**Creating a project**

To use most Truffle commands, you need to run them against an existing Truffle project. So the first step is to create a Truffle project.

You can create a bare project template, but for those just getting started, you can use [Truffle Boxes](https://truffleframework.com/boxes), which are example applications and project templates. We'll use the [MetaCoin box](https://truffleframework.com/boxes/metacoin), which creates a token that can be transferred between accounts:

1. Create a new directory for your Truffle project:
2. mkdir MetaCoin
3. cd MetaCoin
4. Download ("unbox") the MetaCoin box:
5. truffle unbox metacoin

**Note**: You can use the truffle unbox <box-name> command to download any of the other Truffle Boxes.

**Note**: To create a bare Truffle project with no smart contracts included, use truffle init.

Once this operation is completed, you'll now have a project structure with the following items:

* contracts/: Directory for Solidity contracts
* migrations/: Directory for scriptable deployment files
* test/: Directory for test files for testing your application and contracts
* truffle.js: Truffle configuration file

**Exploring the project**

1. Open the contracts/MetaCoin.sol file in a text editor. This is a smart contract (written in Solidity) that creates a MetaCoin token. Note that this also references another Solidity file contracts/ConvertLib.sol in the same directory.
2. Open the contracts/Migrations.sol file. This is a separate Solidity file that manages and updates [the status of your deployed smart contract](https://truffleframework.com/docs/truffle/getting-started/running-migrations). This file comes with every Truffle project, and is usually not edited.
3. Open the migrations/1\_initial\_migration.js file. This file is the migration (deployment) script for the Migrations contract found in the Migrations.sol file.
4. Open the migrations/2\_deploy\_contracts.js file. This file is the migration script for the MetaCoin contract. (Migration scripts are run in order, so the file beginning with 2 will be run after the file beginning with 1.)
5. Open the test/TestMetacoin.sol file. This is a [test file written in Solidity](https://truffleframework.com/docs/truffle/testing/writing-tests-in-solidity) which ensures that your contract is working as expected.
6. Open the test/metacoin.js file. This is a [test file written in JavaScript](https://truffleframework.com/docs/truffle/testing/writing-tests-in-javascript) which performs a similar function to the Solidity test above.
7. Open the truffle.js file. This is the Truffle [configuration file](https://truffleframework.com/docs/truffle/reference/configuration), for setting network information and other project-related settings. The file is blank, but this is okay, as we'll be using a Truffle command that has some defaults built-in.

**Testing**

1. On a terminal, run the Solidity test:
2. truffle test ./test/TestMetacoin.sol

**Note**: If you're on Windows and encountering problems running this command, please see the documentation on [resolving naming conflicts on Windows](https://truffleframework.com/docs/truffle/reference/configuration#resolving-naming-conflicts-on-windows).

These tree tests were run against the contract, with descriptions displayed on what the tests are supposed to do.

1. Run the JavaScript test: truffle test ./test/metacoin.js

**Compiling**

1. Compile the smart contracts: truffle compile

**Migrating with Truffle Develop**

**Note**: To use [Ganache](https://truffleframework.com/ganache), please skip to the next section.

To deploy our smart contracts, we're going to need to connect to a blockchain. Truffle has a built-in personal blockchain that can be used for testing. This blockchain is local to your system and does not interact with the main Ethereum network.

You can create this blockchain and interact with it using [Truffle Develop](https://truffleframework.com/docs/truffle/getting-started/using-truffle-develop-and-the-console#truffle-develop).

1. Run Truffle Develop:
2. truffle develop

This shows ten accounts (and their private keys) that can be used when interacting with the blockchain.

1. On the Truffle Develop prompt, Truffle commands can be run by omitting the truffle prefix. For example, to run truffle compile on the prompt, type compile. The command to deploy your compiled contracts to the blockchain is truffle migrate, so at the prompt, type: migrate

**Alternative: Migrating with Ganache**

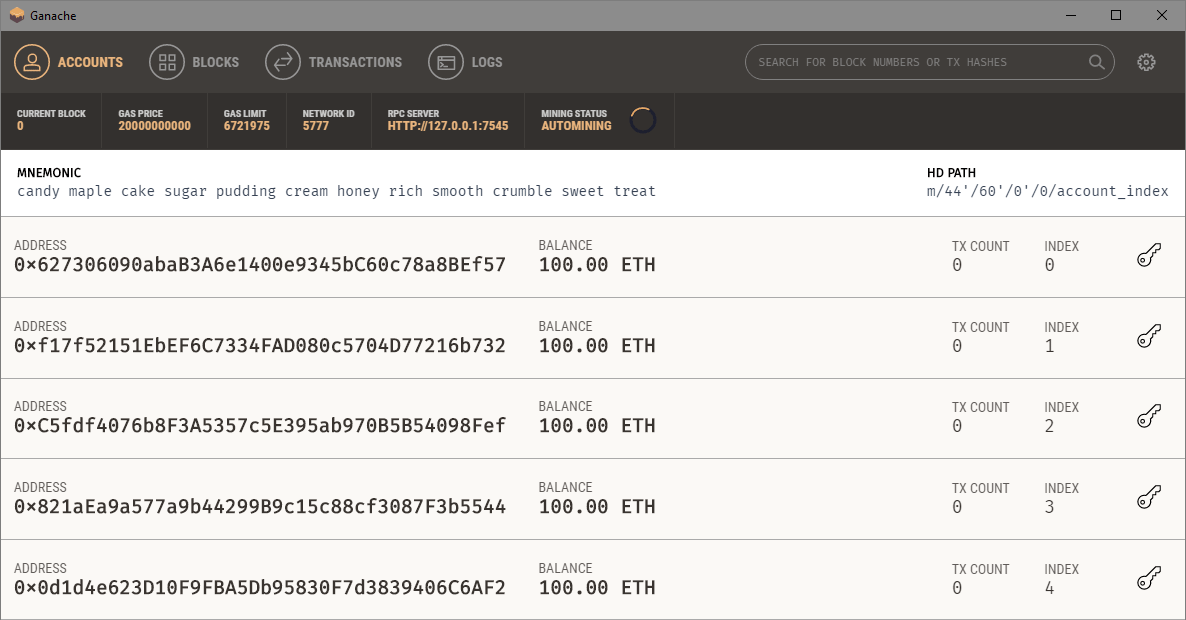
While Truffle Develop is an all-in-one personal blockchain and console, you can also use [Ganache](https://truffleframework.com/ganache), a desktop application, to launch your personal blockchain. Ganache can be a more easy-to-understand tool for those new to Ethereum and the blockchain, as it displays much more information up-front.

The only extra step, aside from running Ganache, is that it requires editing the Truffle configuration file to point to the Ganache instance.

1. Download and install [Ganache](https://truffleframework.com/ganache).
2. Open truffle.js in a text editor. Replace the content with the following:
3. module.exports = {
4. networks: {
5. development: {
6. host: "127.0.0.1",
7. port: 7545,
8. network\_id: "\*"
9. }
10. }
11. };

This will allow a connection using Ganache's default connection parameters.

1. Save and close that file.
2. Launch Ganache.



*Ganache*

1. On the terminal, migrate the contract to the blockchain created by Ganache:
2. truffle migrate
3. In Ganache, click the "Transactions" button to see that the transactions have been processed.
4. To interact with the contract, you can use the Truffle console. The Truffle console is similar to Truffle Develop, except it connects to an existing blockchain (in this case, the one generated by Ganache).
5. truffle console

You will see the following prompt:

truffle(development)>

**Interacting with the contract**

Interact with the contract using the console in the following ways:

**Note**: We're using web3.eth.getAccounts() in these examples, which returns a promise which resolves to an array of all the accounts generated by the mnemonic. So, given the addresses generated by our mnemonic above, specifying (await web3.eth.getAccounts())[0] is equivalent to the address 0x627306090abab3a6e1400e9345bc60c78a8bef57.

As of Truffle v5, the console supports async/await functions, enabling much simpler interactions with the contract.

* Begin by establishing both the deployed MetaCoin contract instance and the accounts created by either Truffle's built-in blockchain or Ganache:
* truffle(development)> let instance = await MetaCoin.deployed()
* truffle(development)> let accounts = await web3.eth.getAccounts()
* Check the metacoin balance of the account that deployed the contract:
* truffle(development)> let balance = await instance.getBalance(accounts[0])
* truffle(development)> balance.toNumber()
* See how much ether that balance is worth (and note that the contract defines a metacoin to be worth 2 ether):
* truffle(development)> let ether = await instance.getBalanceInEth(accounts[0])
* truffle(development)> ether.toNumber()
* Transfer some metacoin from one account to another:
* truffle(development)> instance.sendCoin(accounts[1], 500)
* Check the balance of the account that *received* the metacoin:
* truffle(development)> let received = await instance.getBalance(accounts[1])
* truffle(development)> received.toNumber()
* Check the balance of the account that *sent* the metacoin:
* truffle(development)> let newBalance = await instance.getBalance(accounts[0])
* truffle(development)> newBalance.toNumber()