# Develop using Fluidity Truffle Box

TLDR; Fluidity has a pretty awesome truffle-box with lots of goodies to build secure, well-tested smart contracts. With two simple commands, you'll be set up for development using these testing and security tools preconfigured.

To use:

npm install -g truffle truffle unbox fluidity/fluidity-truffle-box

## Develop using Fluidity Truffle Box¶

Today, I'd like to share Fluidity's Truffle Box. It contains a combination of tools we use to develop, test, and analyze our smart contracts. As our team has created several products and proof-of-concepts, we've developed a set of standards for creating smart contract repos and having a truffle-box makes this easier. We've tried and tested different tools and this box contains our favorite tools for efficient development.

The Fluidity Truffle Box allows us to bootstrap our smart contract projects. This box focuses primarily on building safe and secure smart contracts that can be easily audited. We're able to consolidate many of the new tools, as well as ensure that any projects we start always use up to date versions. Prior to starting new projects, we double check that the packages and tools within truffle-box are up to date. We acknowledge this truffle-box is not a kitchen sink of all the tools. It has been curated based upon what we consistently use. The box doesn't have a UI component and mostly contains configurations and tools that aid in testing.

## Core Tools¶

At Fluidity we write smart contracts with Solidity using the latest fixed version, currently 0.5.10, and we use Ganache as our primary test chain. We use Open-Zeppelin as the base for many of our contracts' systems. In addition, there's a linter for both the smart contracts and tests usingsolhint andstandard . A linter analyzes code and tests and then either recommends or updates the code for potential syntax errors and style recommendations. Solhint's configuration is present in fluidity-truffle-box and tries to closely align with the Solidity documentation. We use standard because it's opinionated and keeps the codebase consistent with minimal fuss. Thus, standard doesn't have any custom configurations.

## How we handle testing and code coverage¶

### Gnosis Mock Contract¶

Gnosis-Mock is a comprehensive mocking smart contract that allows us to perform unit-testing of our smart contracts. For those not familiar, mocking means creating dummy objects to simulate the behavior of real objects. Mocking allows one to isolate the behavior of the testable object from any other dependencies. As an example, certain behavior in smart contract designs, such as closing a token sale, may require dependencies on several state variables and contracts to already be triggered to allow it to burn tokens. To close a token sale, let's say you need to have started the sale, sold some threshold of tokens, reached a certain future block, and given close sale privileges to another user. Any one of these requirements, outside of just calling close sale, would need to be run if mocking were not used. By simulating these states with dummy smart contract, it narrows down exactly what needs to be tested. We heavily use the Gnosis mock library which allows us to test contracts independently before running any integration tests.

### Truffle Assertions¶

Truffle assertions has streamlined how we test events and states of smart contracts. The packages works inside Truffle tests and provides more assertions to call about smart contract states. It supports revert reason strings so we can be confident that exact cases are hit on revert. In addition, it provides more granular ways to test for function failures.

### Eth-gas-reporter¶

Eth gas reporter shows us the gas usage per unit tests in a nice report after running your test suite. We've tried many other tools before as well as trying to write custom gas usage with web3. This has been a recent find for us and has made gas analysis much easier.

### Solidity-coverage¶

We use solidity-coverage to determine what percentage of the codebase is covered by unit tests. We use a forked version byleapdao because this version supports many of the breaking changes with Solidity 0.5.0+. Code coverages give a helpful metric to ensure we're testing as we code and that all branch paths are tested. It has also served for us to remove redundant code paths that we've found were not accessible. In addition, we have incorporated code-coverage with our CI process, meaning it is automatically executed on every pull request we make, allowing us to easily check PRs for code coverage. The configuration is stored in/.solcover.js , and can be personalised for your project.

### Slither and Mythril¶

Both slither and mythril are python packages that we've included in our truffle-box. They are security tools byTrail of Bits that allow us to better self-audit our code and evaluate many of our invariants and assumptions internally. Instructions to run each of these packages are stored in the README.md file of the fluidity-truffle-box.

Slither is a static code analyzer by Trail of Bits. It runs across the contracts directory fairly quickly (< 1 min) and picks up potentially vulnerable Solidity code. Mythril is a security analysis tool for EVM bytecode from ConsenSys. The program takes much longer to run, and while it's in the truffle-box, we'll usually run this tool on AWS or another cloud environment and analyze those results. While there is some overlap of errors that are picked up by truffle-compile, slither, and mythril, each focuses on different types of error. We therefore find it helpful to use all 3 on our contracts.

### Circle CI Template¶

At Fluidity we use circle-ci for our continuous integration. By having sample templates in the repo, it's more convenient to making sure we enable it whenever we create new repositories. The new repos therefore constantly have to build successfully and migrate via Ganache, and test coverage is apparent to all members on the team. With the template in place, it's quick for us to add additional commands such as linting or gas reporting dependent on where we are in the development process.

## Unique features¶

### Using allowUnlimitedContractSize¶

We allow our initial contract designs not to be limited by any contract size constraints with the use of the flagallowUnlimitedContractSize . Ganache has a default gas limit of 6,721,975 which does not reflect the actual gas limit of public testnet and mainnets, being 7,000,000 and 8,000,000 respectively. While our team does want to reduce the gas footprint while developing and iterating through different token designs/debugging, it's been helpful not to be limited by a strict gas price or gas limit with testing. For example, we use longer descriptive messages in revert reasons when developing, but reduce them to be more succinct for final code versions. Reducing the string lengths saved us 3,000,000 gas.

Run each line in a different terminal window

yarn ganache-unlimited yarn truffle truffle-unlimited After settling on a design, we'll start iterating to reduce gas usage. As mentioned above, we use eth-gas-reporter to do this. Thus, with 100% test coverage and integration tests, eth-gas-reporter gives a pretty comprehensive understanding of gas usage throughout.

## Custom Migration and Test Utilities¶

There are two custom utilities files that we have created and keep available for the right circumstance of contracts. First, we have a custom migration utilities script. Truffle has deployed contract management but only stores a single address for each contract deploy per network id; each deployed contract is treated as a singleton. However, there have been cases where we've deployed multiple instances of the same contract and have needed to save each of the deployed contract addresses. Our solution was to create a custom migration_utils.js that stored the contract addresses per migration in a JSON file. From there, integration tests and accessory scripts use the contract addresses while still relying on the build directory for ABIs.

The second script is related to testing time sensitive smart contracts. We've written a post about it calledStanding the Time of Test and we've used this script and features in Ganache to run both idempotent tests as well as run simulations.

Running the command for a specific date

yarn ganache-unlimited --time '2019-02-15T15:53:00+00:00'

## Anything else¶

Most of the features are in theREADME in the directory and I'd love to take any questions or issues using this repository. Linking all these tools allows for a solid, sane development process for our team. They're always at our fingertips with fluidity-truffle-box. Hopefully, you'll be able to use the Fluidity Truffle Box to enhance your own smart contract development process!

## Github repos¶

- Fluidity Truffle Box:https://github.com/fluidity/fluidity-truffle-box
- Gnosis-Mock:https://github.com/gnosis/mock-contract
- Truffle Assertions:https://github.com/rkalis/truffle-assertions
- Eth gas reporter:https://github.com/cgewecke/eth-gas-reporter

- Leapdao Solidity-Coverage:https://github.com/leapdao/solidity-coverage
- Original Solidity-Coverage:https://github.com/sc-forks/solidity-coverage
- Slither:https://github.com/crytic/slither
- Mythril:https://github.com/ConsenSys/mythril

## Fluidity Resources:¶

- Website:https://fluidity.io
- Twitter:https://twitter.com/fluidityio
- Facebook:https://facebook.com/fluidityio/
- LinkedIn:https://linkedin.com/company/fluidityio/
- YouTube:https://youtube.com/c/fluidityio
- Blog:https://medium.com/fluidity