**Brownie**

Brownie is a Python-based development and testing framework for smart contracts targeting the [Ethereum Virtual Machine](https://solidity.readthedocs.io/en/v0.6.0/introduction-to-smart-contracts.html#the-ethereum-virtual-machine).

**Note**

All code starting with $ is meant to be run on your terminal. Code starting with >>> is meant to run inside the Brownie console.

**Note**

This project relies heavily upon web3.py and the documentation assumes a basic familiarity with it. You may wish to view the [Web3.py docs](https://web3py.readthedocs.io/en/stable/index.html) if you have not used it previously.

**Features**

* Full support for [Solidity](https://github.com/ethereum/solidity) and [Vyper](https://github.com/vyperlang/vyper)
* Contract testing via [pytest](https://github.com/pytest-dev/pytest), including trace-based coverage evaluation
* Property-based and stateful testing via [hypothesis](https://github.com/HypothesisWorks/hypothesis/tree/master/hypothesis-python)
* Powerful debugging tools, including python-style tracebacks and custom error strings
* Built-in console for quick project interaction
* Support for [ethPM](https://www.ethpm.com/) packages

# Quickstart

This page provides a quick overview of how to use Brownie. It relies mostly on examples and assumes a level of familiarity with Python and smart contract development. For more in-depth content, you should read the documentation sections under “Getting Started” in the table of contents.

If you have any questions about how to use Brownie, feel free to ask on [Ethereum StackExchange](https://ethereum.stackexchange.com/) or join us on [Gitter](https://gitter.im/eth-brownie/community).

## Creating a New Project

Main article: [Creating a New Project](https://eth-brownie.readthedocs.io/en/v1.19.3/init.html#init)

The first step to using Brownie is to initialize a new project. To do this, create an empty folder and then type:

$ brownie init

You can also initialize “[Brownie mixes](https://github.com/brownie-mix)”, simple templates to build your project upon. For the examples in this document we will use the [token](https://github.com/brownie-mix/token-mix) mix, which is a very basic ERC-20 implementation:

$ brownie bake token

This will create a token/ subdirectory, and download the template project within it.

## Exploring the Project

Main article: [Structure of a Project](https://eth-brownie.readthedocs.io/en/v1.19.3/structure.html#structure)

Each Brownie project uses the following structure:

* contracts/: Contract sources
* interfaces/: Interface sources
* scripts/: Scripts for deployment and interaction
* tests/: Scripts for testing the project

The following directories are also created, and used internally by Brownie for managing the project. You should not edit or delete files within these folders.

* build/: Project data such as compiler artifacts and unit test results
* reports/: JSON report files for use in the GUI

## Compiling your Contracts

Main article: [Compiling Contracts](https://eth-brownie.readthedocs.io/en/v1.19.3/compile.html#compile)

To compile your project:

$ brownie compile

You will see the following output:

Brownie - Python development framework **for** Ethereum

Compiling contracts...

Optimizer: Enabled Runs: 200

- Token.sol...

- SafeMath.sol...

Brownie project has been compiled at token/build/contracts

You can change the compiler version and optimization settings by editting the [config file](https://eth-brownie.readthedocs.io/en/v1.19.3/config.html#config-solc).

**Note**

Brownie automatically compiles any new or changed source files each time it is loaded. You do not need to manually run the compiler.

## Core Functionality

The console is useful when you want to interact directly with contracts deployed on a non-local chain, or for quick testing as you develop. It’s also a great starting point to familiarize yourself with Brownie’s functionality.

The console feels very similar to a regular python interpreter. From inside a project directory, load it by typing:

$ brownie console

Brownie will compile your contracts, start the local RPC client, and give you a command prompt. From here you may interact with the network with the full range of functionality offered by the [Brownie API](https://eth-brownie.readthedocs.io/en/v1.19.3/api.html#api).

**Hint**

You can call the builtin **[dir](https://docs.python.org/3.8/library/functions.html" \l "dir" \o "(in Python v3.8))** method to see available methods and attributes for any class. Classes, methods and attributes are highlighted in different colors.

You can also call [**help**](https://docs.python.org/3.8/library/functions.html#help) on any class or method to view information on it’s functionality.

### Accounts

Main article: [Working with Accounts](https://eth-brownie.readthedocs.io/en/v1.19.3/core-accounts.html#core-accounts)

Access to local accounts is through [**accounts**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html#brownie.network.account.Accounts), a list-like object that contains [**Account**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html#brownie.network.account.Account) objects capable of making transactions.

Here is an example of checking a balance and transfering some ether:

**>>>** accounts[0]

<Account object '0xC0BcE0346d4d93e30008A1FE83a2Cf8CfB9Ed301'>

**>>>** accounts[1].balance()

100000000000000000000

**>>>** accounts[0].transfer(accounts[1], "10 ether")

Transaction sent: 0x124ba3f9f9e5a8c5e7e559390bebf8dfca998ef32130ddd114b7858f255f6369

Transaction confirmed - block: 1 gas spent: 21000

<Transaction object '0x124ba3f9f9e5a8c5e7e559390bebf8dfca998ef32130ddd114b7858f255f6369'>

**>>>** accounts[1].balance()

110000000000000000000

### Contracts

Main article: [Working with Contracts](https://eth-brownie.readthedocs.io/en/v1.19.3/core-contracts.html#core-contracts)

Brownie provides a **[ContractContainer](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html" \l "brownie.network.contract.ContractContainer" \o "brownie.network.contract.ContractContainer)** object for each deployable contract in your project. They are list-like objects used to deploy new contracts.

**>>>** Token

[]

**>>>** Token.deploy

<ContractConstructor object 'Token.constructor(string \_symbol, string \_name, uint256 \_decimals, uint256 \_totalSupply)'>

**>>>** t = Token.deploy("Test Token", "TST", 18, 1e21, {'from': accounts[1]})

Transaction sent: 0x2e3cab83342edda14141714ced002e1326ecd8cded4cd0cf14b2f037b690b976

Transaction confirmed - block: 1 gas spent: 594186

Contract deployed at: 0x5419710735c2D6c3e4db8F30EF2d361F70a4b380

<Token Contract object '0x5419710735c2D6c3e4db8F30EF2d361F70a4b380'>

**>>>** t

<Token Contract object '0x5419710735c2D6c3e4db8F30EF2d361F70a4b380'>

When a contact is deployed you are returned a [**Contract**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html#brownie.network.contract.ProjectContract) object that can be used to interact with it. This object is also added to the **[ContractContainer](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html" \l "brownie.network.contract.ContractContainer" \o "brownie.network.contract.ContractContainer)**.

[**Contract**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html#brownie.network.contract.ProjectContract) objects contain class methods for performing calls and transactions. In this example we are checking a token balance and transfering tokens:

**>>>** t

<Token Contract object '0x5419710735c2D6c3e4db8F30EF2d361F70a4b380'>

**>>>** t.balanceOf(accounts[1])

1000000000000000000000

**>>>** t.transfer

<ContractTx object 'transfer(address \_to, uint256 \_value)'>

**>>>** t.transfer(accounts[2], 1e20, {'from': accounts[1]})

Transaction sent: 0xcd98225a77409b8d81023a3a4be15832e763cd09c74ff431236bfc6d56a74532

Transaction confirmed - block: 2 gas spent: 51241

<Transaction object '0xcd98225a77409b8d81023a3a4be15832e763cd09c74ff431236bfc6d56a74532'>

**>>>** t.balanceOf(accounts[1])

900000000000000000000

**>>>** t.balanceOf(accounts[2])

100000000000000000000

When a contract source includes [NatSpec documentation](https://solidity.readthedocs.io/en/latest/natspec-format.html), you can view it via the [**ContractCall.info**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html#ContractCall.info) method:

**>>>** t.transfer.info()

transfer(address \_to, uint256 \_value)

@dev transfer token for a specified address

@param \_to The address to transfer to.

@param \_value The amount to be transferred.

### Transactions

Main article: [Inspecting and Debugging Transactions](https://eth-brownie.readthedocs.io/en/v1.19.3/core-transactions.html#core-transactions)

The **[TransactionReceipt](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html" \l "brownie.network.transaction.TransactionReceipt" \o "brownie.network.transaction.TransactionReceipt)** object contains all relevant information about a transaction, as well as various methods to aid in debugging.

**>>>** tx = Token[0].transfer(accounts[1], 1e18, {'from': accounts[0]})

Transaction sent: 0x0d96e8ceb555616fca79dd9d07971a9148295777bb767f9aa5b34ede483c9753

Token.transfer confirmed - block: 2 gas used: 51019 (33.78%)

**>>>** tx

<Transaction object '0x0d96e8ceb555616fca79dd9d07971a9148295777bb767f9aa5b34ede483c9753'>

Use **[TransactionReceipt.events](https://eth-brownie.readthedocs.io/en/v1.19.3/api-network.html" \l "TransactionReceipt.events" \o "TransactionReceipt.events)** to examine the events that fired:

**>>>** len(tx.events)

1

**>>>** 'Transfer' **in** tx.events

True

**>>>** tx.events['Transfer']

{

'from': "0x4fe357adbdb4c6c37164c54640851d6bff9296c8",

'to': "0xfae9bc8a468ee0d8c84ec00c8345377710e0f0bb",

'value': "1000000000000000000",

}

To inspect the transaction trace:

**>>>** tx.call\_trace()

Call trace for '0x0d96e8ceb555616fca79dd9d07971a9148295777bb767f9aa5b34ede483c9753':

Token.transfer 0:244 (0x4A32104371b05837F2A36dF6D850FA33A92a178D)

├─Token.transfer 72:226

├─SafeMath.sub 100:114

└─SafeMath.add 149:165

For information on why a transaction reverted:

**>>>** tx = Token[0].transfer(accounts[1], 1e18, {'from': accounts[3]})

Transaction sent: 0x5ff198f3a52250856f24792889b5251c120a9ecfb8d224549cb97c465c04262a

Token.transfer confirmed (reverted) - block: 2 gas used: 23858 (19.26%)

<Transaction object '0x5ff198f3a52250856f24792889b5251c120a9ecfb8d224549cb97c465c04262a'>

**>>>** tx.traceback()

Traceback for '0x5ff198f3a52250856f24792889b5251c120a9ecfb8d224549cb97c465c04262a':

Trace step 99, program counter 1699:

File "contracts/Token.sol", line 67, in Token.transfer:

balances[msg.sender] = balances[msg.sender].sub(\_value);

Trace step 110, program counter 1909:

File "contracts/SafeMath.sol", line 9, in SafeMath.sub:

require(b <= a);

## Writing Scripts

Main article: [Writing Scripts](https://eth-brownie.readthedocs.io/en/v1.19.3/interaction.html#scripts)

You can write scripts to automate contract deployment and interaction. By placing from brownie import \* at the beginning of your script, you can access objects identically to how you would in the console.

To execute the main function in a script, store it in the scripts/ folder and type:

$ brownie run [script name]

Within the token project, you will find an example script at [scripts/token.py](https://github.com/brownie-mix/token-mix/blob/master/scripts/token.py) that is used for deployment:

|  |  |
| --- | --- |
| 1  2  3  4 | **from** **brownie** **import** \*  **def** main():  Token.deploy("Test Token", "TEST", 18, 1e23, {'from': accounts[0]}) |

## Testing your Project

Main article: [Writing Unit Tests](https://eth-brownie.readthedocs.io/en/v1.19.3/tests-pytest-intro.html#pytest)

Brownie uses the pytest framework for contract testing.

Tests should be stored in the tests/ folder. To run the full suite:

$ brownie test

### Fixtures

Brownie provides pytest fixtures to allow you to interact with your project and to aid in testing. To use a fixture, add an argument with the same name to the inputs of your test function.

Here is an example test function using Brownie’s automatically generated fixtures:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **def** test\_transfer(Token, accounts):  token = Token.deploy("Test Token", "TST", 18, 1e20, {'from': accounts[0]})  **assert** token.totalSupply() == 1e20  token.transfer(accounts[1], 1e19, {'from': accounts[0]})  **assert** token.balanceOf(accounts[1]) == 1e19  **assert** token.balanceOf(accounts[0]) == 9e19 |

See the [Pytest Fixtures](https://eth-brownie.readthedocs.io/en/v1.19.3/tests-pytest-fixtures.html" \l "pytest-fixtures-reference) section for a complete list of fixtures.

### Handling Reverted Transactions

Transactions that revert raise a **[VirtualMachineError](https://eth-brownie.readthedocs.io/en/v1.19.3/api-brownie.html" \l "brownie.exceptions.VirtualMachineError" \o "brownie.exceptions.VirtualMachineError)** exception. To write assertions around this you can use **[brownie.reverts](https://eth-brownie.readthedocs.io/en/v1.19.3/api-test.html" \l "brownie.test.plugin.RevertContextManager" \o "brownie.test.plugin.RevertContextManager)** as a context manager, which functions very similarly to **[pytest.raises](https://docs.pytest.org/en/latest/reference/reference.html" \l "pytest.raises" \o "(in pytest v0.1.dev463+g5dbfb8e))**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **import** **brownie**  **def** test\_transfer\_reverts(accounts, Token):  token = accounts[0].deploy(Token, "Test Token", "TST", 18, 1e23)  **with** brownie.reverts():  token.transfer(accounts[1], 1e24, {'from': accounts[0]}) |

You may optionally include a string as an argument. If given, the error string returned by the transaction must match it in order for the test to pass.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **import** **brownie**  **def** test\_transfer\_reverts(accounts, Token):  token = accounts[0].deploy(Token, "Test Token", "TST", 18, 1e23)  **with** brownie.reverts("Insufficient Balance"):  token.transfer(accounts[1], 1e24, {'from': accounts[0]}) |

### Isolating Tests

Test isolation is handled through the **[module\_isolation](https://eth-brownie.readthedocs.io/en/v1.19.3/api-test.html" \l "fixtures.module_isolation" \o "fixtures.module_isolation)** and **[fn\_isolation](https://eth-brownie.readthedocs.io/en/v1.19.3/api-test.html" \l "fixtures.fn_isolation" \o "fixtures.fn_isolation)** fixtures:

* [**module\_isolation**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-test.html#fixtures.module_isolation) resets the local chain before and after completion of the module, ensuring a clean environment for this module and that the results of it will not affect subsequent modules.
* [**fn\_isolation**](https://eth-brownie.readthedocs.io/en/v1.19.3/api-test.html#fixtures.fn_isolation) additionally takes a snapshot of the chain before running each test, and reverts to it when the test completes. This allows you to define a common state for each test, reducing repetitive transactions.

This example uses isolation and a shared setup fixture. Because the token fixture uses a session scope, the transaction to deploy the contract is only executed once.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | **import** **pytest**  **from** **brownie** **import** accounts  **@pytest**.fixture(scope="module")  **def** token(Token):  **yield** Token.deploy("Test Token", "TST", 18, 1e20, {'from': accounts[0]})  **def** test\_transferFrom(fn\_isolation, token):  token.approve(accounts[1], 6e18, {'from': accounts[0]})  token.transferFrom(accounts[0], accounts[2], 5e18, {'from': accounts[1]})  **assert** token.balanceOf(accounts[2]) == 5e18  **assert** token.balanceOf(accounts[0]) == 9.5e19  **assert** token.allowance(accounts[0], accounts[1]) == 1e18  **def** test\_balance\_allowance(fn\_isolation, token):  **assert** token.balanceOf(accounts[0]) == 1e20  **assert** token.allowance(accounts[0], accounts[1]) == 0 |