Unit Testing

Unit tests allow you to test the contract methods individually. They are suitable to check the storage is updated correctly, and that methods return their expected values. They are written in the contract's language and execute locally.

If you used one of ourexamples as template, then you simply need to navigate to the contract's folder, and useyarn test . In case you didn't, then we recommend you copy the necessary node files (e.g.package.json) from one of our templates.

tip You can runyarn test from the root folder of each project to run both unit andintegration tests.

Snippet I: Testing a Counter

The tests in theCounter Example rely on basic functions to check that theincrement ,decrement , andreset methods work properly.

- Rust

contract-rs/src/lib.rs loading ... See full example on GitHub

Snippet II: Modifying the Context

While doing unit testing you can modify theEnvironment variables through theVMContextBuilder . This will enable you to, for example, simulate calls from different users, with specific attached deposit and GAS. Here we present a snippet on how we test thedonate method from ourDonation Example by manipulating thepredecessor andattached_deposit .

- Rust

contract-rs/src/lib.rs loading ... See full example on GitHub

⚠ Limitations

Unit tests are useful to check for code integrity, and detect basic errors on isolated methods. However, since unit tests do not run on a blockchain, there are many things which they cannot detect. Unit tests are not suitable for:

- Testinggas
- andstorage
- usage
- Testingtransfers
- Testingcross-contract calls
- Testing complex interactions, i.e. multiple users depositing money on the contract

For all these cases it is necessary tocomplement unit tests withintegration tests . Edit this page Last updatedonFeb 8, 2024 bymatiasbenary Was this page helpful? Yes No

Previous Introduction Next Integration Tests