Bitcoin Test Framework

vinteum's study hour

qlrd - Floresta/Krux contributor

Bitcoin Test Framework

Definition 1.

The functional tests [that] test the RPCs. [1]

RPC [2]

Each **remote procedure call** has two sides: an active client side that makes the call to a server side, which sends back a reply. (…) The caller first sends a call message to the server process and waits (blocks) for a reply message. The call message includes the procedure's parameters, and the reply message includes the procedure's results. Once the reply message is received, the results of the procedure are extracted, and the caller's execution is resumed.

Bitcoin Test Framework

Definition 2.

The functional test frameworks uses a version of `python-bitcoinrpc` which can be found here [3]. This library allows the test framework to call RPC commands as if they were python functions. [1]

RPC example with `bitcoin-cli`

```
# mannually add a node
$> bitcoin-cli -regtest addnode 127.0.0.1:38334 add true
```

RPC example with `curl`

```
# mannually add a v2transport enabled node
$> curl --user youruser:yourpass \
   --data-binary
'{"jsonrpc":"1.0","id":"curltest","method":"addnode","params":
["127.0.0.1:38334","add", true]}' \
   -H 'content-type: text/plain;' \
   http://127.0.0.1:18443/
```

RPC example with python

```python
class NetTest(BitcoinTestFramework):

  ...

  def run_test(self):
      # We need miniwallet to make a transaction
      self.wallet = MiniWallet(self.nodes[0])

      # By default, the test framework sets up an addnode
      # connection from node 1 --> node0. By connecting
      # node0 --> node 1, we're left with
      # the two nodes being connected both ways.
      # Topology will look like: node0 <--> node1
      self.connect_nodes(0, 1)
      self.sync_all()
```

What is a functional test?

Definition 3.

Functional test in general is defined as a test that tests functionalities or features of software from a user’s perspective (...) [4]

Consequence

(…) it takes pretty long. In general they take longer than unit tests. (…) pay some attention to how you write the tests and how many you write. [4]

The test framework

Definition 4.

This is a collection of files that have helpful functionalities that help you to write a test. [4]

The `test/funcional/test_framework/test_framework.py`

This is the most important file that you are going to use in every test. This implements the BitcoinTestFramework class and every test is a subclass of the BitcoinTestFramework class [4]

When do you add/edit functional tests?

It is not something where you would add or edit a functional test if you implement something new. It is when you don’t really add something new, when you do a refactoring, you don’t really change any functionality that the user sees or that the user would notice [4]

Where are the files?

https://github.com/bitcoin/bitcoin > test > functional

Which are the file types?

(…) you find files that are following a naming scheme [4]:

- example:
  - `test/functional/example_test.py`;

Which are the file types?

- features (assumevalid, assumeutxo, etc…):
  - `test/functional/feature_*.py`;
- interface (cli, rest, zmq, …):
  - `test/functional/interface_*.py`;
- mempool:
  - `test/functional/mempool_*.py`;

Which are the file types?

- mining:
  - `test/functional/mining_*.py`;
- p2p:
  - `test/functional/p2p_*.py`;
- rpc (getblockcaininfo, getblock, deriveaddresses, etc...):
  - `test/functional/rpc_*.py`;
- wallet :
  - `test/functional/wallet_*.py`;

Which are the file types?

There is also a single test that has the prefix `tool_` [4]

Which are the file types?

- tools (signet miner, utxo to sqlite, etc…):
    - `test/functional/tool_*.py`;

Running functional tests [4]

- (…) you can run these tests directly like any other Python file by specifying the path.
- (…) you can also run the tests through test harness and that is very helpful if you want to run all the tests in one time, the full functional test suite.
- (…) you want to do some pattern matching on the names. You want to run all the wallet tests.

Run these tests directly [4]

Tests come with shebang (`#!/usr/bin/env python3`)

```
$> test/functional/feature_rbf.py
```

Run the tests through test harness [4]

```
$> test/functional/test_runner.py feature_rbf.py
```

Run the tests through test harness [4]

Run only the tests that starts with the `wallet_` prefix:

```
$> test/functional/test_runner.py test/functional/wallet\*
```

BitcoinTestFramework class

```python
class BitcoinTestMetaClass(type):
    """Metaclass for BitcoinTestFramework.

    Ensures that any attempt to register a subclass of
    `BitcoinTestFramework` adheres to a standard whereby the
    subclass overrides `set_test_params` and `run_test` but DOES
    NOT override either `__init__` or `main`. If any of those
    standards are violated, a ``TypeError`` is raised."""
```

BitcoinTestFramework class

```
class BitcoinTestFramework(metaclass=BitcoinTestMetaClass):
    """Base class for a bitcoin test script.

    Individual bitcoin test scripts should subclass this class
    and override the set_test_params() and run_test() methods.

    Individual tests can also override the following methods to
    customize the test setup:

    - add_options()
    - setup_chain()
    - setup_network()
    - setup_nodes()

    The __init__() and main() methods should not be overridden.

    This class also contains various public and private helper
methods."""
```

Documentation and logs [4]

You will see:

- documentation and logs in the test;
- docstrings at the beginning of every class and every important function;
- see comments and you will also see these `self.log.info()` outputs.

A new test class [4]

The two functions that are going to see overriding almost every test is one setting the `test_params()` by overriding `set_test_params()`.

Node calls

In every test what you are going to see is calls on the nodes:

Node calls: connections

You will typically have an array of nodes on self and then you will refer to these nodes by just giving them a number but you can also alias them if you want:

```
self.nodes[0].add_p2p_connection(BaseNode())
```

Node calls: mining and control [5]

These are going to be `regtest` nodes so you can use `regtest` RPC commands like:

```
block_hash = self.generate(self.nodes[0], 1, sync_fun=self.no_op)
[0]
block = self.nodes[0].getblock(blockhash=block_hash, verbosity=0)
for n in self.nodes:
    n.submitblock(block)
    chain_info = n.getblockchaininfo()
    assert_equal(chain_info["blocks"], 200)
    assert_equal(chain_info["initialblockdownload"], False)
```

P2P introspection [4]

Oftentimes you will have a node where you are testing something on but you want to make sure that first of all the network is synced up to that node or your node has to sync up to the network. Or just the block has been sent, stuff like that.

P2P introspection: synchronization [4]

Often you will see functions that are doing a wait for you until everything is synced up. They are going to fail the test if they don't:

- `sync_all()`;
- `sync_blocks()`.

P2P introspection: hooks [4]

You can also go deeper and subclass the `P2PInterface` class and redefine hooks on this [6].

P2P introspection: hooks [4]

```python
def on_message(self, message):
    """Receive message and dispatch message to appropriate
    callback.

    We keep a count of how many of each message type has been
    received and the most recent message of each type."""
    with p2p_lock:
        try:
            msgtype = message.msgtype.decode('ascii')
            self.message_count[msgtype] += 1
            self.last_message[msgtype] = message
            getattr(self, 'on_' + msgtype)(message)
        except Exception:
            print(
              "ERROR delivering %s (%s)" %
              (repr(message), sys.exc_info()[0])
            )
            raise
```

P2P introspection: hooks [4]

```
def on_open(self):
    pass

def on_close(self):
    pass

def on_addr(self, message): pass
def on_addrv2(self, message): pass
def on_block(self, message): pass
def on_blocktxn(self, message): pass
def on_cfcheckpt(self, message): pass
def on_cfheaders(self, message): pass
def on_cfilter(self, message): pass
def on_cmpctblock(self, message): pass
def on_feefilter(self, message): pass
```

Example [4]

`test/functional/rpc_blockchain.py`:

- docstring which is describing what this is actually testing, a bunch of RPCs.
- imports. It is very important we don't do any wildcard imports.
- subclass of BitcoinTestFramework and this is overriding the `set_test_params` function.
- the `run_test` which is the actual test.

Derivations of Bitcoin Test Framework

- Warnet: Run scenarios of network behavior across the network which can be programmed using the Bitcoin Core functional test_framework language.
- Floresta test framework: integration tests for Floresta.

Bibliography

Bibliography

[1] A. Chown, "How does bitcoin functional test framework work?." [Online]. Available: https://bitcoin.stackexchange.com/questions/73932/how-does-bitcoin-functional-test-framework-work

[2] N. W. Group, "RPC: Remote Procedure Call Protocol Specification Version 2." [Online]. Available: https://www.rfc-editor.org/rfc/rfc5531

[3] jgarzik, "Python interface to bitcoin's JSON-RPC API ." [Online]. Available: https://github.com/jgarzik/python-bitcoinrpc

[4] M. F. Fabian Jahr Bryan Bishop, "Bitcoin Core Functional Test Framework." [Online]. Available: https://btctranscripts.com/edgedevplusplus/2019/bitcoin-core-functional-test-framework

[5] T. B. C. developers, "test framework: setup_nodes method." [Online]. Available: https://github.com/bitcoin/bitcoin/blob/master/test/functional/test_framework/test_framework.py#L480

Bibliography (ii)

[6] T. B. C. developers, “test framework: P2PInteface class.” [Online]. Available: https://github.com/bitcoin/bitcoin/blob/master/test/functional/test_framework/p2p.py#L448