



Chainlink

Smart Contract Developer Bootcamp

Day 2: Hardhat Track



Smart Contract Developer Bootcamp

Day 2

Hardhat Development Environment, and the
Hardhat Starter Kit





Harry Papacharissiou

Developer Advocate, [Chainlink Labs](#)

✉️ harry@chainlinklabs.com

🐦 [@Pappas9999](#)

linkedin [harrypapacharissiou](#)

Smart Contract Developer Bootcamp Outline



Day 1: Introduction

Learn about blockchain, smart contracts and solidity, followed by examples of how to use Chainlink Price Feeds, Any-API and VRF in your smart contracts. Working with Solidity



Day 2: Development Environments

Learn how to use the Hardhat development environment, followed by a guide on using our starter kits to create smart contracts that use Chainlink. Refining your project with local deployments and testing

Agenda: Day 2

- 1.** Introduction to Development Environments
- 2.** Hardhat Development Environment
- 3.** Chainlinks Hardhat Starter Kit
- 4.** Working with a Local Network
- 5.** Smart Contract Security
- 6.** Testing Smart Contracts
- 7.** Where Can I Learn More

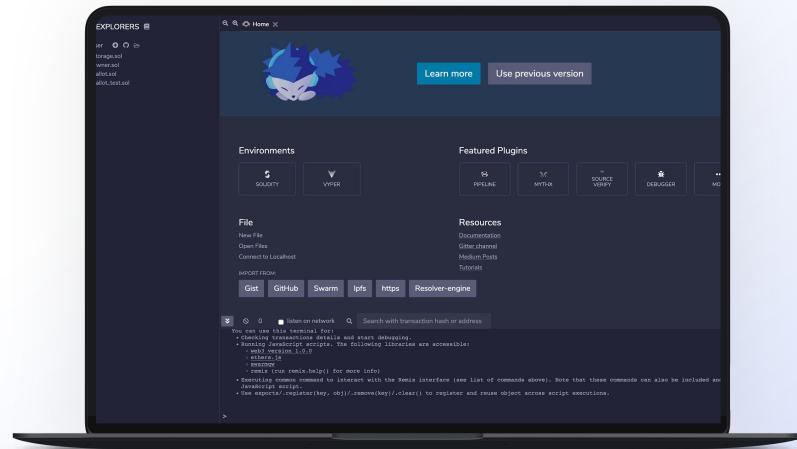
Housekeeping Rules

- 4 hour session, broken up into sections, with breaks
- Presentation, followed by questions, demo and exercise
- Questions can be asked at the end of each section
- Zoom: If you need help during a practical exercise, raise your hand and an instructor will eventually get to you and send you an invite to a private breakout session 
- YouTube: Technical questions can be asked in the chat or in the #developer-bootcamp channel in the Chainlink Discord

Introduction to Development Environments

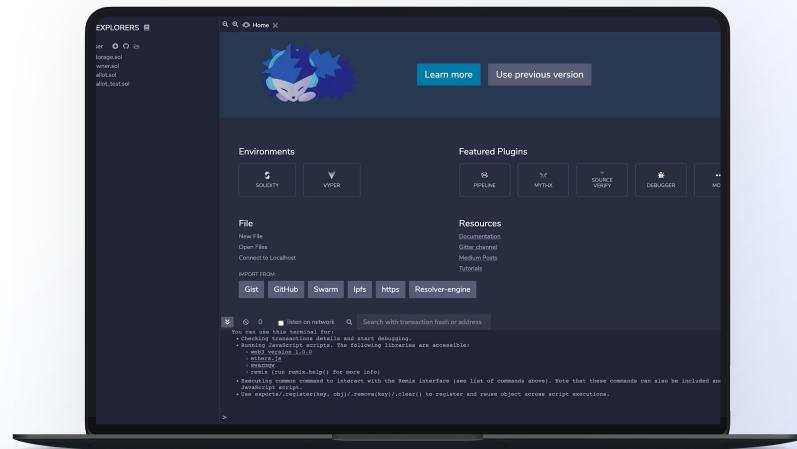
Introduction to Development Environments

- Set of software tools and processes to develop source code
- Used to build applications
- Designed to maximize productivity and efficiency
- Can be local or web-based



Remix IDE (Integrated Development Environment)

- Web-based (also a desktop version)
- Modules for testing, debugging and deploying smart contracts
- Local JavaScript VM to simulate Ethereum
- Extremely quick and easy to create and test smart contracts



Limitations of Remix

- Only deals with smart contracts
- Can't integrate other parts of a project
- Limited support for tests or custom deployments
- Can't save files locally without a plugin
- Any custom functionality requires a plugin



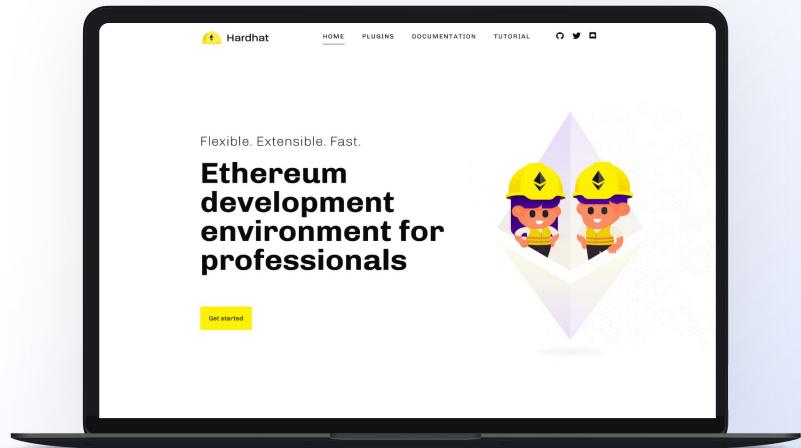


Hardhat



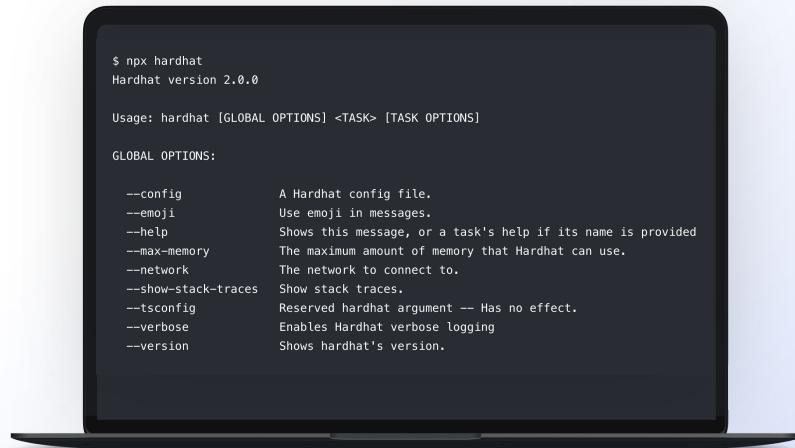
Hardhat Development Environment

- Flexible JavaScript based Development Environment to compile, deploy, test and debug EVM based smart contracts
- Enables easy integration of code and external tools
- Local Hardhat Network to simulate Ethereum
- Extensible plugin features
- High level of debugging features



Hardhat Tasks and Plugins

- JavaScript asynchronous function with associated metadata.
- Used to automate common tasks
- Everything you can do in Hardhat is defined as a task
- Can package them into plugins that can be imported by another project
- Hardhat has a large range of [plugins](#) listed, including support for [web3.js](#) and [ethers.js](#)



```
$ npx hardhat
Hardhat version 2.0.0

Usage: hardhat [GLOBAL OPTIONS] <TASK> [TASK OPTIONS]

GLOBAL OPTIONS:

--config           A Hardhat config file.
--emoji            Use emoji in messages.
--help             Shows this message, or a task's help if its name is provided.
--max-memory      The maximum amount of memory that Hardhat can use.
--network          The network to connect to.
--show-stack-traces Show stack traces.
--tsconfig         Reserved hardhat argument -- Has no effect.
--verbose          Enables Hardhat verbose logging
--version          Shows hardhat's version.
```

Demo 1: Creating Your First Hardhat Project

Exercise 1: Creating Your First Hardhat Project

- Hardhat: <https://hardhat.org/>
- 45 mins + 5 mins break



Hardhat Starter Kit

Solidity and Smart Contract Starter Kits



SmartContract

Part of SMARTCONTRACT INC

Repositories 140 Packages People 88 Teams 13 Projects 1 Insights

Pinned repositories

 [truffle-starter-kit](#)

An example smart contract utilizing Chainlink

JavaScript 66 36

 [chainlink-mix](#)

Working with smart contracts with eth-brownie, python, and Chainlink.

Solidity 39 13

 [hardhat-starter-kit](#)

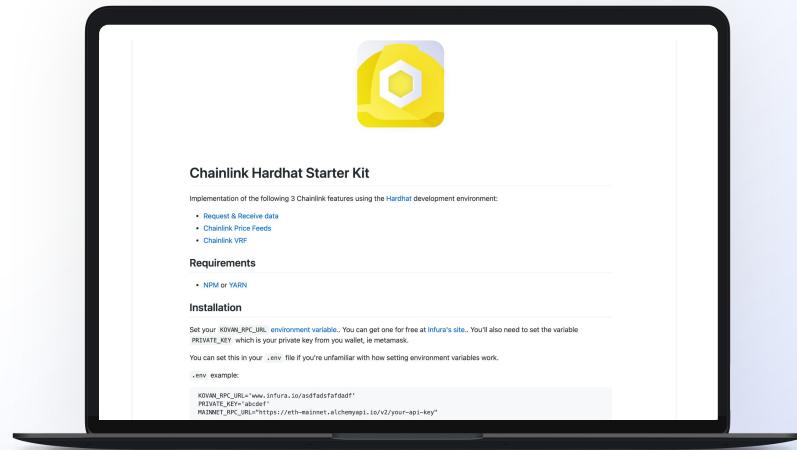
A repo for boilerplate code for testing, deploying, and shipping chainlink solidity code.

JavaScript 13 4



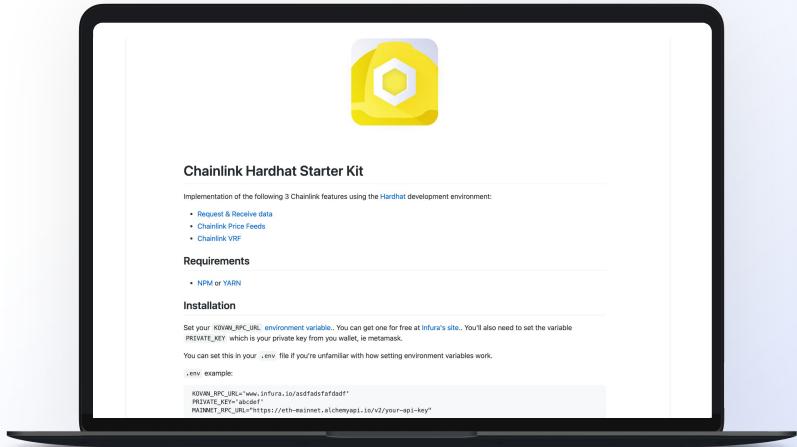
Hardhat Starter Kit

- <https://github.com/smartcontractkit/hardhat-starter-kit>
- Prepackaged Hardhat repository for building smart contracts that use Chainlink
 - Smart contracts
 - Deployment scripts
 - Tests
 - Tasks
 - Configuration



Hardhat Starter Kit

- Allows quick building, deploying and testing of smart contracts that use Chainlink
- Can be integrated with other parts of a project
- Increases productivity via the use of custom Chainlink tasks



Demo 2: Installing and Using the Hardhat Starter Kit

Exercise 2: Installing and Using the Hardhat Starter Kit

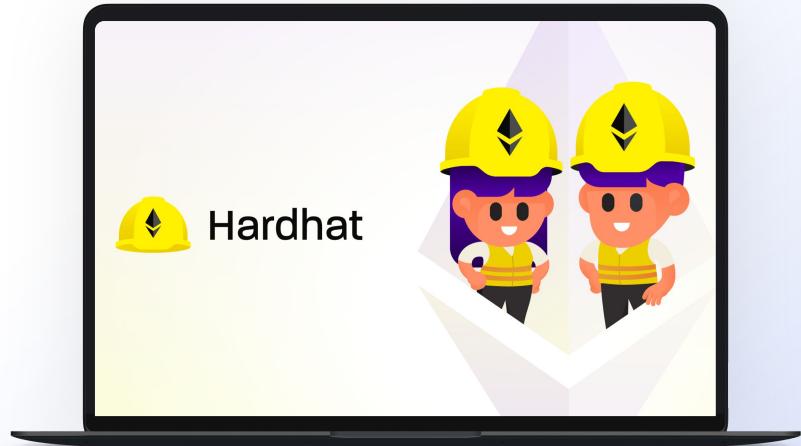
- <https://github.com/smartcontractkit/hardhat-starter-kit>
- 45 minutes



Working With a Local Network

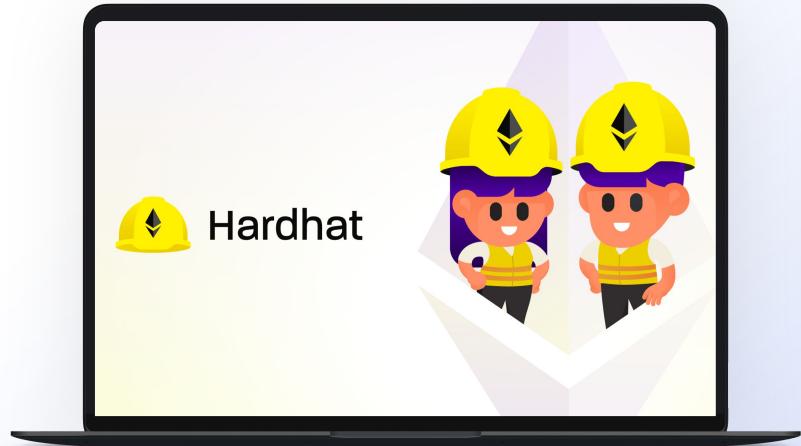
Hardhat Network

- Local Ethereum network
- Allows you to deploy, run and test your code
- Can be run in a standalone fashion, and connected to external clients



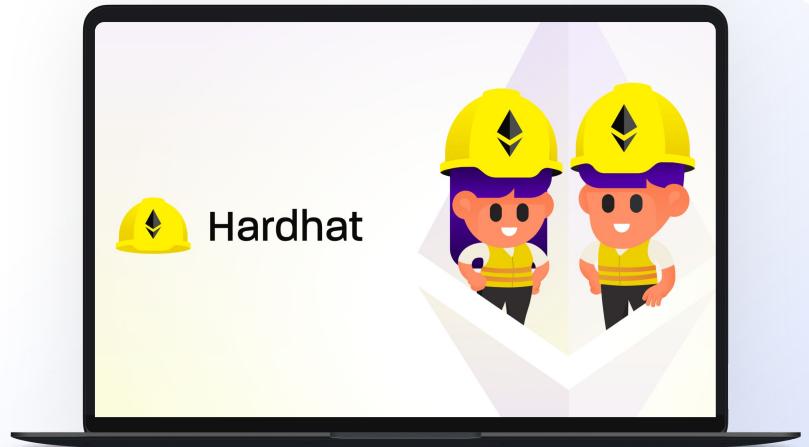
Hardhat Network

- Mines a block with each transaction received, in order and with no delay
- Can be used to run tests
- Compatible with plugins
- Can be thought of as another network (like Kovan, Mainnet etc)



Hardhat Network

- Can fork Mainnet!
- Initial state:
 - Genesis block
 - 20 accounts with 10000 ETH each



Design Patterns for Local Deployments: Mocks

- Common design pattern involving creating a second version of something that imitates the original
- Useful for tests on local networks where state isn't the same as a public network, or where there is no integration to external services such as Chainlink oracle networks

```
20● contract("ERC20Regular Contract Test Suite", accounts => {
21  "use strict";
22
23  if(accounts.length > 0) accounts = (new Chance()).pickset(accounts, 1); // avoid too many accounts
24  const EventNames = { Transfer: 'Transfer', Approval: 'Approval' };
25
26● before(async() => {
27  });
28● describe("Initial State", () => []);
29● describe("Minting", () => []);
30● describe("Transfer", () => {
31  it(`Can transfer from decreasing sender's balance and increasing recipient's balance as much.`, async() => {
32    const chance = new Chance();
33    const admin = chance.pickone(accounts);
34    const token = await Token.new("Color Token", "RGB", {from: admin});
35    const tokenContract = await token.deploy({address: $`{token.address}`});
36    console.debug(`New token contract deployed at address: ${token.address}`);
37
38    let balance = 0;
39    for(let i = 0; i < accounts.length; i++) {
40      balance = toBN(i*10).mul(chance.natural({min: 1, max: 100}));
41      await token.mint(accounts[i], balance, {from: admin});
42    }
43
44    const loops = 20;
45    let sender = null, recipient = null, delta = 0;
46    let senderBal = 0, recipientBal = 0, recipientBal2 = 0;
47    for(let i = 0; i < loops; i++) {
48      [sender, recipient] = chance.pickset(accounts, 2);
49      senderBal = await token.balanceOf(sender);
50      recipientBal = await token.balanceOf(recipient);
51
52      delta = chance.bool({likelihood: 10}) ? toBN(1E10).mul(chance.natural({min: 1, max: 1000000}));
53      await token.transfer(recipient, delta, {from: sender});
54
55      senderBal = await token.balanceOf(sender);
56      recipientBal2 = await token.balanceOf(recipient);
57
58      expect(senderBal).to.be.below(delta);
59      expect(recipientBal2).to.be.above(delta);
60    }
61  });
62});
```



Demo 3: Deploying to a Local Network

Exercise 3: Deploying to a Local Network

- Deploy our smart contracts to a local network
- Fork Mainnet
- 30 minutes + 5 minute break



Smart Contract Security

“
*Everyone here is a
target for attack.
Be paranoid.*



- **Mertin Swende**
Ethereum Foundation Security Lead

Smart Contract Security

- Two main sources of security problems:
 - Bugs in the blockchain infrastructure
 - Protocols
 - EVM
 - Bugs or Vulnerabilities in the smart contract code
 - Developer mistakes or unanticipated code execution



Smart Contract Security

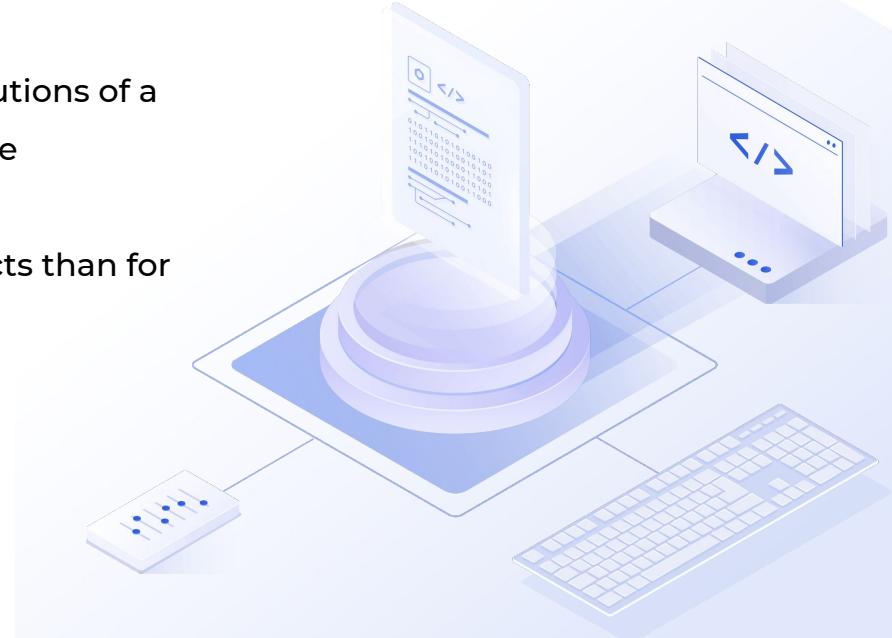
- Programming smart contracts requires a different engineering mindset
- Cost of failure can be high
- Changes can be difficult
 - Similar to hardware development
- Multiple well known smart contract attacks:
 - Reentrancy attack
 - Cross-function race conditions
 - Integer overflow
 - DoS attack (revert, block gas limit)
 - Forcibly sending ether to a contract
 - TX reordering attacks
 - Timestamp dependence attack



Testing Smart Contracts

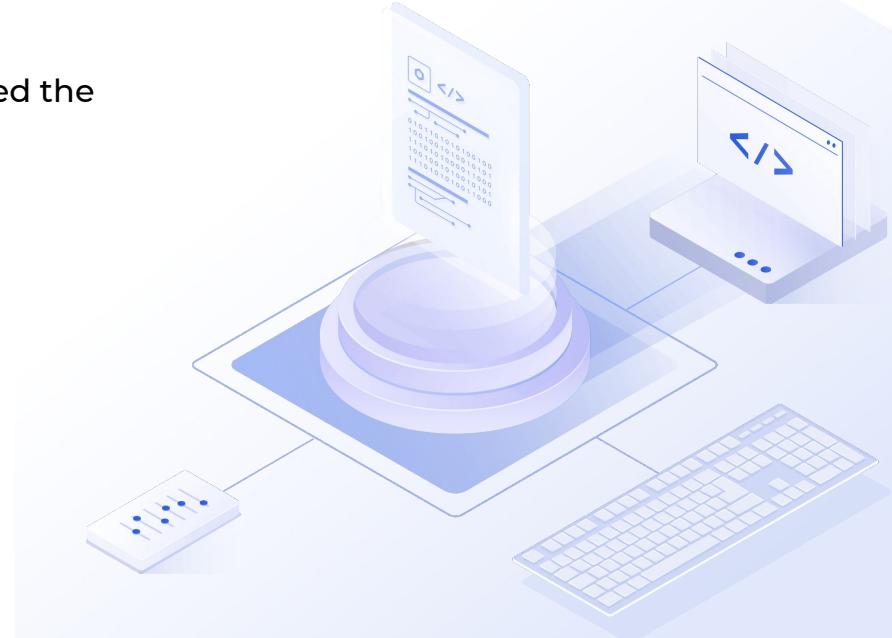
Testing Your Smart Contracts

- Executing code in order to identify any gaps, errors, or missing requirements
- Combination of subsequent, structured executions of a smart contract, validating the result each time
- Smart contracts are immutable
- Testing is even more crucial for smart contracts than for traditional applications.



Testing Your Smart Contracts

- Rule of thumb: Every line of contract logic should have a corresponding test.
- The most important scenarios should be tested the most thoroughly.
- Tests can catch careless mistakes.
- Tests are great for edge cases.



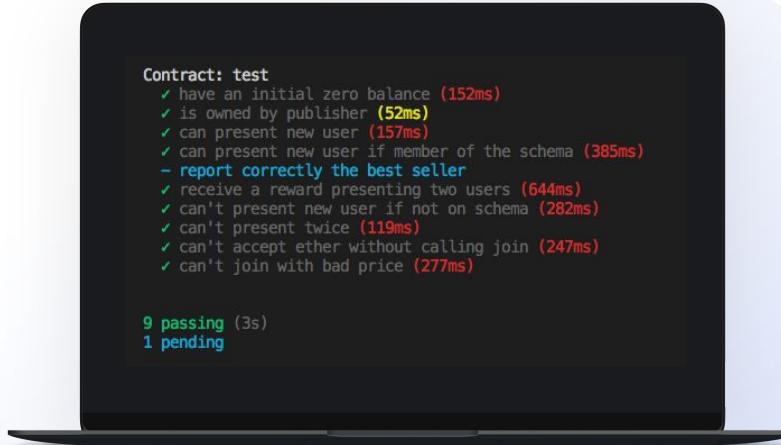
Bug vs Vulnerability

- A bug is when an issue leads to a planned scenario not executing, or executing incorrectly
- A vulnerability is when an issue leads to an unplanned scenario executing
- Tests do not prevent vulnerabilities.



Types of Tests

- Unit tests
 - Simple tests that verify a single behavior or component within a smart contract
- Integration tests
 - More complex tests that validate interactions between multiple components, including off-chain components such as oracles

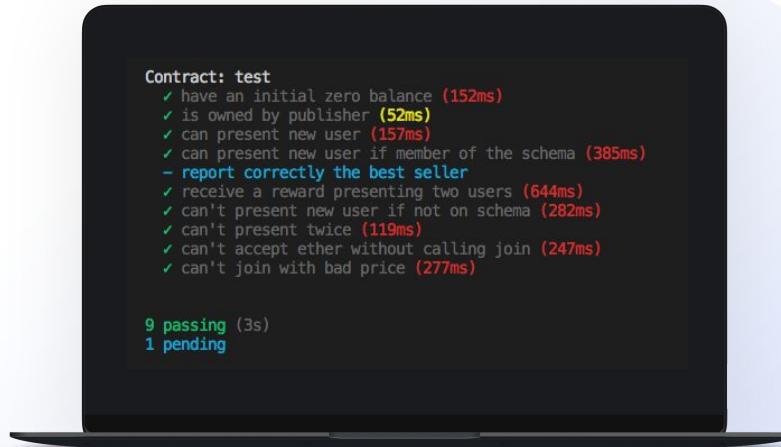


```
Contract: test
✓ have an initial zero balance (152ms)
✓ is owned by publisher (52ms)
✓ can present new user (157ms)
✓ can present new user if member of the schema (385ms)
- report correctly the best seller
✓ receive a reward presenting two users (644ms)
✓ can't present new user if not on schema (282ms)
✓ can't present twice (119ms)
✓ can't accept ether without calling join (247ms)
✓ can't join with bad price (277ms)

9 passing (3s)
1 pending
```

Testing Approach

- Arrange-Act-Assert (AAA) Testing Pattern
 - Arrange the initial conditions required
 - Act, by calling the function to be tested
 - Assert the result of the action



```
Contract: test
✓ have an initial zero balance (152ms)
✓ is owned by publisher (52ms)
✓ can present new user (157ms)
✓ can present new user if member of the schema (385ms)
- report correctly the best seller
✓ receive a reward presenting two users (644ms)
✓ can't present new user if not on schema (282ms)
✓ can't present twice (119ms)
✓ can't accept ether without calling join (247ms)
✓ can't join with bad price (277ms)

9 passing (3s)
1 pending
```

Writing Good Unit Tests



Simple

Each unit test should be evaluating a single behavior

Writing Good Unit Tests



Simple

Each unit test should be evaluating a single behavior

Repeatable

Each unit test should always produce the same result

Writing Good Unit Tests



Simple

Each unit test should be evaluating a single behavior

Repeatable

Each unit test should always produce the same result

Isolated

Each unit test should not depend upon or affect other tests

Writing Good Unit Tests



Simple

Each unit test should be evaluating a single behavior

Repeatable

Each unit test should always produce the same result

Isolated

Each unit test should not depend upon or affect other tests

Readable

Someone else should easily be able to work out what's being tested

Writing Good Unit Tests



Simple

Each unit test should be evaluating a single behavior

Repeatable

Each unit test should always produce the same result

Isolated

Each unit test should not depend upon or affect other tests

Readable

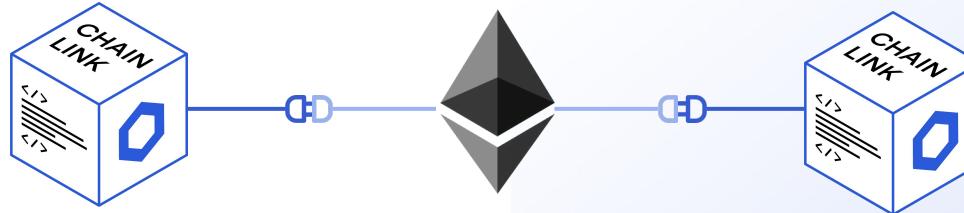
Someone else should easily be able to work out what's being tested

Fast

Unit tests should be written with speed in mind

Integration Tests

- Interactions between contracts
- Interaction between on-chain contracts and off-chain services such as Oracles
- JavaScript tests
 - Ensure contracts exhibit the right external behaviour

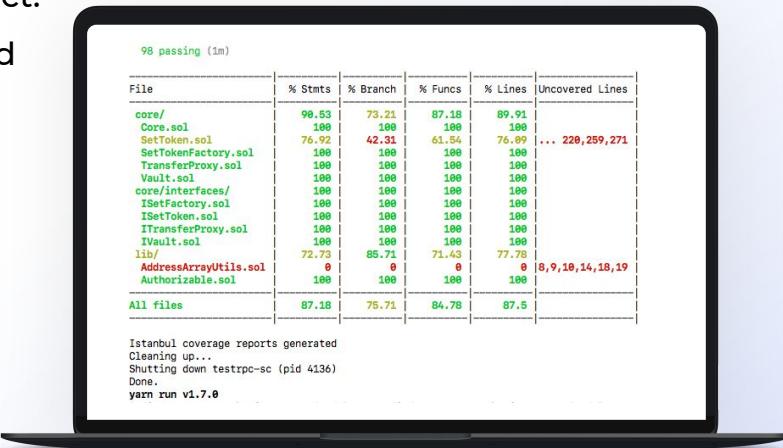


Tools for Testing

- [Waffle](#)
 - Library for writing and testing smart contracts
- [Mocha](#)
 - JavaScript testing framework for browser/Node.js
- [Chai](#)
 - Assertion library for browser/NodeJS

Tools for Testing: Testing Coverage

- <https://github.com/sc-forks/solidity-coverage>
- Tests should ideally ‘touch’ all of the code in a contract.
- Checks to see how much of the code has been tested



Demo 4: Testing Smart Contracts

Exercise 4: Testing Smart Contracts

- Unit Tests
- Integration Tests
- Solidity Coverage
- Publishing to GitHub
- 30 minutes



Developer Bootcamp Day 2 Summary

- Development Environments
- Hardhat Development Environment
- Hardhat Starter Kit
- Local Development in Hardhat
- Smart Contract Security
- Unit and Integration Testing
- Publishing projects to GitHub



Where Can I Learn More

Solidity Resources

- [CryptoZombies](#)
- [Ivan on Tech](#)
- [Dapp University](#)
- [Eat The Blocks](#)
- [Chainshot](#)



docs.chain.link

Beginners - The Basics

Introduction

New to smart contracts?

If you're new to smart contract development this is a great place to start. We walk you through developing your first smart contract that interacts with Chainlink.

Chainlink's most popular feature is our **Price Feeds**. They are the quickest way to connect smart contracts to the real-world market prices of assets. They are also extremely easy to implement.

In this tutorial we go through:

- What smart contracts are
- How to write them
- Why Oracles are important
- How Chainlink price feeds can be used in smart contracts

By the end, you will have written and deployed your very own Chainlink smart contract to an Ethereum testnet!

1. What is a smart contract?

When deployed to a blockchain, a smart contract is a set of instructions that can be executed without intervention from third parties. The code of a smart contract determines how it responds to input, just like the code of any other computer program.

A valuable feature of smart contracts is that they can store and manage on-chain assets (like ETH or ERC20 tokens), just like you or I can with an Ethereum wallet. Because they have an on-chain address, like a wallet, they can do everything any other address can. This opens the door for programming automated actions when receiving and transferring assets.

LINK Token Contracts

★ ★ ★ ★ ★
Suggest Edit

LINK tokens are used to pay node operators for retrieving data for smart contracts and also for deposits placed by node operators as required by contract creators.

The LINK token is an ERC20 token that inherits functionality from the ERC20 token standard and allows token transfers to contain a data payload. Read more about the [ERC20 transferAndCall token standard](#).

Ethereum

Mainnet

Kovan

Kovan Faucets

Testnet LINK is available from <https://kovan.chain.link/>
Testnet ETH is available from <https://faucet.kovan.network/>

Rinkeby

Rinkeby Faucets

Testnet LINK is available from <https://rinkeby.chain.link/>
Testnet ETH is available from <https://faucet.rinkeby.io/>

docs.chain.link

Goerli

The screenshot shows the homepage of the blog.chain.link website. At the top, there's a navigation bar with the Chainlink logo and links for VISION, ANNOUNCEMENTS, EVENTS, DEVELOPERS, EDUCATION, GRANTS, CHINESE, and KOREAN. Below the navigation is a large blue header with the text "blog.chain.link". The main content area is titled "Developers" and features several cards with developer tutorials:

- Winners of the Spring 2021 Chainlink Hackathon**
- Get Index Prices in Solidity Smart Contracts**
- Blockchain Hackathon for Building a Winning Submission**
- Build a dApp on xDai Chain With Secure Data Feeds**
- How to Get a Random Number on Polygon**
- Chainlink Bounty Winners ETHDenver 2021 Hackathon**
- Congratulations to the ETHDenver 2021 Hackathon Bounty Winners**
- Build a dApp on xDai Chain With Secure Data Feeds**
- Fetch Commodity Prices in Solidity Smart Contracts**
- Verify Stablecoin Collateral With Chainlink Proof of Reserve**
- Interpreting Native 2FA To Trigger Smart Contract Events**
- How to Connect a Two-Factor Authentication (2FA) API to a Smart Contract With Chainlink**
- Build Your Own Dynamic NFT With Hardhat**
- Craft Whiskey Crypto Payments With Chainlink Oracles**

Chainlink Youtube

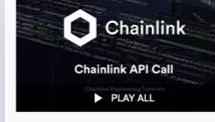
Chainlink Engineering Tutorials & Developer Workshops

 Smart Contract Environment Setup
▶ PLAY ALL

Chainlink Engineering Tutorials
16 videos • 8,933 views • Last updated on May 4, 2021

≡+ X ⌂ ⌂ ⌂

 1 Smart Contract Environment Setup | Chainlink Engineering Tutorials
Smart Contract Development 101 | Chainlink Engineering Tutorials
Getting A Random Number with Chainlink VRF | Chainlink Engineering Tutorials
Chainlink API Call | Chainlink Engineering Tutorials
What is Ethereum? | Chainlink Engineering Tutorials
What is a Blockchain Oracle? What is the Oracle Problem? | Chainlink Engineering Tutorials
Truffle, Hardhat, and Defi - Chainlink Hackathon Workshop
Brownie, Python, and Defi - Chainlink Hackathon Workshop
Testing Smart Contracts & Multi-Chains | Chainlink Hackathon Workshop
Dynamic NFT Creation - Code-Along | Chainlink Hackathon Workshop
Building and Using External Adapters | Chainlink Engineering Tutorials
See Description! Install, Configure, & Fund MetaMask | Chainlink Engineering Tutorials

 Chainlink API Call
▶ PLAY ALL

Chainlink Developer Workshops
43 videos • 2,383 views • Last updated on May 4, 2021

≡+ X ⌂ ⌂ ⌂

 1 Chainlink API Call | Chainlink Engineering Tutorials
Data Providers | Chainlink Hackathon Workshop
Running a local Chainlink Node | Chainlink Hackathon Workshop
Introduction to Smart Contracts, Blockchain, & Solidity - Chainlink Hackathon Workshop
Introduction to Chainlink - Chainlink Hackathon Workshops
Brownie, Python, and Defi - Chainlink Hackathon Workshop
Truffle, Hardhat, and Defi - Chainlink Hackathon Workshop
Ultimate Customization with External Adapters | Chainlink Hackathon Workshop
Dynamic NFT Creation - Code-Along | Chainlink Hackathon Workshop
Testing Smart Contracts & Multi-Chains | Chainlink Hackathon Workshop
Getting A Random Number with Chainlink VRF | Chainlink Engineering Tutorials

Using Chainlink Nodes In Your Project

- [Market.link](#): Check job status with Node Operators
- [Market.link](#): Verification currently being re-worked
- Need a job or an adapter hosted? Post in **#operator-requests** channel in the Chainlink Discord
- Anything can be realized by building customized DoNs (decentralized oracle networks)



Learn more!



Discord Chat

<http://chn.lk/chainlink-discord>



Starter Kits

<https://github.com/smartcontractkit>



Documentation:

<http://docs.chain.link>



Hackathon Resources

<http://chn.lk/hack-resources>



Chainlink

**Congratulations, you've completed the
Summer 2021 Smart Contract Developer
Bootcamp**



Milestone

Milestone

Milestone