the crate in a special mode to construct an executable that will run the tests in the crate. The `--test` flag will make the following changes:

- The crate will be built as a `bin` crate type, forcing it to be an executable.
- Links the executable with `libtest`, the test harness that is part of the standard library, which handles running the tests.
- Synthesizes a `main` function which will process command-line arguments and run the tests. This new `main` function will replace any existing `main` function as the entry point of the executable, though the existing `main` will still be compiled.
- Enables the `test` cfg option, which allows your code to use conditional compilation to detect if it is being built as a test.
- Enables building of functions annotated with the `test` and `bench` attributes, which will be run by the test harness.

After the executable is created, you can run it to execute the tests and receive a report on what passes and fails. If you are using Cargo to manage your project, it has a built-in `cargo test` command which handles all of this automatically. An example of the output looks like this:

```
running 4 tests
test it_works ... ok
test check_valid_args ... ok
test invalid_characters ... ok
test walks_the_dog ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished
in 0.00s
```

> ***Note**: Tests must be built with the `unwind` panic strategy. This is because all tests run in the same process, and they are intended to catch panics, which is not possible with the `abort` strategy. See the unstable `-Z panic-abort-tests` option for experimental support of the `abort` strategy by spawning tests in separate processes.*

Test attributes

Tests are indicated using attributes on free functions. The following attributes are used for testing, see the linked documentation for more details:

- `#[test]` — Indicates a function is a test to be run.
- `#[bench]` — Indicates a function is a benchmark to be run. Benchmarks are currently unstable and only available in the nightly channel, see the unstable docs for more details.
- `#[should_panic]` — Indicates that the test function will only pass if the function panics.
- `#[ignore]` — Indicates that the test function will be compiled, but not run by default. See the `--ignored` and `--include-ignored` options to run these tests.

CLI arguments

The libtest harness has several command-line arguments to control its behavior.

> *Note: When running with `cargo test`, the libtest CLI arguments must be passed after the `--` argument to differentiate between flags for Cargo and those for the harness. For example: `cargo test -- --nocapture`*

Filters

Positional arguments (those without a `-` prefix) are treated as filters which will only run tests whose name matches one of those strings. The filter will match any substring found in the full path of the test function. For example, if the test function `it_works` is located in the module `utils::paths::tests`, then any of the filters `works`, `path`, `utils::`, or `utils::paths::tests::it_works` will match that test.

See Selection options for more options to control which tests are run.

Action options

The following options perform different actions other than running tests.

`--list`

Prints a list of all tests and benchmarks. Does not run any of the tests. Filters can be used to list only matching tests.

`-h`, `--help`

Displays usage information and command-line options.

Selection options

The following options change how tests are selected.

`--test`

This is the default mode where all tests will be run as well as running all benchmarks with only a single iteration (to ensure the benchmark works, without taking the time to actually perform benchmarking). This can be combined with the `--bench` flag to run both tests and perform full benchmarking.

`--bench`

This runs in a mode where tests are ignored, and only runs benchmarks. This can be combined with `--test` to run both benchmarks and tests.

`--exact`

This forces filters to match the full path of the test exactly. For example, if the test `it_works` is in the module `utils::paths::tests`, then only the string `utils::paths::tests::it_works` will match that test.

`--skip` ***FILTER***

Skips any tests whose name contains the given *FILTER* string. This flag may be passed multiple times.

`--ignored`

Runs only tests that are marked with the `ignore` attribute.

`--include-ignored`

Runs both ignored and non-ignored tests.

`--exclude-should-panic`

Excludes tests marked with the `should_panic` attribute.

⚠️ 🚧 This option is unstable, and requires the `-Z unstable-options` flag. See tracking issue #82348 for more information.

Execution options

The following options affect how tests are executed.

`--test-threads` ***NUM_THREADS***

Sets the number of threads to use for running tests in parallel. By default, uses the amount of concurrency available on the hardware as indicated by `available_parallelism`.

This can also be specified with the `RUST_TEST_THREADS` environment variable.

`--force-run-in-process`

Forces the tests to run in a single process when using the `abort` panic strategy.

⚠️ 🚧 This only works with the unstable `-Z panic-abort-tests` option, and requires the `-Z unstable-options` flag. See tracking issue #67650 for more information.

`--ensure-time`

⚠️ 🚧 This option is unstable, and requires the `-Z unstable-options` flag. See tracking issue #64888 and the unstable docs for more information.

`--shuffle`

Runs the tests in random order, as opposed to the default alphabetical order.

This may also be specified by setting the `RUST_TEST_SHUFFLE` environment variable to anything but `0`.

The random number generator seed that is output can be passed to `--shuffle-seed` to run the tests in the same order again.

Note that `--shuffle` does not affect whether the tests are run in parallel. To run the tests in random order sequentially, use `--shuffle --test-threads 1`.

⚠️ 🚧 This option is unstable, and requires the `-Z unstable-options` flag. See tracking issue #89583 for more information.

`--shuffle-seed` ***SEED***

Like `--shuffle`, but seeds the random number generator with *SEED*. Thus, calling the test harness with `--shuffle-seed` *SEED* twice runs the tests in the same order both times.

SEED is any 64-bit unsigned integer, for example, one produced by `--shuffle`.

This can also be specified with the `RUST_TEST_SHUFFLE_SEED` environment variable.

⚠️ 🚧 This option is unstable, and requires the `-Z unstable-options` flag. See tracking issue #89583 for more information.

Output options

The following options affect the output behavior.

`-q`, `--quiet`

Displays one character per test instead of one line per test. This is an alias for `--format=terse`.

`--nocapture`

Does not capture the stdout and stderr of the test, and allows tests to print to the console. Usually the output is captured, and only displayed if the test fails.

This may also be specified by setting the `RUST_TEST_NOCAPTURE` environment variable to anything but `0`.

`--show-output`

Displays the stdout and stderr of successful tests after all tests have run.

Contrast this with `--nocapture` which allows tests to print *while they are running*, which can cause interleaved output if there are multiple tests running in parallel, `--show-output` ensures the output is contiguous, but requires waiting for all tests to finish.

`--color` ***COLOR***

Control when colored terminal output is used. Valid options:

- `auto`: Colorize if stdout is a tty and `--nocapture` is not used. This is the default.
- `always`: Always colorize the output.
- `never`: Never colorize the output.

`--format` ***FORMAT***

Controls the format of the output. Valid options:

- `pretty`: This is the default format, with one line per test.
- `terse`: Displays only a single character per test. `--quiet` is an alias for this option.
- `json`: Emits JSON objects, one per line. ⚠️ 🚧 This option is unstable, and requires the `-Z unstable-options` flag. See tracking issue #49359 for more information.

`--logfile` ***PATH***

Writes the results of the tests to the given file.

`--report-time`

⚠️ 🚧 This option is unstable, and requires the `-Z unstable-options` flag. See tracking issue #64888 and the unstable docs for more information.

Unstable options

Some CLI options are added in an "unstable" state, where they are intended for experimentation and testing to determine if the option works correctly, has the right design, and is useful. The option may not work correctly, break, or change at any time. To signal that you acknowledge that you are using an unstable option, they require passing the `-Z unstable-options` command-line flag.

Benchmarks

The libtest harness supports running benchmarks for functions annotated with the `#[bench]` attribute. Benchmarks are currently unstable, and only available on the nightly channel. More information may be found in the unstable book.

Custom test frameworks

Experimental support for using custom test harnesses is available on the nightly channel. See tracking issue #50297 and the custom_test_frameworks documentation for more information.