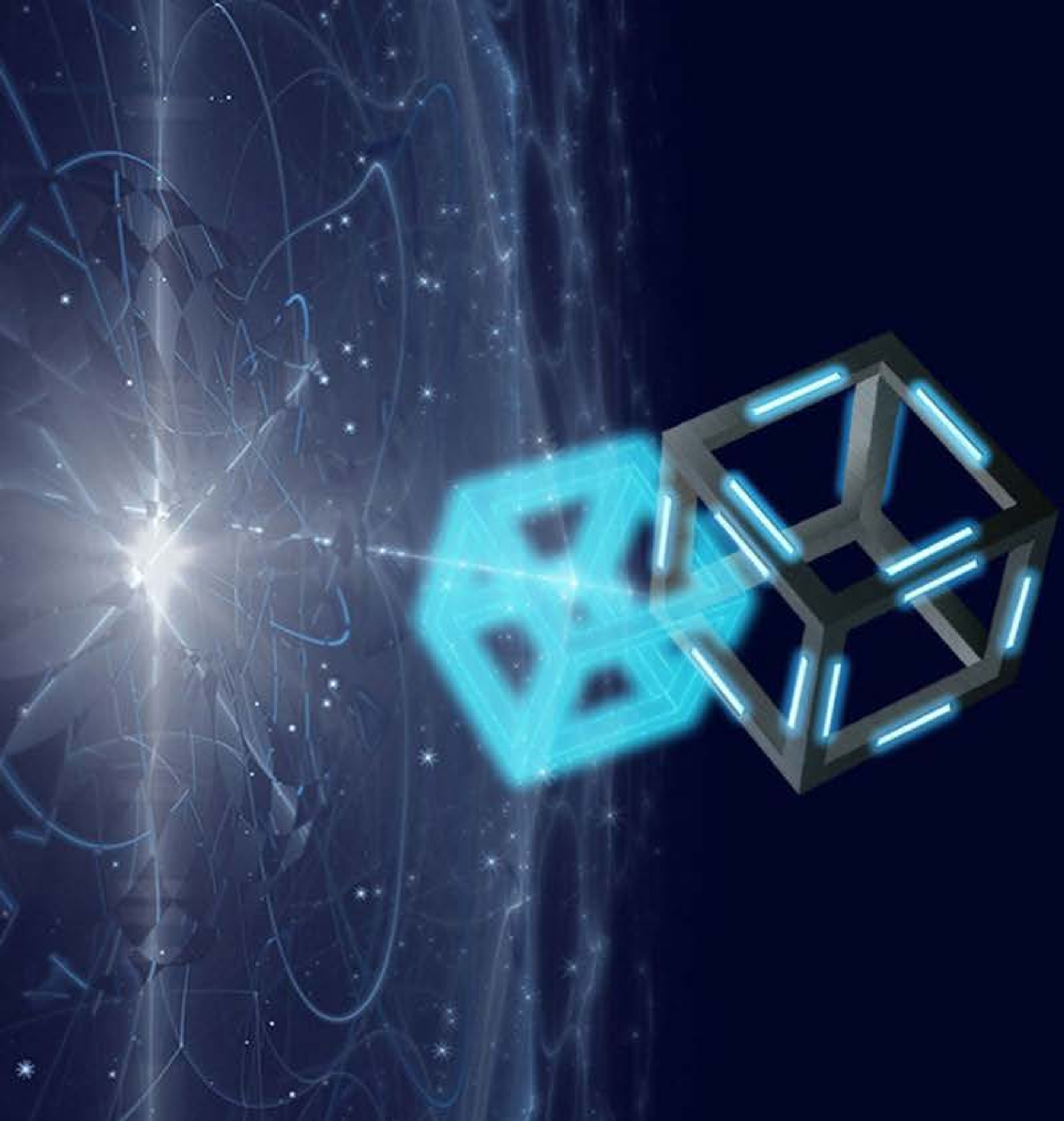
# ETHEREUM SMART CONTRACT DEVELOPMENT

Testing Smart Contracts
With Foundry



## Installing Foundry

## Installing Foundry

**Install Git (required for Foundry):**

- Linux Fedora: $ sudo dnf install git-all
- Linux Ubuntu: $ sudo apt install git-all

- MacOS: $ git --version => will prompt you to install it

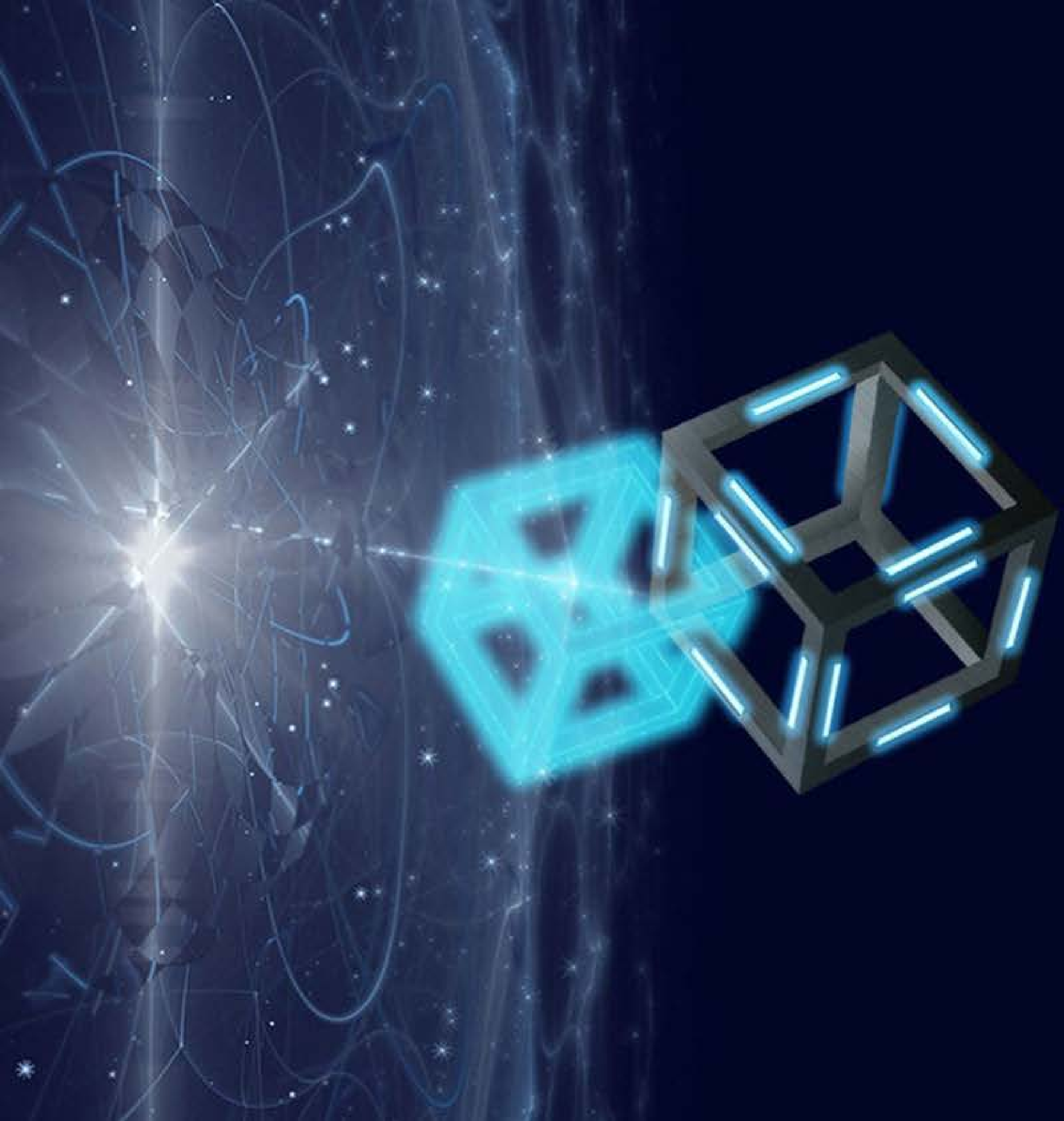
- Windows: https://git-scm.com/download/win

**Install Foundry using Foundryup (Foundry toolchain installer)**

- For Windows, install GitBash or WSL - PowerShell and CMD are not supported!

- In terminal / GitBash: ***curl -L https://foundry.paradigm.xyz | bash => installs foundryup***

- In terminal / GitBash: ***foundryup*** => update to the latest versions of foundry, forge, cast, anvil and chisel



## Basic Foundry Commands

## Basic Foundry Commands 1/2

- Foundry Book – official docs: [https://book.getfoundry.sh](https://book.getfoundry.sh)
- Create a new project: ***forge init projectName***

  Automatically installs the Forge Standard Library => preferred testing library for Foundry projects
- Build a project: ***forge build***
- **Run test(s):**
  - ***forge test***
  - ***forge test --match-contract ContractName --match-test testName --gas-report -vvvv***
- Code coverage: ***forge coverage***

## Basic Foundry Commands 2/2

- Install all dependencies for an existing project (git clone...): ***forge install***

- Install specific dependencies:

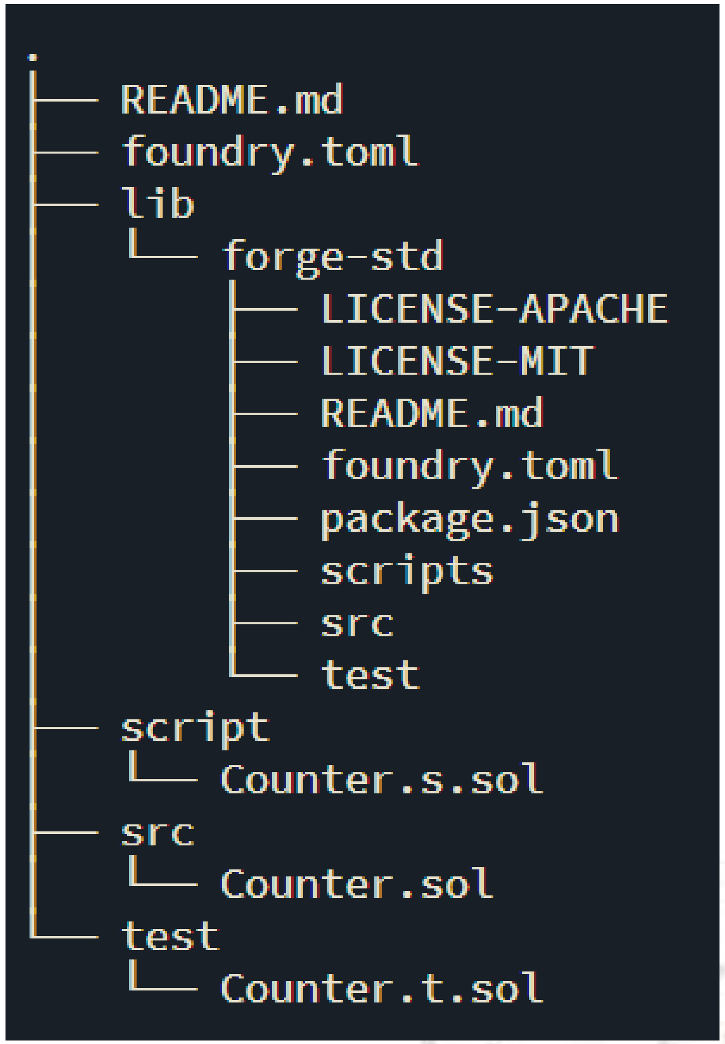
  ***forge install OpenZeppelin/openzeppelin-contracts***
  ***forge install transmissions11/solmate@v7***

- Update a dependancy to latest commit: ***forge update lib/solmate***

- Remove a dependency: ***forge remove lib/solmate***

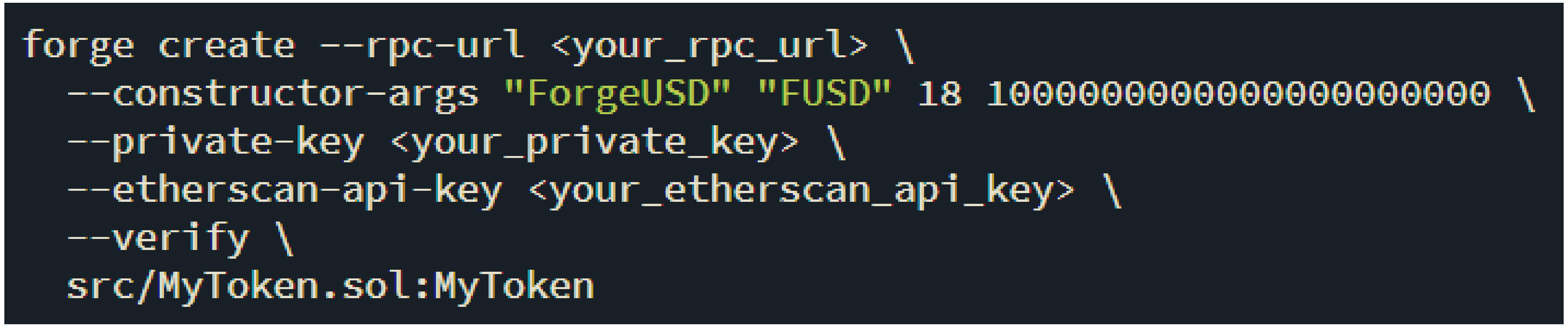
## Project Layout

- Config file: foundry. toml
- The default directory for contracts is src
- The default directory for tests is test => any contract with a function that starts with test is considered to be a test
- Dependencies are stored in lib



```
.
├── README.md
├── foundry.toml
├── lib
│   └── forge-std
│       ├── LICENSE-APACHE
│       ├── LICENSE-MIT
│       ├── README.md
│       ├── foundry.toml
│       ├── package.json
│       ├── scripts
│       ├── src
│       └── test
├── script
│   └── Counter.s.sol
├── src
│   └── Counter.sol
└── test
    └── Counter.t.sol
```

## Deploying & Verifying Contracts



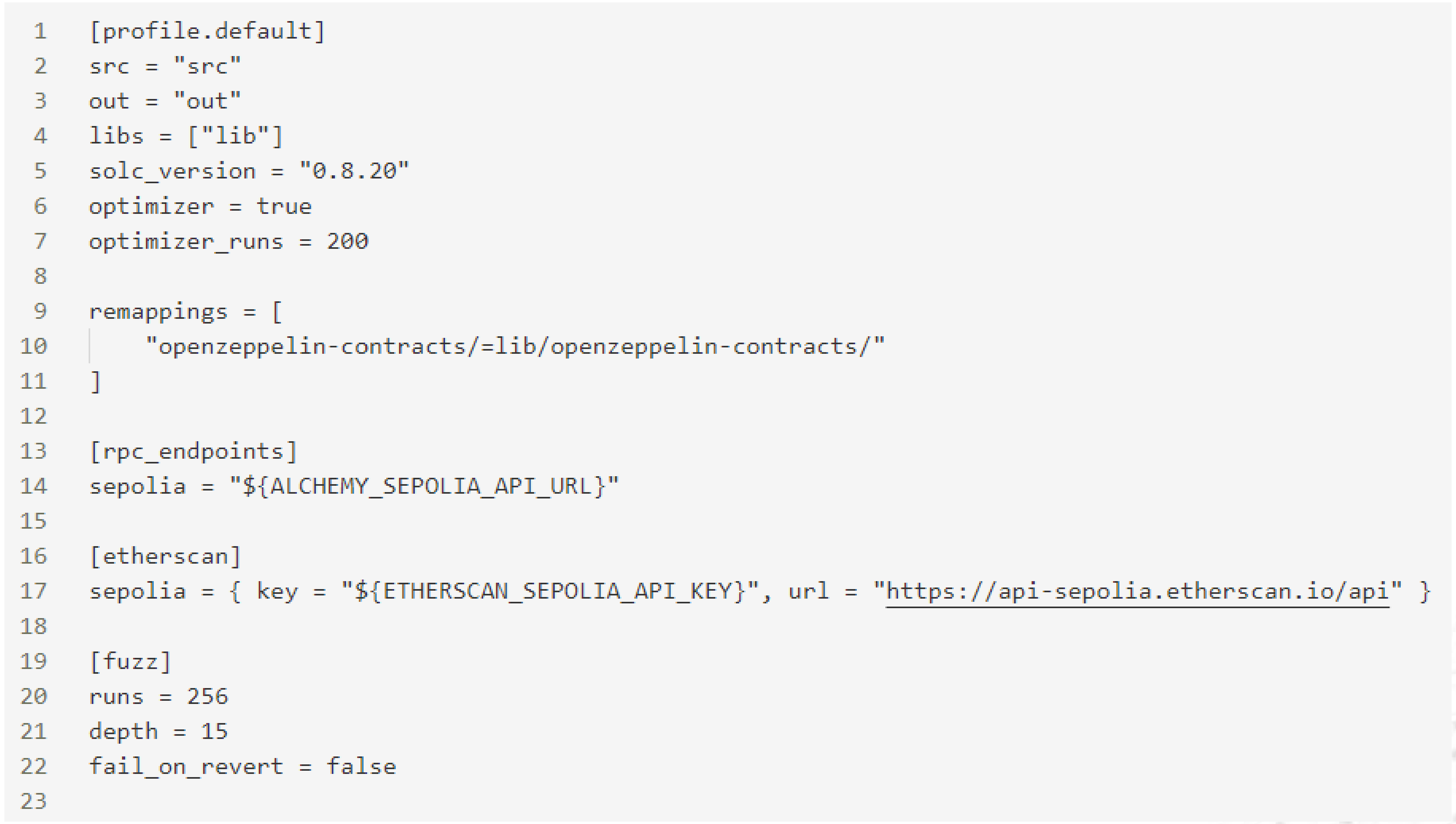
```
forge create --rpc-url <your_rpc_url> \
  --constructor-args "ForgeUSD" "FUSD" 18 1000000000000000000000 \
  --private-key <your_private_key> \
  --etherscan-api-key <your_etherscan_api_key> \
  --verify \
  src/MyToken.sol:MyToken
```

**Example:**

forge create --rpc-url sepolia --private-key $PK_SEPOLIA --constructor-args "My NFT" "MNFT" "baseUri" --etherscan-api-key sepolia --verify src/2_StandardUnitTests.sol:NFT

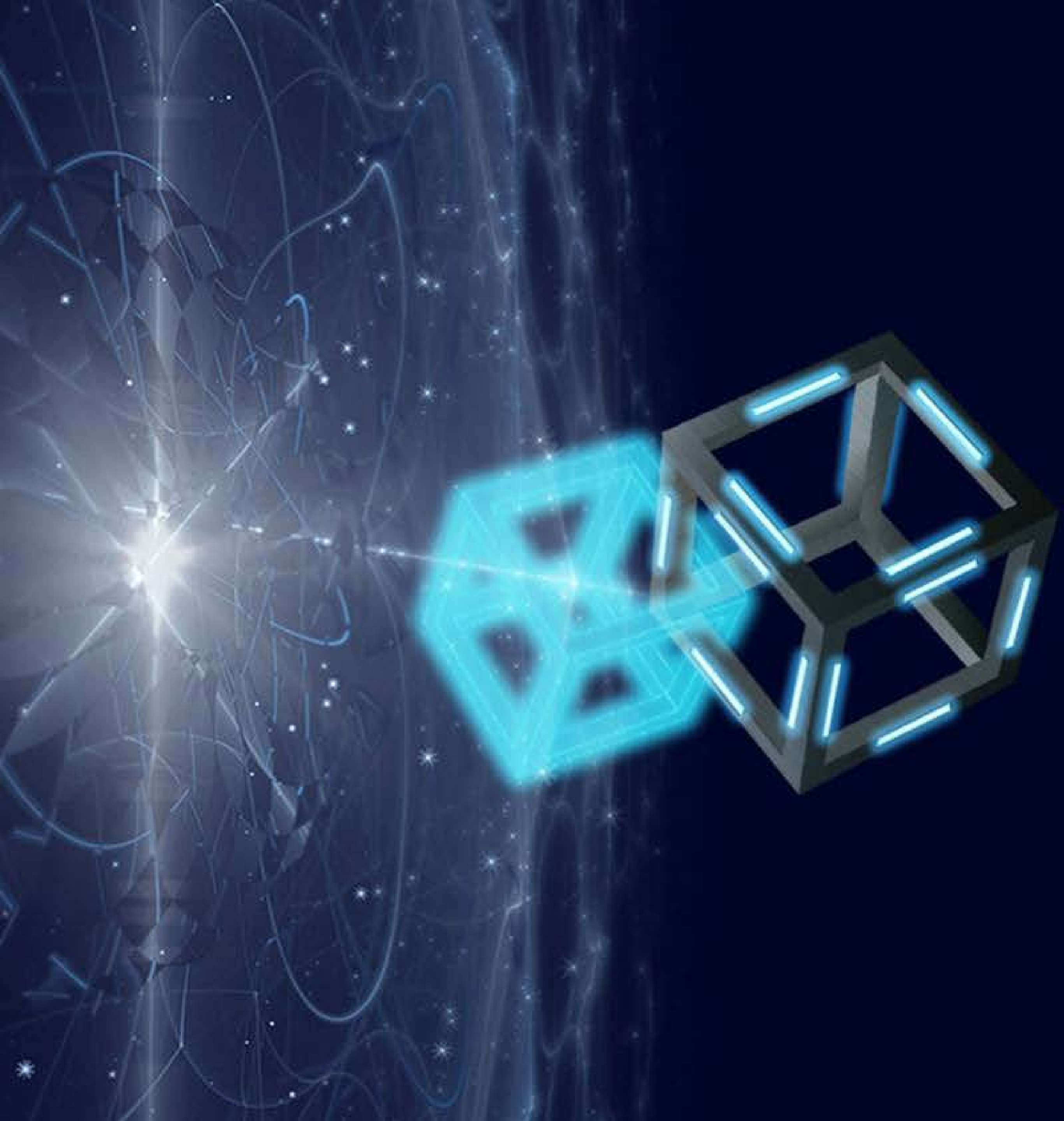
- Configurations for RPC endpoints and Etherscan (for example: Sepolia) are in config.toml
- Setup of environment variables (e.g.: PK_SEPOLIA): in .bash_profile (C:/User/USERNAME/) => export PK_SEPOLIA="0x77..."

## Foundry Configuration - foundry.toml



```toml
[profile.default]
src = "src"
out = "out"
libs = ["lib"]
solc_version = "0.8.20"
optimizer = true
optimizer_runs = 200

remappings = [
    "openzeppelin-contracts/=lib/openzeppelin-contracts/"
]

[rpc_endpoints]
sepolia = "${ALCHEMY_SEPOLIA_API_URL}"

[etherscan]
sepolia = { key = "${ETHERSCAN_SEPOLIA_API_KEY}", url = "https://api-sepolia.etherscan.io/api" }

[fuzz]
runs = 256
depth = 15
fail_on_revert = false

```

**Additional options:** https://book.getfoundry.sh/reference/config/

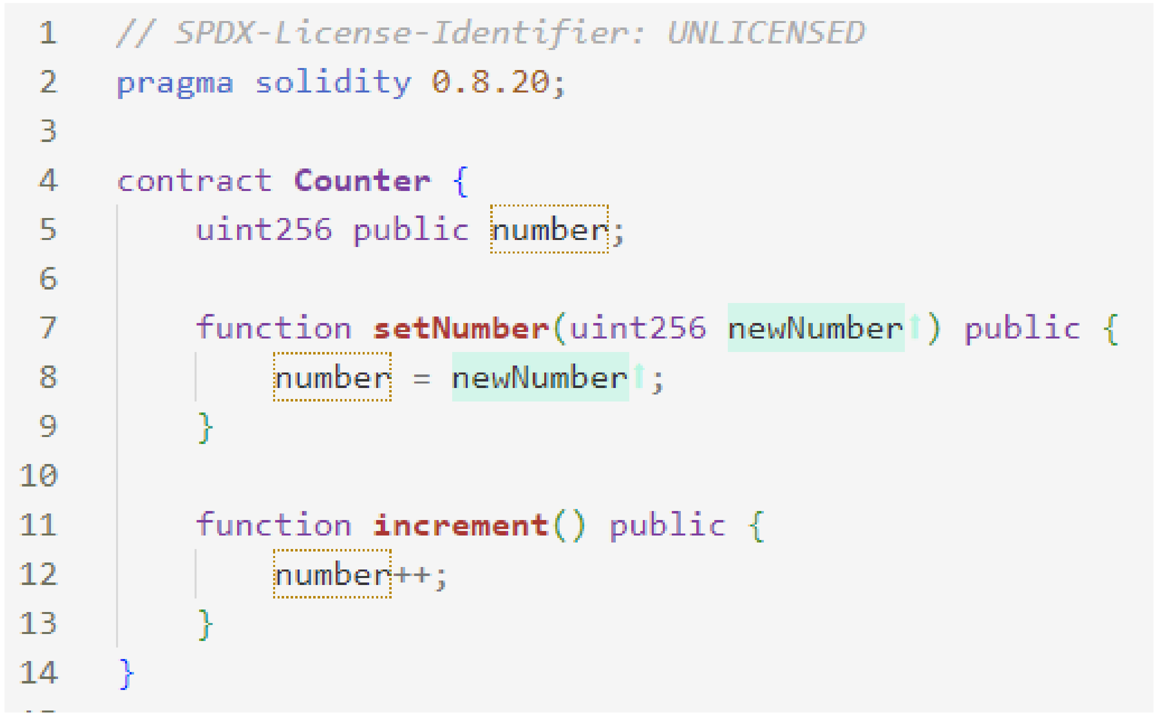


## Testing with Foundry

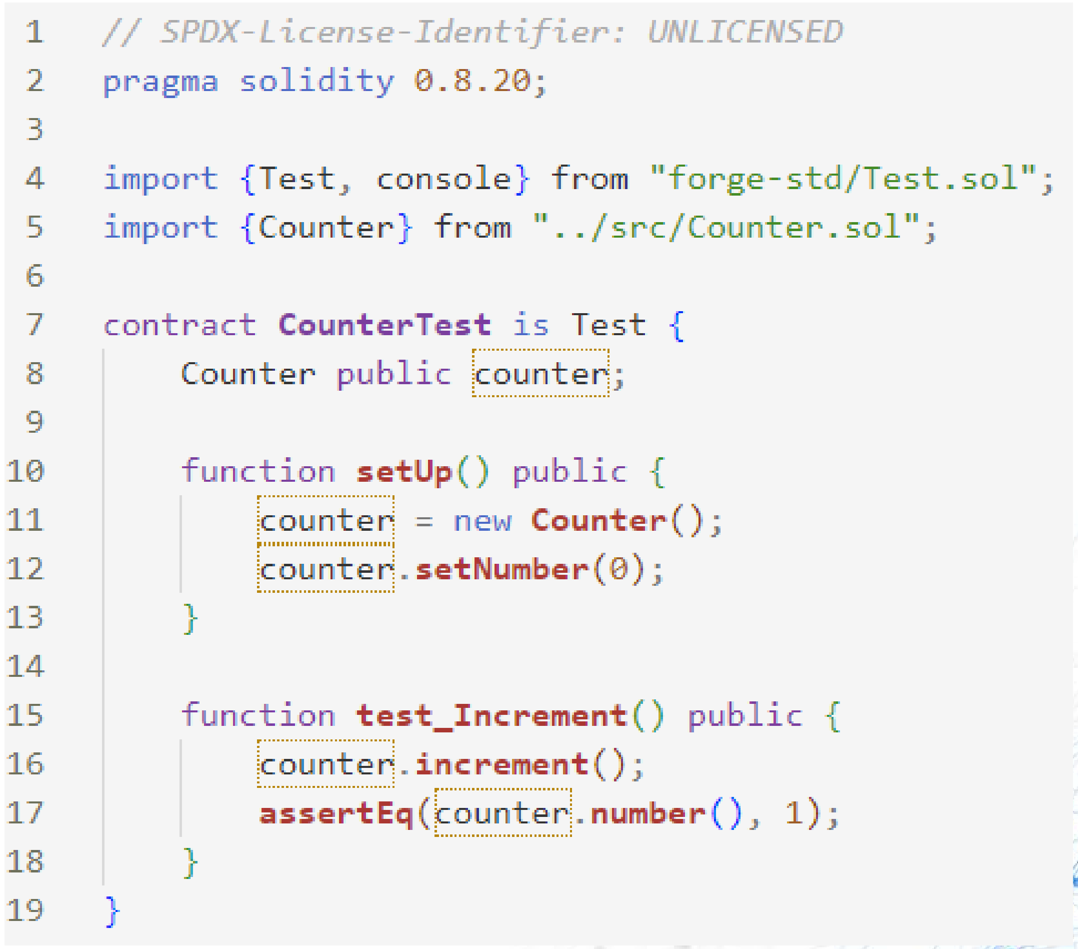
## Testing with Foundry

- Tests are written in Solidity. If the test function reverts, the test fails, otherwise it passes
- Functions prefixed with test_ are run as a test case
- Test functions must have either external or public visibility
- Preferred way of writing tests: using the Forge Standard Library’s Test contract =>
  ***import "forge-std/Test.sol";***
- ***setup():*** optional function invoked before each test case is run => often used to deploy other contract(s) that should be tested

## Testing with Foundry



```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.20;

contract Counter {
    uint256 public number;

    function setNumber(uint256 newNumber) public {
        number = newNumber;
    }

    function increment() public {
        number++;
    }
}
```



```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.20;

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

contract CounterTest is Test {
    Counter public counter;

    function setUp() public {
        counter = new Counter();
        counter.setNumber(0);
    }

    function test_Increment() public {
        counter.increment();
        assertEq(counter.number(), 1);
    }
}
```

# Foundry Asserts

- ***assertEq***: assert equal
- ***assertLt***: assert less than
- ***assertLe***: assert less than or equal to
- ***assertGt***: assert greater than
- ***assertGe***: assert greater than or equal to
- ***assertTrue***: assert to be true
- The first two arguments of the assert are the comparison arguments. An error message can be provided as a third argument:

  ***assertEq(someNumber, 55, "expect someNumber to equal to 55");***

# Foundry Cheatcodes - Accounts

**Changing the msg.sender to a specific address:**

- ***vm.prank(someAddress)***;
  myContract.someFunction(); //msg.sender is someAddress

- vm.prank only works for the transaction that happens immediately after. If several transactions should use the same address, use vm.startPrank and vm.stopPrank

  ***vm.startPrank(owner);***
  myContract.function1();
  myContract.function2();
  ***vm.stopPrank();***

**Addresses:**

- address owner = address(1234);
- address owner = 0x0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045;
- address alice = ***makeAddr("alice");***

# Foundry Cheatcodes – Reverts, Errors & Events

- **Testing Reverts:**

  ***vm.expectRevert("incorrect amount");***
  someContract.depositExactly1Ether{value: 1 ether + 1 wei}();

- **Testing custom errors:**

  error SomeError(uint256); // the specific error needs to be declared in the test contract

  ***vm.expectRevert(abi.encodeWithSelector(SomeError.selector, 5));***
  customErrorContract.functionReverstsWithSomeError(5);

- **Testing Events:**

  The event needs to be emitted in the test to ensure it worked in the smart contract

  ***vm.expectEmit();***
  emit EventName();
  someContract.functionThatEmitsEvent();

## Foundry Cheatcodes – Timestamp & Balances

- **Adjusting the block.timestamp**

  ***vm.warp(1680616584 + 3 days);***

- **Adjusting the block.number**

  ***vm.roll(1000);***

- **Setting address balances:**

  address alice = makeAddr("alice");
  ***vm.deal(alice, balanceToGive);***

## Example: Testing a simple NFT Contract

- **Install dependencies:**

  ***forge install*** transmissions11/solmate Openzeppelin/openzeppelin-contracts

- **Testing the contract:**

  ***forge test*** --match-contract NFTTest --gas-report –vv
  ***forge test*** --match-contract NFTTest --match-test test_RevertMintWithoutValue --gas-report -vv

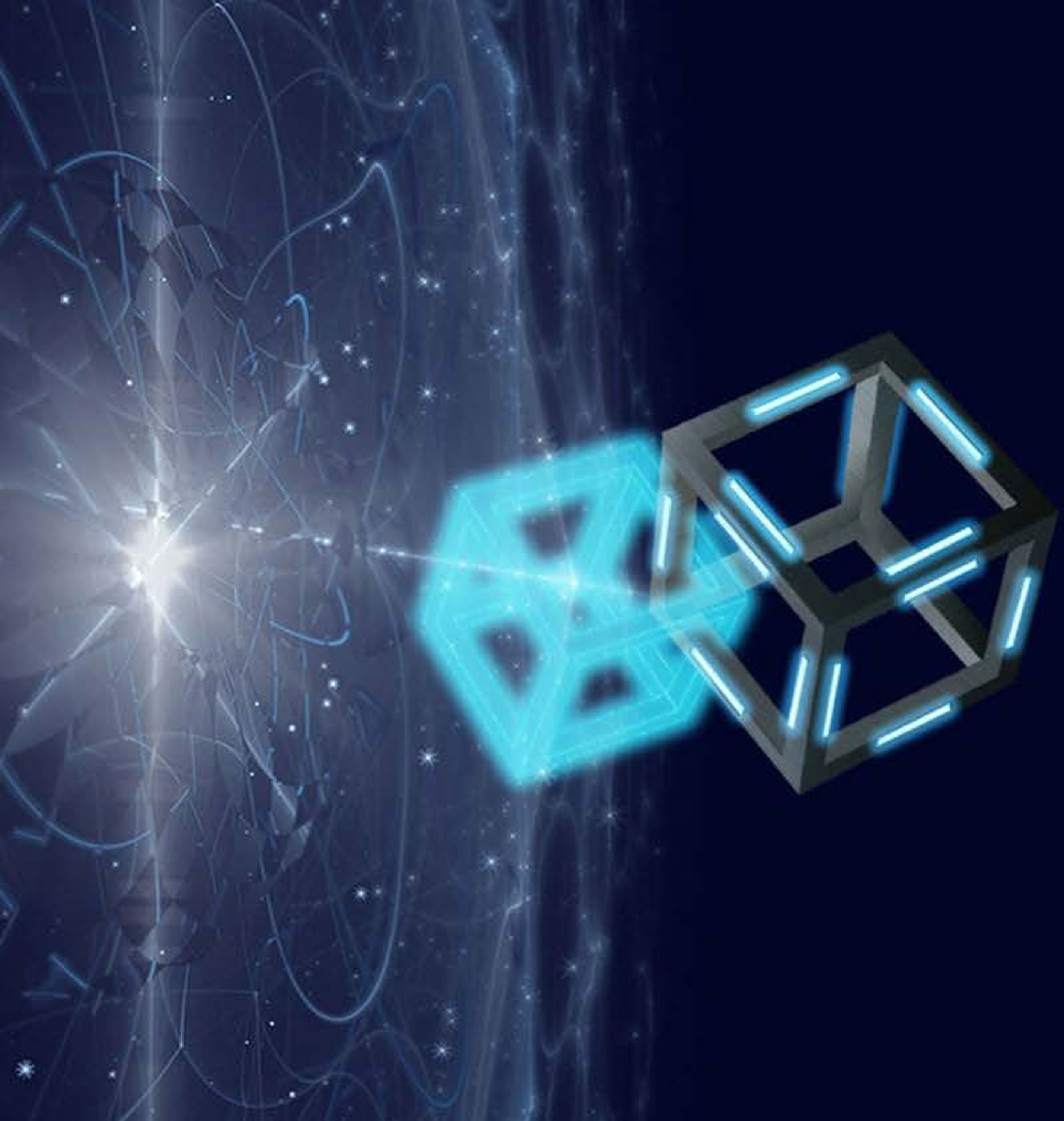
- **Deploy and verify the contract:**

  ***forge create*** --rpc-url sepolia --private-key $PK_SEPOLIA --constructor-args "My NFT" "MNFT" "baseUri" --etherscan-api-key sepolia --verify src/NFT.sol:NFT

- **Executing contract functions using cast**

  ***cast send*** --rpc-url=sepolia --private-key=$PK_SEPOLIA "CONTRACT_ADDRESS" "mintTo(address)" "RECEIVER_ADDRESS"

  ***cast call*** --rpc-url=sepolia "CONTRACT_ADDRESS" "ownerOf(uint256)" 1



## Foundry Fuzz Tests
## Stateless Fuzzing

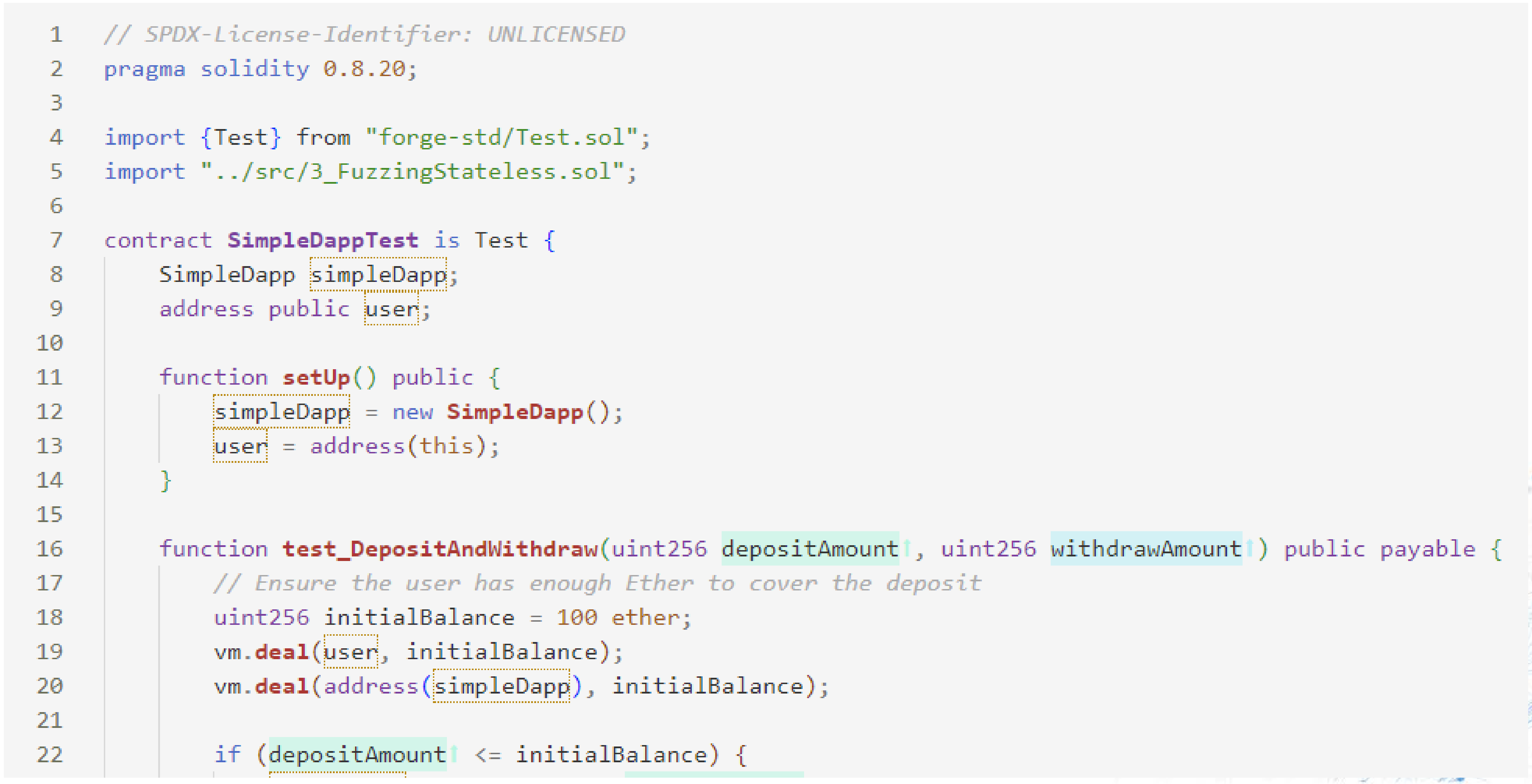
# Foundry Fuzz Tests - Stateless Fuzzing

- Stateless fuzzing: the state of the variables will be forgotten on each run
- Foundry runs any test that takes at least one parameter as a fuzz test
- Foundry runs the test with different values for the specified arguments
- **Fuzz Configuration:**
  - The number of runs and other parameters can be configured in the [fuzz] section of the foundry.toml file =>
  - ***runs***: The amount of fuzz runs to perform for each fuzz test case - default: 256
  - ***depth***: The number of calls executed to attempt to break invariants in one run – default: 15
  - ***fail_on_revert***: Fails the fuzz test if a revert occurs – default: false
  - Additional parameters: https://book.getfoundry.sh/reference/config/testing#fuzz

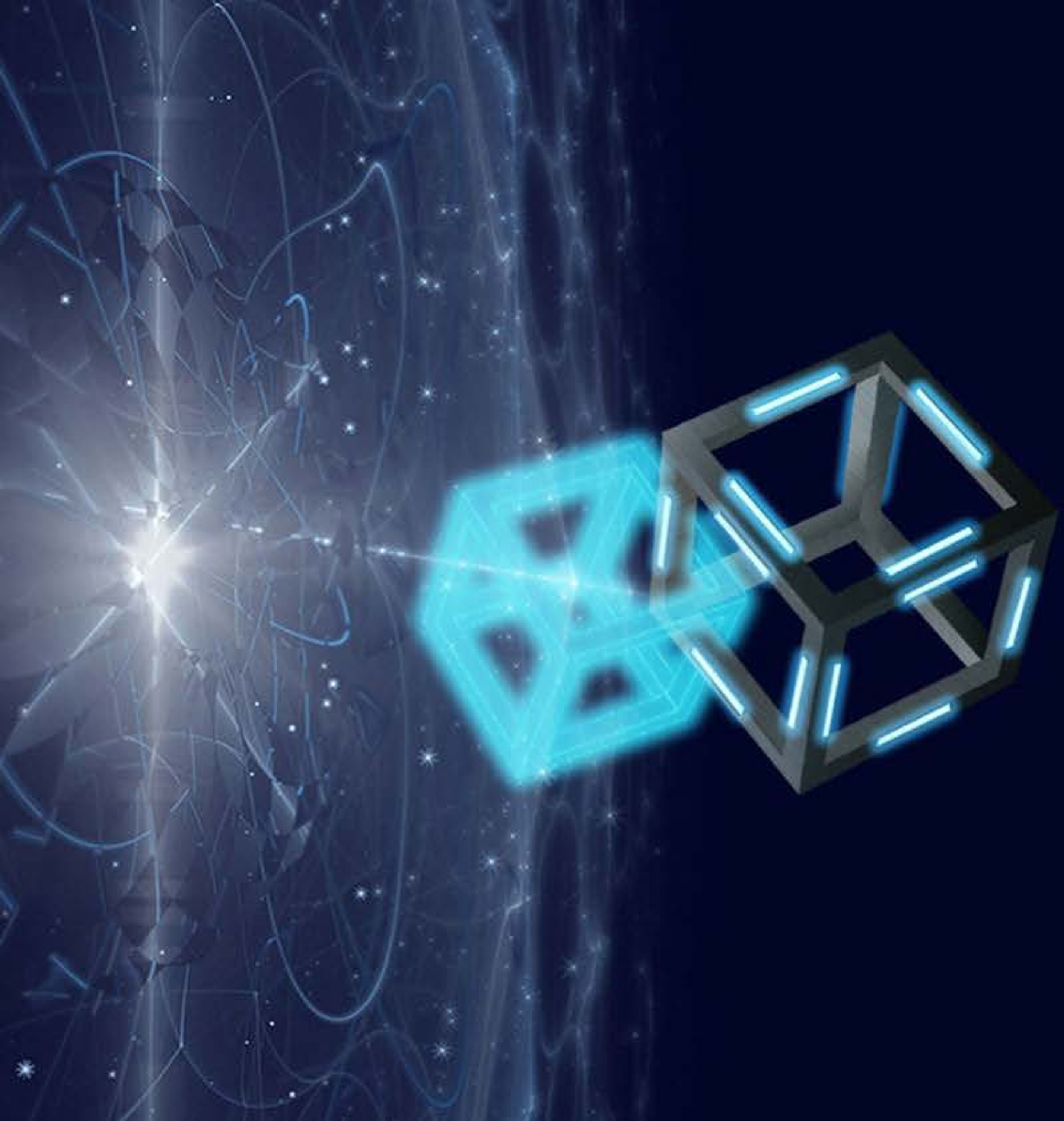
# Example: SimpleDapp – Stateless Fuzzing

- An invariant is a condition that must always be true
- ***Our Invariant***: a user should never be able to withdraw more money than they deposited
- Import standard test library (forge-std/Test.sol)
- Inherit test contract from the standard test library
- In the setup() function, deploy the contract that should be tested
- Create test functions with input parameters
- Foundry will call those test functions with random input parameters
- Make sure, the invariant holds, otherwise the test should fail

# Example: SimpleDapp – Stateless Fuzzing



```solidity
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.20;

import {Test} from "forge-std/Test.sol";
import "../src/3_FuzzingStateless.sol";

contract SimpleDappTest is Test {
    SimpleDapp simpleDapp;
    address public user;

    function setUp() public {
        simpleDapp = new SimpleDapp();
        user = address(this);
    }

    function test_DepositAndWithdraw(uint256 depositAmount, uint256 withdrawAmount) public payable {
        // Ensure the user has enough Ether to cover the deposit
        uint256 initialBalance = 100 ether;
        vm.deal(user, initialBalance);
        vm.deal(address(simpleDapp), initialBalance);

        if (depositAmount <= initialBalance) {
```



## Foundry Invariant Tests
## Stateful Fuzzing

## Foundry Invariant Tests - Stateful Fuzzing

- **Stateful fuzzing:** the state of our previous run is the starting state of our next fuzz run

- In a stateful fuzz test, a contract's functions are called randomly with random inputs by the fuzzer, trying to break any specified invariant

- To write a stateful fuzz test in Foundry, use the invariant keyword:

  ***function invariant_testAlwaysReturnsZero () public { ...***

## Example: Stateful Fuzzing

- **Our invariant:** any amount deposited should be withdraw-able by the same person
- **targetContract():** allows us to define the contract we will put to the test. By defining a target contract, Foundry will automatically start executing all the contract functions randomly and setting random input parameters

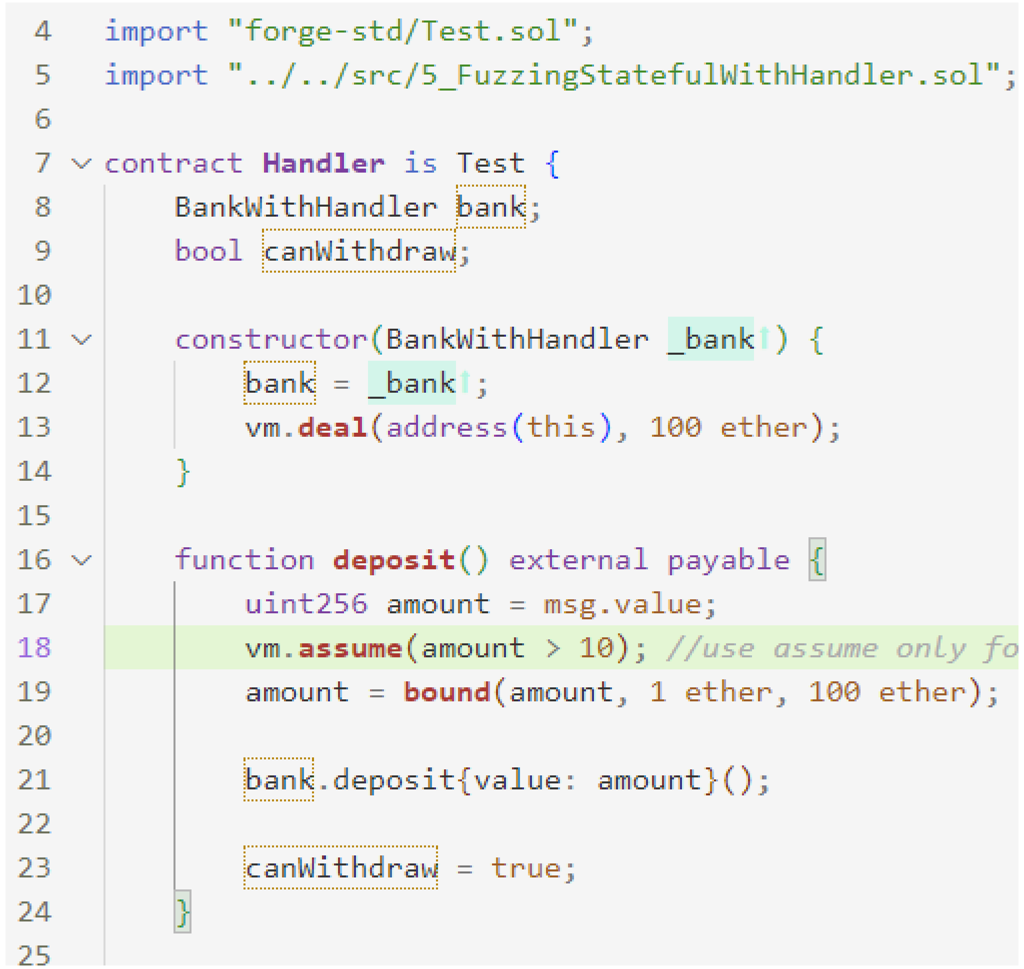
  ***targetContract(address(SelectedContract));***

- **Run the test:** forge test --match-contract BankTest --mt invariant_alwaysWithdrawable
- Why is changeBalance() called ?

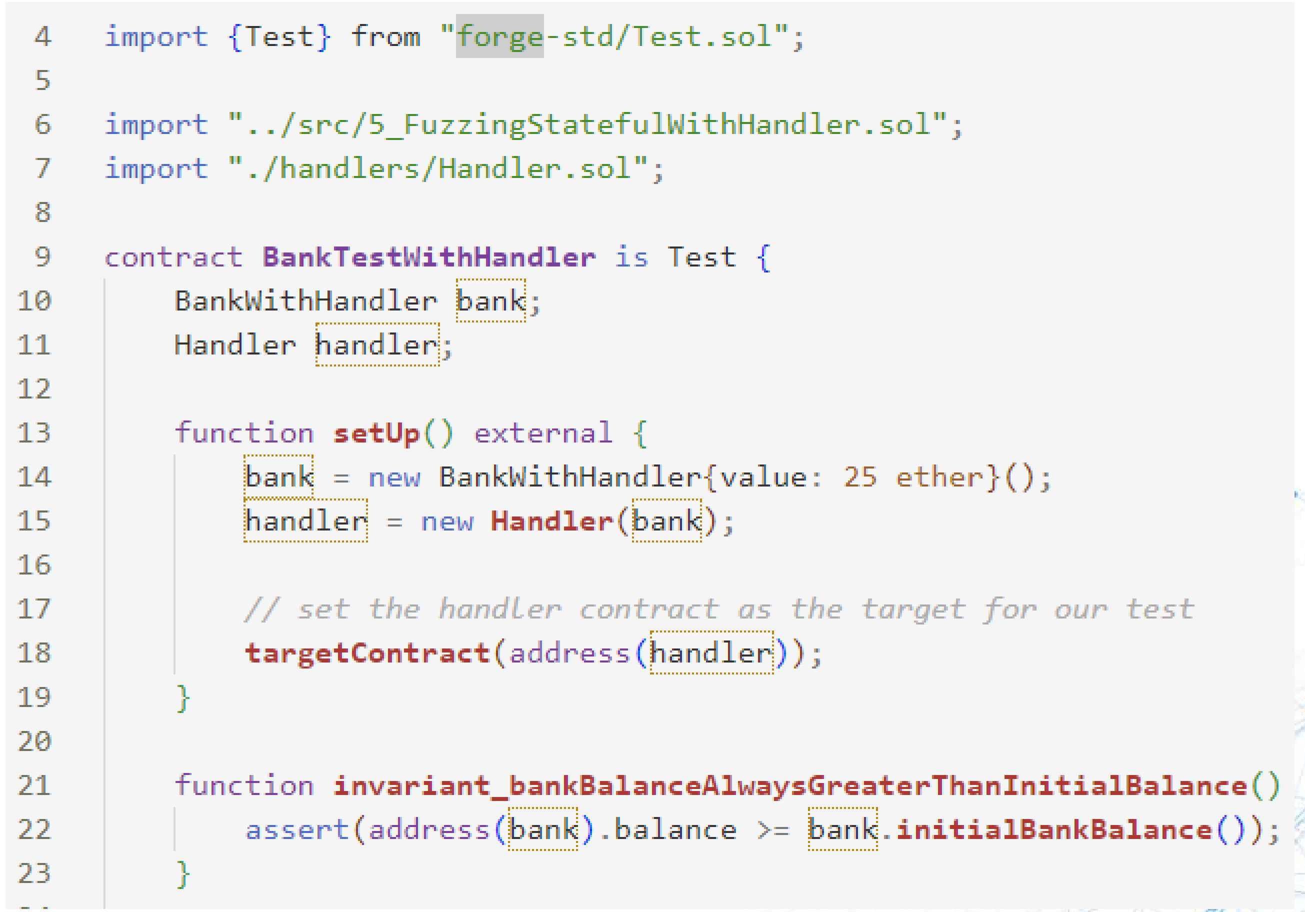
## Handler-based Testing

- A handler contract is used to test more complex protocols or contracts - necessary when the environment needs to be configured in a specific way
- The handler is a wrapper contract that is used to interact with the target contract
- Create a ***handler*** folder inside the test folder and a ***handler.sol*** file inside of it with wrapper functions for all target functions that should be called by the fuzzer
- Create a test file and deploy the handler in the ***setup***() function
- Set the handler contract as the target contract using the ***targetContract()*** helper function
- Add invariant test functions to the test file => function names start with: ***invariant_*** and they assert protocol specific invariants
- Only the functions defined in the handler contract will be called randomly by the fuzzer
- If a function in the main contract requires a certain condition before it can be called, we can easily define it in the handler contract before the function call

# Handler-based Testing



```
import "forge-std/Test.sol";
import "../../src/5_FuzzingStatefulWithHandler.sol";

contract Handler is Test {
    BankWithHandler bank;
    bool canWithdraw;

    constructor(BankWithHandler _bank) {
        bank = _bank;
        vm.deal(address(this), 100 ether);
    }

    function deposit() external payable {
        uint256 amount = msg.value;
        vm.assume(amount > 10); //use assume only fo
        amount = bound(amount, 1 ether, 100 ether);

        bank.deposit{value: amount}();

        canWithdraw = true;
    }

```



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

import "../src/5_FuzzingStatefulWithHandler.sol";
import "./handlers/Handler.sol";

contract BankTestWithHandler is Test {
    BankWithHandler bank;
    Handler handler;

    function setUp() external {
        bank = new BankWithHandler{value: 25 ether}();
        handler = new Handler(bank);

        // set the handler contract as the target for our test
        targetContract(address(handler));
    }

    function invariant_bankBalanceAlwaysGreaterThanInitialBalance()
        assert(address(bank).balance >= bank.initialBankBalance());
    }
```