Bitcoin Test Framework

vinteum's study hour

qlrd - Floresta/Krux contributor

Introduction

Bitcoin Test Framework

Could someone explain how and where in the code the bitcoin test framework calls map to C++ bitcoin core code or RPC calls such that regression suite tests the bitcoin core functionality? [1]

Bitcoin Test Framework

Definition 1.

The functional tests test the RPCs. [1]

Bitcoin Test Framework

Definition 2.

The unit tests test the C++ code directly by calling the functions. [1]

Bitcoin Test Framework

Definition 3.

The functional test frameworks uses a version of `python-bitcoinrpc` which can be found here [2]. This library allows the test framework to call RPC commands as if they were python functions. [1]

Back to the basics

What is a functional test?

Functional test in general is defined as a test that tests functionalities or features of software from a user’s perspective. [3]

And in Bitcoin-Core’s user perspective, what is a “functionality” or a “feature”?

(...) thinking about it is is that nodes that you interact with in the network are also users who are using their own node. You have to look at features from your own perspective but also from the network's perspective. [3]

And what means “(…) to look at features from your own perspective but also from the network’s perspective”?

The perspective

It tests the full stack. [3]

Consequence of full stack perspective

(…) it takes pretty long. In general they take longer than unit tests. [3]

Why they take so long?

(…) pay some attention to how you write the tests and how many you write. Oftentimes I see people deferring to writing functional tests because they are Python tests. That is often easier to write for some people. A hint to keep in mind

Example

Example

From sogoagain's gist [4]

- Generate 1 block and test block count and immature balance;
- generate 100 more blocks and test block count and balance;
- sync nodes and test best block hash;
- test unspent transaction outputs;
- send 1 BTC from node0 to node1;
- sync nodes and test mempool;
- generate a block with node2 and test block count;
- sync nodes and test best block hash;
- test nodes' balances.

```python
from test_framework.util import (
  assert_equal,
  assert_approx
)
from test_framework.test_framework import BitcoinTestFramework
```

```python
class SimpleFunctionalTest(BitcoinTestFramework):

    def add_options(self, parser):
        self.add_wallet_options(parser)

    def set_test_params(self):
        self.setup_clean_chain = True
        self.num_nodes = 3

    def skip_test_if_missing_module(self):
        self.skip_if_no_wallet()

    def setup_network(self):
        self.setup_nodes()
        self.connect_nodes(0, 1)
        self.connect_nodes(0, 2)
        self.sync_all(self.nodes[0:2])
```

```python
def run_test(self):
      # Generate 1 block and test block count and immature balance
      blocks = self.generate(
        self.nodes[0],
        nblocks=1,
        sync_fun=self.no_op
      )
      assert_equal(self.nodes[0].getblockcount(), 1)

      walletinfo_node0 = self.nodes[0].getwalletinfo()
      assert_equal(walletinfo_node0['immature_balance'], 50)
      assert_equal(walletinfo_node0['balance'], 0)
```

```python
# Generate 100 more blocks and test block count and balance
blocks = self.generate(
  self.nodes[0],
  nblocks=100,
  sync_fun=self.no_op
)
assert_equal(self.nodes[0].getblockcount(), 101)
assert_equal(self.nodes[0].getbalance(), 50)
```

```python
# Sync nodes and test best block hash
self.sync_blocks(self.nodes[0:2])
assert_equal(self.nodes[1].getbestblockhash(), blocks[99])
assert_equal(self.nodes[2].getbestblockhash(), blocks[99])
```

```python
# Test unspent transaction outputs
utxos = self.nodes[0].listunspent()
assert_equal(len(utxos), 1)
assert_equal(utxos[0]['confirmations'], 101)
```

```python
# Send 1 BTC from node0 to node1
txid = self.nodes[0].sendtoaddress(
  address=self.nodes[1].getnewaddress(),
  amount=1
)
```

```python
# Sync nodes and test mempool
self.sync_mempools(self.nodes[0:2])
assert txid in self.nodes[1].getrawmempool()
assert txid in self.nodes[2].getrawmempool()
```

```python
# Generate a block with node2 and test block count
blocks = self.generate(
  self.nodes[2],
  nblocks=1,
  sync_fun=self.no_op
)
assert_equal(self.nodes[2].getblockcount(), 102)
```

```
# Sync nodes and test best block hash
self.sync_blocks(self.nodes[0:2])
assert_equal(self.nodes[0].getbestblockhash(), blocks[0])
assert_equal(self.nodes[1].getbestblockhash(), blocks[0])
```

```
# Test nodes' balances
assert_approx(self.nodes[0].getbalance(), 99, 0.001)
assert_equal(self.nodes[1].getbalance(), 1)
assert_equal(self.nodes[2].getbalance(), 0)
```

When do you add/edit functional tests?

When do you add/edit functional tests?

Usually when you want to test for features. [3]

When do you add/edit functional tests?

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. [3]

Where are the files?

Where are the files?

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

Where are the files?

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

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

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

[4] sogoagain, “Simple functional test example for Bitcoin Core.” [Online]. Available: https://gist.github.com/sogoagain/d645de960aa4959219e83f928889d64b