**Writing tests for your Solidity smart contract**

**Testing**

Testing is a crucial step in your smart contract development journey, as the lack of tests can be a roadblock in the deployment stage or during a smart contract audit.

So, buckle up as we unveil what separates the best developers from the rest: comprehensive, effective tests!

Inside the test folder create a file called FundMeTest.t.sol. .t. is a naming convention of Foundry, please use it.

The writing of a test contract shares the initial steps with the writing of a normal smart contract. We state the SPDX-License-Identifier, solidity version and a contract name:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

contract FundMeTest {

}

**Now the fun part!**

To be able to run tests using Foundry we need to import the set of smart contracts Foundry comes with that contains a lot of prebuilt functions that make our lives 10x easier.

import {Test} from "forge-std/Test.sol";

We also make sure our test contract inherits what we just imported.

contract FundMeTest is Test{

The next logical step in our testing process is deploying the FundMe contract. In the future, we will learn how to import our deployment scripts, but for now, let's do the deployments right in our test file.

We do this inside the setUp function. This function is always the first to execute whenever we run our tests. Here we will perform all the prerequisite actions that are required before doing the actual testing, things like:

* Deployments;
* User addresses;
* Balances;
* Approvals;
* And various other operations depending on what's required to initiate the tested contracts.

But before that, please create another function, called testDemo.

Your test contract should look like this:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

import {Test} from "forge-std/Test.sol";

contract FundMeTest is Test {

function setUp() external { }

function testDemo() public { }

}

Now run forge test in your terminal. This command has a lot of options, you can find more about those [here](https://book.getfoundry.sh/reference/cli/forge/test?highlight=forge%20test#forge-test).

Our (empty) test passed! Great!

Ok, but how does it work? What's the order of things?

Please update the contract to the following:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

import {Test} from "forge-std/Test.sol";

contract FundMeTest is Test {

uint256 favNumber = 0;

bool greatCourse = false;

function setUp() external {

favNumber = 1337;

greatCourse = true;

}

function testDemo() public {

assertEq(favNumber, 1337);

assertEq(greatCourse, true);

}

}

Call forge test again.

As you can see our test passed. What do we learn from this?

1. We declare some state variables.
2. Next up setUp() is called.
3. After that forge runs all the test functions.

Another nice way of testing this and also an important tool for debugging is console.log. The console library comes packed in the Test.sol that we imported, we just need to update the things we import to this:

import {Test, console} from "forge-std/Test.sol";

Let's insert some console.log calls inside our contract:

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.18;

import {Test, console} from "forge-std/Test.sol";

contract FundMeTest is Test {

uint256 favNumber = 0;

bool greatCourse = false;

function setUp() external {

favNumber = 1337;

greatCourse = true;

console.log("This will get printed first!");

}

function testDemo() public {

assertEq(favNumber, 1337);

assertEq(greatCourse, true);

console.log("This will get printed second!");

console.log("Updraft is changing lives!");

console.log("You can print multiple things, for example this is a uint256, followed by a bool:", favNumber, greatCourse);

}

}

forge test has an option called verbosity. By controlling this option we decide how verbose should the output of the forge test be. The default forge test has a verbosity of 1. Here are the verbosity levels, choose according to your needs:

Verbosity levels:

- 2: Print logs for all tests

- 3: Print execution traces for failing tests

- 4: Print execution traces for all tests, and setup traces for failing tests

- 5: Print execution and setup traces for all tests

Given that we want to see the printed logs, we will call forge test -vv (the number of v's indicates the level).

Ran 1 test for test/FundMe.t.sol:FundMeTest

[PASS] testDemo() (gas: 9482)

Logs:

This will get printed first!

This will get printed second!

Updraft is changing lives!

You can print multiple things, for example this is a uint256, followed by a bool: 1337 true

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 422.20µs (63.30µs CPU time)

You can read more about console.log [here](https://book.getfoundry.sh/reference/forge-std/console-log?highlight=console.#console-logging).

Let's delete the logs for now, but keep the console import. We could use it later for debugging.

Let's deploy the FundMe contract.

For that, we will first need to import it into our test file, then declare it as a state variable and deploy it in the setUp function.

**Testing FundMe**

Delete testDemo. Make a new function called testMinimumDollarIsFive. As the name states, we will test if the MINIMUM\_USD is equal to 5e18.

function testMinimumDollarIsFive() public {

assertEq(fundMe.MINIMUM\_USD(), 5e18);

}

Run it with forge test.

[⠰] Compiling...

[⠆] Compiling 1 files with 0.8.25

[⠰] Solc 0.8.25 finished in 827.51ms

Compiler run successful!

Ran 1 test for test/FundMe.t.sol:FundMeTest

[PASS] testMinimumDollarIsFive() (gas: 5453)

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 487.20µs (43.20µs CPU time)

Great job! Let's delete the favNumber and greatCourse to keep our test file nice and clean.

Try to change the right side of the assertEq line to check what a failed test looks like.

Awesome! You just completed your first Forge test, get used to writing those because whether you are going to be a developer or an auditor you will write ***A LOT*** of them.