

# Introduction to Truffle

---

## Set up your gitpod environment

---

in your browser open

[gitpod.io/#https://github.com/ExtropyIO/Academy](http://gitpod.io/#https://github.com/ExtropyIO/Academy) (<http://gitpod.io/#https://github.com/ExtropyIO/Academy>)

Open the terminal and `cd` to `TruffleIntroduction`

Run

```
npm i
npm i -g ganache-cli
npm i -g truffle
```

## What is Truffle?

---

Truffle is an Ethereum development environment which has built-in compiling and migration, and testing frameworks.

Truffle is part of the Truffle Suite. The Suite contains a range of Ethereum development tools.

## Guide to Truffle

---

### 1. The folder structure

```
contracts
|___ Migrations.sol
|___ VolcanoCoin.sol
|
migrations
|___ 1_initial_migration.js
|
node_modules
|___ @openzeppelin/contracts
|
test
|
| package-lock.json
|
```

```
| package.json
|
| truffle-config.json
```

## Understanding the folder structure

`contracts`

The location for all your contracts.

`Migrations.sol`

The contracts directory is set up with `Migrations.sol`. `Migrations.sol` keeps a track of the migrations to the network.

`migrations`

The location for all your deployment scripts.

`1_initial_migration.js`

The migrations directory is set up by default with `1_initial_migration.js`. These deployment scripts are run when migrating the contracts to the network.

The file's number acts as a reference, that is then recorded in `Migrations.sol` on deployment.

For example, to create another migration, the file name would be

`2_second_migration.js` and this would be referenced as the 2nd migration inside the `Migrations` contract.

`node_modules`

Location for npm packages.

`test`

The location for test files. It's best practice to name your test files as

`myContractsName.test.js`

`package.json`

A JSON that holds metadata for your project.

`truffle-config.js`

The network configuration file (see below).

## 2. Setting up the network

`truffle-config.js` is your configuration file. It has been set up to enable configuration of different network settings.

In our file, we have set up a development network at local host port 8545. 8545 is the standard port for Ethereum.

The network settings must always be exported in an object.

### **3. Ganache CLI**

Ganache CLI is also part of the Truffle suite. It simulates your own personal blockchain for Ethereum development.

Open the terminal and `cd` to `TruffleIntroduction` if you are not already in that directory

In a terminal, run the command `ganache-cli`. You should see a list of 10 test accounts with their private keys, with 100 test Ether to play around with,

Ganache provides a mnemonic key if you want to import the wallet.

It is listening on `127.0.0.1:8545`, which is standard for a local Ethereum blockchain network.

#### Available Accounts

```
=====
(0) 0xE093F79F43800A94A8732c9F81B3ae3C93eAEd3 (100 ETH)
(1) 0xB1501A9b99F7d99267741bE5975f0eff143163FA (100 ETH)
(2) 0x1974A9b5c35B031405e8275D38f101e7339903cE (100 ETH)
(3) 0xD594155F44b07fdeBdCa2e9DdD00c1e15c879105 (100 ETH)
(4) 0xBdefDCfADaeFED3A71A246748373299e59D4a3C3 (100 ETH)
(5) 0x4eDA1d5baB7DaA46F184C4E2320A1b6a79a4DE48 (100 ETH)
(6) 0x8fE2aDD4B7EBd7EaAD15A7Dcef72528E8c7CF231 (100 ETH)
(7) 0x7399Ad93428113bf1bD826E6666A9395e7dE0086 (100 ETH)
(8) 0x1Bc7b99Ec844eB9F3BA4b52fB85A116AF4553e46 (100 ETH)
(9) 0x20768DD1561A00dc9cF9d3D3E64e51dEC93e8bB8 (100 ETH)
```

#### Private Keys

```
=====
(0) 0xd294bc56e6faa1ccf0ec9df385d4dce56b2d2528c3a743deeded7bff9d2a4c81
(1) 0x17351595dde947b997d49ba0b1827d91c6711e9eed3053ff3a6b10a3c8f77b99
(2) 0x57efafa3b0c26cc58eaa1c193184bc6ac4402b972a198f9f8fec653f9e504c7a
(3) 0x25efa9b37898ffd5b6f6dae6590b5daf40912fe80299ab69e3b1f3f33ed87d00
(4) 0xdec76e65213b7802ac74e88526ba184e4664268a3924df087cdb14ba8673a84
(5) 0xc76a8341609a12a0662b137c98f786d305c31fa5b440a0f98e916728fea4f594
(6) 0x847df3315969cfb7df11c9793812881c3d3ca8901ebae06023c36e6e577b1e08
(7) 0xbbfa57076cdbad7cb87d19c7819fbd4be1adcdc0e454e307a414fc18048a6c79
(8) 0xf75a11d0b597d4df35fc04f7e4e60f881f32203b72e777b88d3e986cc5fd9687
(9) 0xdfd81cda96335756e2e071c2cfe5f9e7200f291a02acf8272cd53ac7c4f2de82
```

#### HD Wallet

```
=====
Mnemonic:      since fever token stand there end forest chicken install embody segment fire
Base HD Path:  m/44'/60'/0'/0/{account_index}
```

#### Gas Price

```
=====
20000000000
```

#### Gas Limit

```
=====
6721975
```

#### Call Gas Limit

```
=====
9007199254740991
```

```
Listening on 127.0.0.1:8545
eth_blockNumber
```

## 4. Truffle console

Truffle console is an interactive console that connects you to any Ethereum client.

In another terminal, run the command `truffle console` which opens up a development console.

### Some useful commands

`accounts` - return a list of available accounts.

`compile` - compiles the contract and generates the ABI in JSON format.

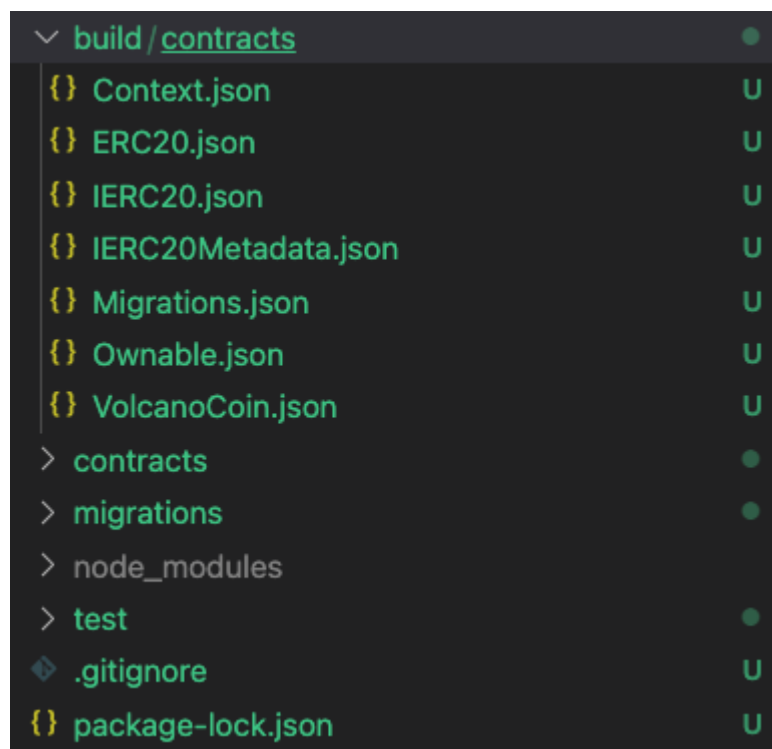
`migrate` - migrates the contract to the blockchain.

`migrate --reset` - Resets and runs all your migrations from the start.

## 5. Compile

Run `compile` in the truffle console. Every contract and inherited contract is compiled and the ABI (Application Binary Interface) is generated. You will see a new `build/contracts` folder has been generated, and each compiled contract has `.json` file containing the formatted ABI.

```
Compiling your contracts...
=====
> Compiling ./contracts/Migrations.sol
> Compiling ./contracts/VolcanoCoin.sol
> Compiling ./node_modules/@openzeppelin/contracts/access/Ownable.sol
> Compiling ./node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol
> Compiling ./node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol
> Compiling ./node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
> Compiling ./node_modules/@openzeppelin/contracts/utils/Context.sol
> Artifacts written to /Users/Kirsty/extropy/codeacademy/materials/Week 2 - introduction to Truffle/build/contracts
> Compiled successfully using:
   - solc: 0.8.4+commit.c7e474f2.Emscripten.clang
```



```

▼ build / contracts
  {} Context.json
  {} ERC20.json
  {} IERC20.json
  {} IERC20Metadata.json
  {} Migrations.json
  {} Ownable.json
  {} VolcanoCoin.json
  > contracts
  > migrations
  > node_modules
  > test
  ◆ .gitignore
  {} package-lock.json
```

## Recap: ABI (Application Binary Interface)

The ABI is the standard way to interact with smart contracts on the Ethereum network. It is a JSON which contains the names, inputs and outputs for the smart contract functions.

When a contract's function is called, the RPC node encodes the JSON data into Ethereum bytecode and broadcasts it to the network.

## Warning !

When the Solidity code is altered, the contract must be **re-compiled** to reflect the latest changes in the contract, then **re-deployed**.

## 6. Migrations

Go to `1_initial_migration.js` in the migrations folder. You would normally see this by default:

```
1  const Migrations = artifacts.require("Migrations");
2
3  module.exports = function (deployer) {
4      deployer.deploy(Migrations);
5  };
6
```

We have imported the VolcanoCoin contract and staged the contract for deployment for you. You should see that the file has been changed to this:

```
1  const Migrations = artifacts.require("Migrations");
2  const VolcanoCoin = artifacts.require("VolcanoCoin");
3
4  module.exports = async function (deployer) {
5      await deployer.deploy(Migrations);
6      await deployer.deploy(VolcanoCoin);
7  };
8
```

## Key explanations

artifacts

This is a JSON file that contains information regarding the contract such as the ABI, compiler version, contract bytecode.

Example:

```
1  {
2    "contractName": "Migrations",
3    "abi": [],
4    "bytecode": "0x...",
5    "sourceMap": "",
6    "deployedSourceMap": "",
7    "sourcePath": "/project/contracts/Migrations.sol",
8    "ast": {},
9    "legacyAST": {},
10   "compiler": {
11     "name": "solc",
12     "version": "0.8.0+commit.1d4f565a.Emscripten.clang"
13   },
14   "networks": {
15     "17": {
16       "events": {},
17       "links": {},
18       "address": "0x42a8d8ba55faAA734Ee07eD3179047169be5419e",
19       "transactionHash": "0x4e5fc578b9ea44401047fc010dd1ee20cd899b8091"
20     }
21   },
22   "schemaVersion": "3.0.0",
23   "updatedAt": "2021-07-11T17:59:54.615Z",
24   "devdoc": {},
25   "userdoc": {}
26 }
```

deployer and deploy

Built-in methods to deploy the contracts to the network.

## 7. Deploy

In the truffle development console, run `migrate` .

Starting migrations...

=====

```
> Network name:      'development'
> Network id:        1625841450743
> Block gas limit: 6721975 (0x6691b7)
```

1\_initial\_migration.js

=====

Deploying 'Migrations'

-----

```
> transaction hash:    0x29145a69717660320f9d242a0a6c73e68ef2b94e24cf4611e3e
> Blocks: 0           Seconds: 0
> contract address:    0x1E4DC90c851ee64edd5B2b0Cc00f94806d620304
> block number:        3
> block timestamp:     1626089841
> account:             0xE093F79F43800A94A8732c9F81B3ae3C93eAEd3
> balance:             99.98927406
> gas used:            246892 (0x3c46c)
> gas price:           20 gwei
> value sent:          0 ETH
> total cost:          0.00493784 ETH
```

Deploying 'VolcanoCoin'

-----

```
> transaction hash:    0xca291d3f8ac775cdd4df728b11045fe2dfd6d3ec317d2b8a38b
> Blocks: 0           Seconds: 0
> contract address:    0x8B1357c98f1316C4ECF965E160F240EbcB1d7503
> block number:        4
> block timestamp:     1626089842
> account:             0xE093F79F43800A94A8732c9F81B3ae3C93eAEd3
> balance:             99.95221212
> gas used:            1853097 (0x1c46a9)
> gas price:           20 gwei
> value sent:          0 ETH
> total cost:          0.03706194 ETH
```

> Saving migration to chain.

> Saving artifacts

-----

```
> Total cost:          0.04199978 ETH
```

Summary

=====

```
> Total deployments:   2
> Final cost:          0.04199978 ETH
```



## Common errors



- When the Solidity code is altered, the contract must be **re-compiled** to reflect the latest changes in the contract. If you need to re-compile your contract, you will also need to re-deploy it.
- Check that the URL in the configuration file is correct. For local host it is standard to be **127.0.0.1:8545** for the Ethereum network.
- If you have copied a contract from Remix to your Truffle project, ensure that you have the **exact** copy of the contract and the correct ABI. Even the smallest, insignificant difference (i.e additional line space, comments) can cause errors!
- In your migrations file, the name of the contract must be the **exact** same.
- Check the migrations contract - does it include the pragma version you're using?
- Are you having migration issues? Try resetting the migrations by running `migrate --reset` in the Truffle console, or `truffle migrate --reset` if you're outside of the console.

## Setting up locally

---

Before setting up, ensure that you have node and npm installed. Here's a useful guide - <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

(<https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>)

```
$ mkdir myProject
```

```
$ cd myProject
```

`$ npm init -y` to initialise a new npm project. If this does not work, try `npm init` and manually input your project information when prompted.

```
$ npm install -g --save truffle
```

In `truffle-config.js` , uncomment your development network settings.

In `Migrations.sol` , change the pragma version to include the version you're using.

## Testing with Truffle

---

Test files written in javascript or solidity can be added to the test directory and will run as a result of the **test** command from the console. The overall results of the test suite are listed in the console.

```
truffle(development)> test
Using network 'development'.

Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Contract: Volcano
  1) should mint 10000 VolcanoCoin
     > No events were emitted

0 passing (330ms)
1 failing

1) Contract: Volcano
   should mint 10000 VolcanoCoin:

    Minting Failed
    + expected - actual

    -10000
    +9999

    at Context.<anonymous> (test/volcano.test.js:7:12)
    at processTicksAndRejections (internal/process/task_queues.js:95:5)
```

the test starts with **it** , you can exclude tests by changing this to **xit** or set only one test to run with **it.only**

You will need to consider the atomicity of the tests, and decide what setup should be done. Before a contract section is run, truffle re deploys your contracts so that you have a clean starting point.

Within the contract section you can use `before` and `beforeEach`

## Testing for Transaction Errors and Events

---

A useful package 'truffle-assertions' allow us to test for transaction failure and events.

See : <https://www.npmjs.com/package/truffle-assertions> (<https://www.npmjs.com/package/truffle-assertions>)

### Testing a revert (or require) statement

It is useful to be able to test that the `require` and `revert` statements in our functions are working correctly.

For example if we have a function **registerDetails** which uses the `onlyOwner` modifier, we would expect it to revert if it is called from a different address.

We can test this with the following test

(You need to specify the exact error message that is returned from the `require` statement.)

```
it('Checks only owner can add an address', async () => {
  const hacker = '0xAE0F0e3bc47Aa699De97373822D825E4A3665698'
  await truffleAssert.reverts(
    RegistryContract.registerDetails('John Smith', { from: hacker }),
    'Ownable: caller is not the owner'
  )
})
```

Similarly we can test that events have been emitted with

```
truffleAssert.eventEmitted(result, 'TestEvent', { param1: 10, param2: 20 });
```

Here `param1` and `param2` are the values passed in the event.

There are a number of different options available, please see the documentation for those.

There are further libraries to help you test, for example the passage of time.

<https://docs.openzeppelin.com/test-helpers/0.5/> (<https://docs.openzeppelin.com/test-helpers/0.5/>)

## Workflow

A workflow that has worked well for us

- Explore / experiment in Remix - good for trying out syntax and quick checks on functionality
- Move to Truffle, write unit tests for TDD, and / or further develop the contracts
- Having a suite of tests gives confidence for refactoring
- Write the UI using mocks objects is necessary
- Use traditional techniques such as CI/CD.

## Useful Resources

<https://www.trufflesuite.com/docs/truffle/overview> (<https://www.trufflesuite.com/docs/truffle/overview>)

<https://github.com/trufflesuite/ganache-cli/blob/master/README.md>

(<https://github.com/trufflesuite/ganache-cli/blob/master/README.md>)

<https://www.trufflesuite.com/docs/truffle/testing/writing-tests-in-javascript>

(<https://www.trufflesuite.com/docs/truffle/testing/writing-tests-in-javascript>)

<https://docs.openzeppelin.com/test-helpers/0.5/> (<https://docs.openzeppelin.com/test-helpers/0.5/>)

<https://github.com/MolochVentures/moloch/tree/4e786db8a4aa3158287e0935dcdbc7b1e4341>

6e38/test#moloch-testing-guide

(<https://github.com/MolochVentures/moloch/tree/4e786db8a4aa3158287e0935dcbc7b1e43416e38/test#moloch-testing-guide>)