# BTC TestNet

## Setting up the environment

* **Install docker runtime environment**

sudo apt-get install docker.io

* **Create a soft link**

sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker

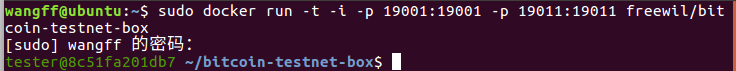
* **Download BTC TestNet to run the image**

sudo docker pull freewil/bitcoin-testnet-box

* **Run TestNet**

The docker image will run two bitcoin nodes in the background and is meant to be attached to allow you to type in commands. The image also exposes the two JSON-RPC ports from the nodes if you want to be able to access them from outside the container.

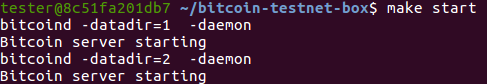
sudo docker run -t -i -p 19001:19001 -p 19011:19011 freewil/bitcoin-testnet-box



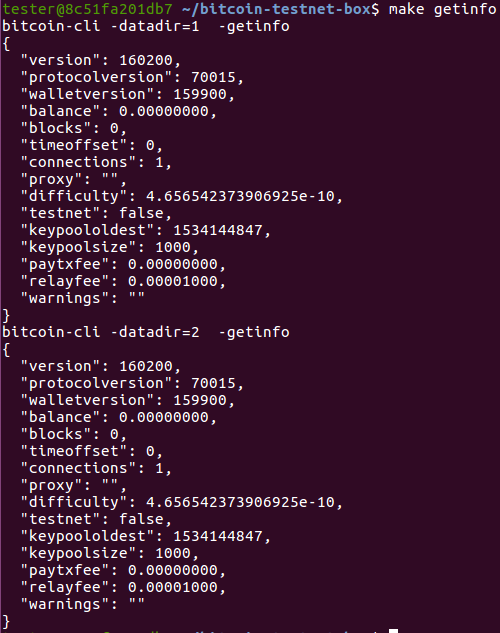
The 19001 and 19011 in the above commands are ports configured to provide RPC services to two nodes.

* **Start BTC**

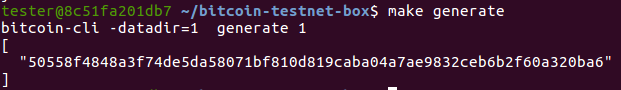
make start After the startup is successful, two bitcoin test wallet nodes will be run in the native simulation to form a private range of bitcoin test network.



make getinfo View test network node status information



make generate Simulate a new block record



make generate BLOCKS=200 Simulate new generation of 200 block records

make sendfrom1 ADDRESS=mkiytxYA6kxUC8iTnzLPgMfCphnz91zRfZ AMOUNT=10 Transfer 10 BTC to the specified test wallet address



make generate BLOCKS=6 After the transfer is complete, you need to enter the command in the docker console to have 6 confirmations for the transaction.

make stop Out of service

make clean clear data

## Node test

Open a new terminal.

* **Install the RPC support library for node.js**

There are several RPC support libraries for commonly used Node.js. We use the open source project kapitalize here. (https://github.com/shamoons/Kapitalize)

npm install kapitalize

* **Modify the code**

Since testnet is relatively new, some APIs have been renamed or added, and kapitalize project code has not been updated for at least two years, so manual modification is required.

node\_modules/kapitalize/lib/commands.js



* **RpcTestnet.js**



* **Query.js**



* **Sending a transaction**

Transfer 0.001 BTCs automatically from mkiytxYA6kxUC8iTnzLPgMfCphnz91zRfZ to mieC38pnPwMqbMAN6sGWwHRQ3msp7nRnNz.

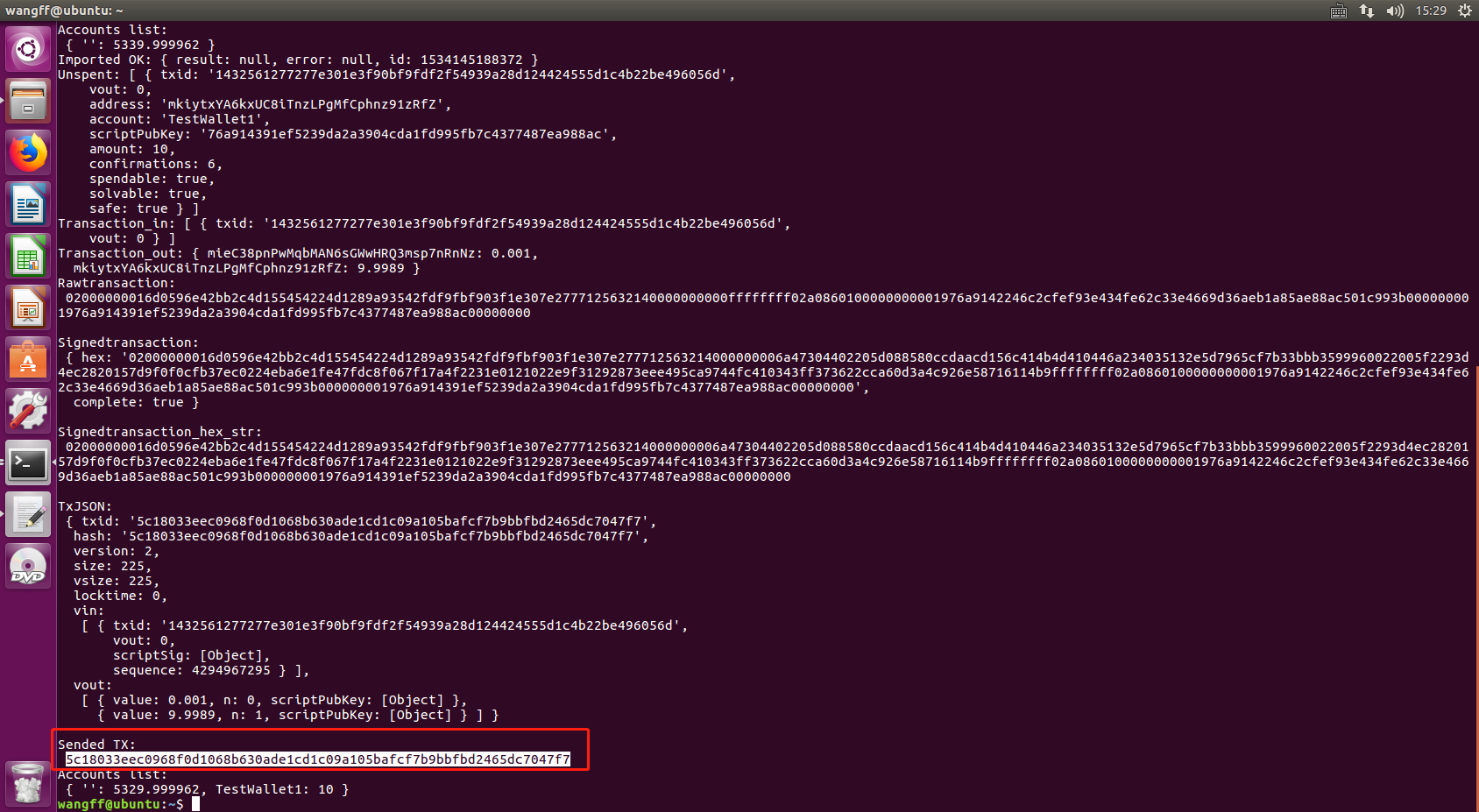
node RpcTestnet.js

After the transfer is complete, enter the command in the docker console to have 6 confirmations

make generate BLOCKS=6

* **Query the corresponding block of the transaction**

After getting the hash of the transaction, modify Query.js





Save and execute,

node Query.js

# Seele TestNet

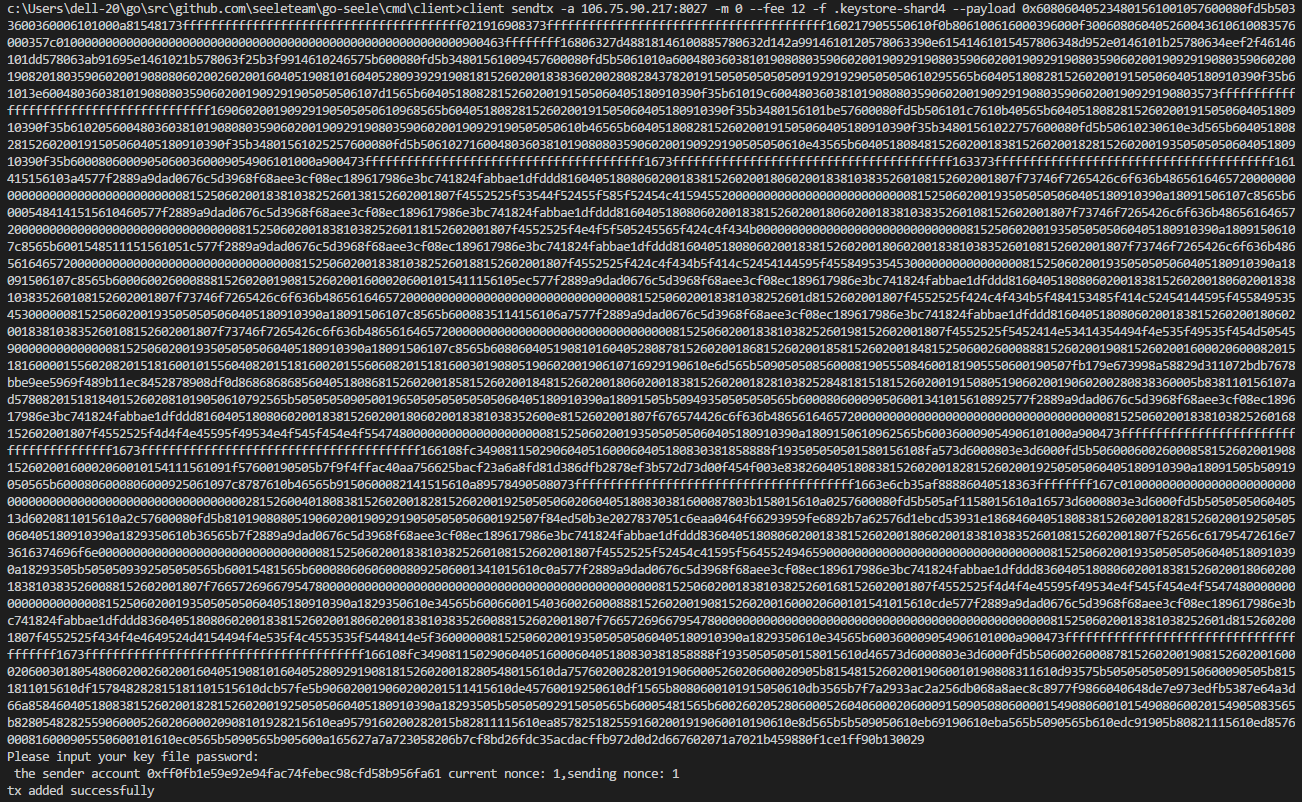
## Deployment contract

* Write btc-relay

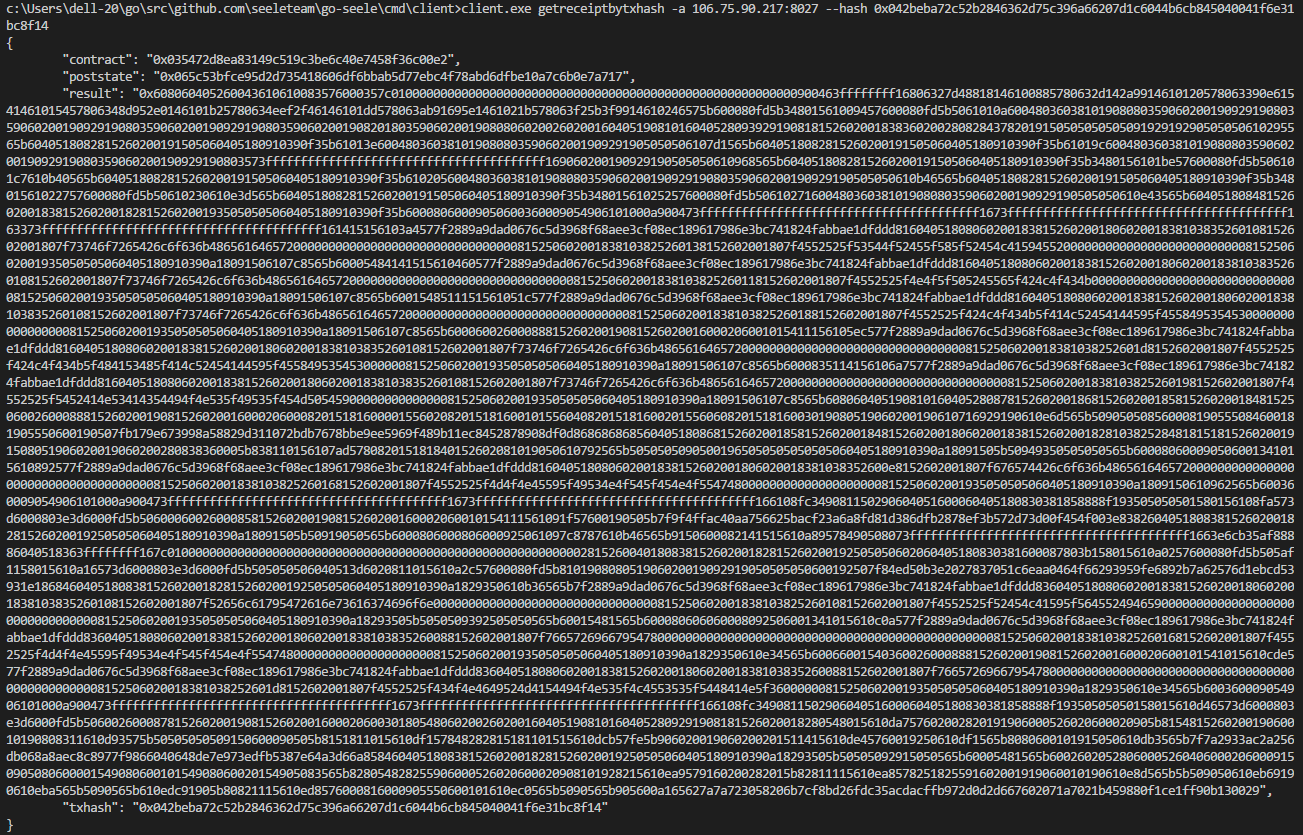


* Deploy to Seele TestNet

Account address : 0xff0fb1e59e92e94fac74febec98cfd58b956fa61



Get the contract address based on the transaction hash.

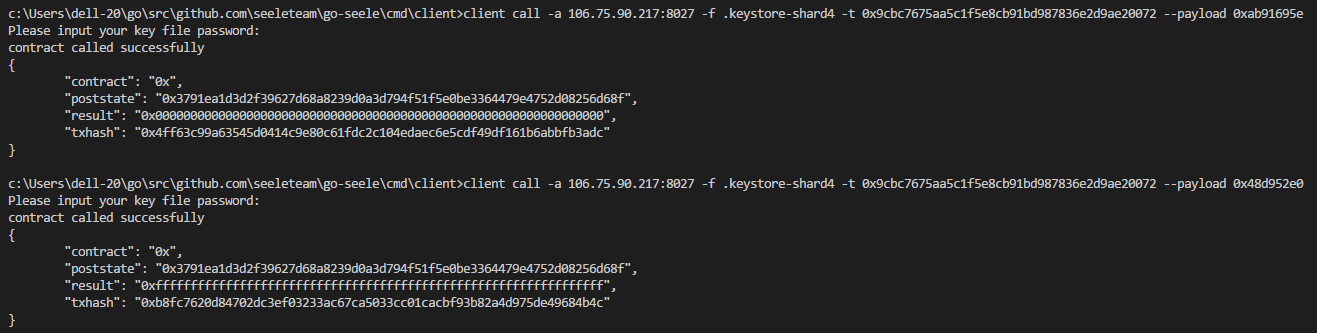


## Test contract

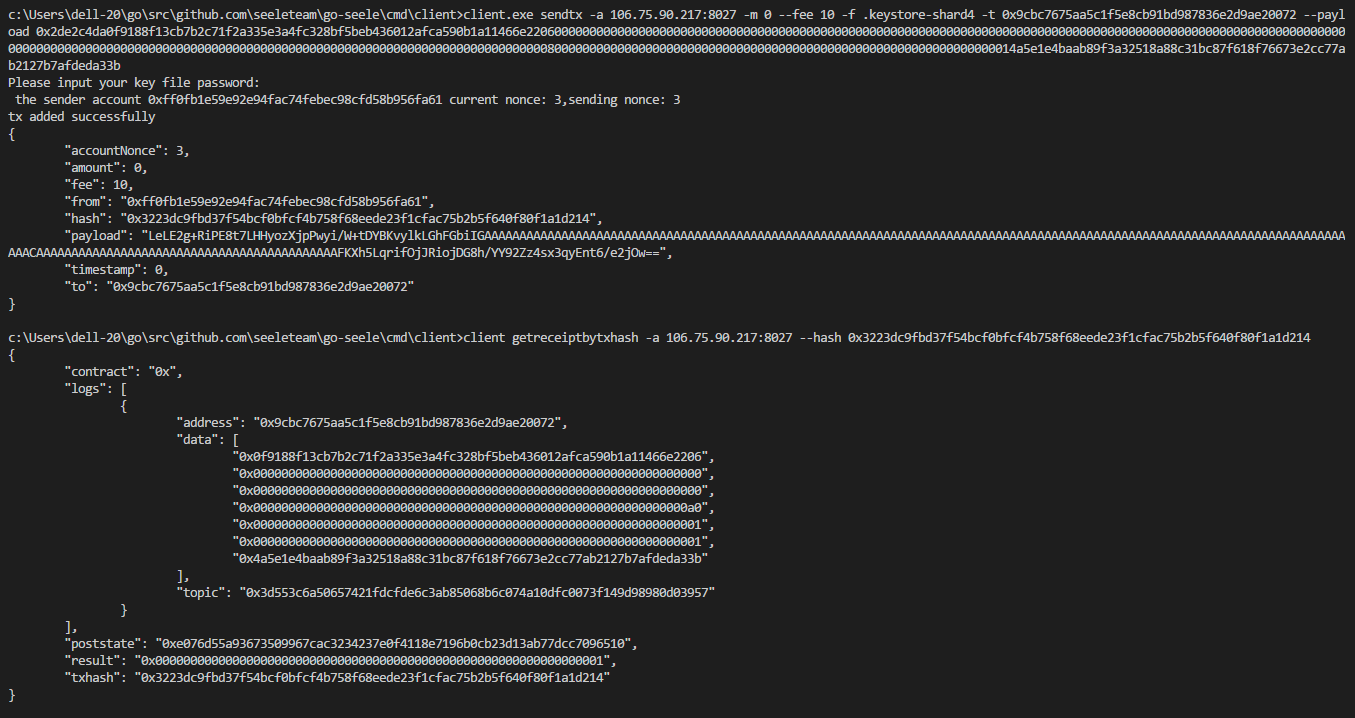
* **Test PreBlockHash and PreBlockHeight**

PreBlockHash Bytecode : 0xab91695e

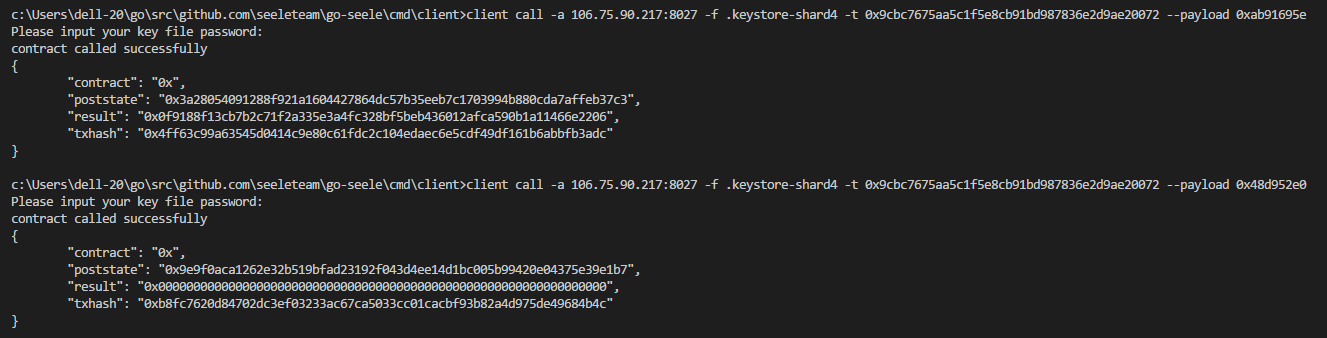
PreBlockHeight Bytecode : 0x48d952e0



As you can see, the initial hash is 0 and the initial height is -1. Next, the storeBlockHeader function is called to store a block. ***blockHash, height, previousblockhash, txs :***

"0x0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206", 0, 0, ["0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"]

Stored successfully, then call to see PreBlockHash and PreBlockHeight:



As you can see, PreBlockHash and PreBlockHeight become the block header information we store.

* **Test storeBlockHeader**

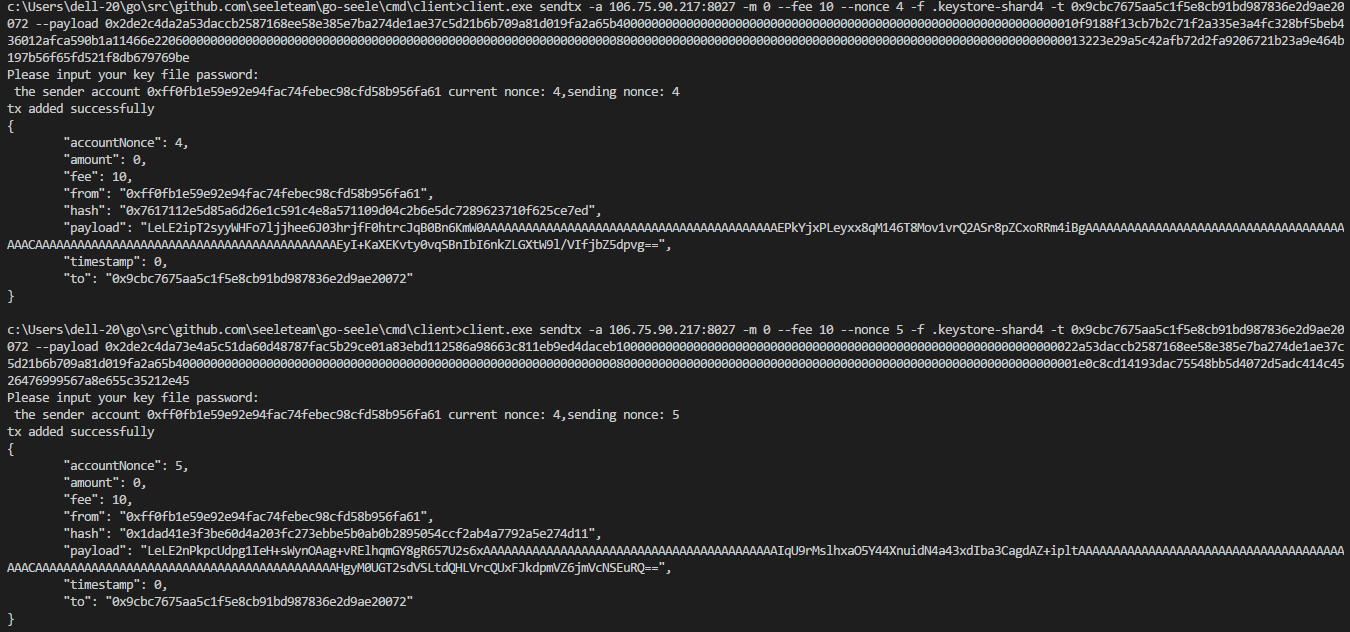
This function is used to store block information. There are five check logics, one store logic and one store result event. Please see the code. Since the creation of the creation block was previously stored, this test directly stores the next block. Note that the storage is not hop-blockable, non-repeatable storage, and cannot be stored in an untrusted account address.

***blockHash, height, previousblockhash, txs :***

"0x2a53daccb2587168ee58e385e7ba274de1ae37c5d21b6b709a81d019fa2a65b4", 1, "0x0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206", ["0x3223e29a5c42afb72d2fa9206721b23a9e464b197b56f65fd521f8db679769be"]

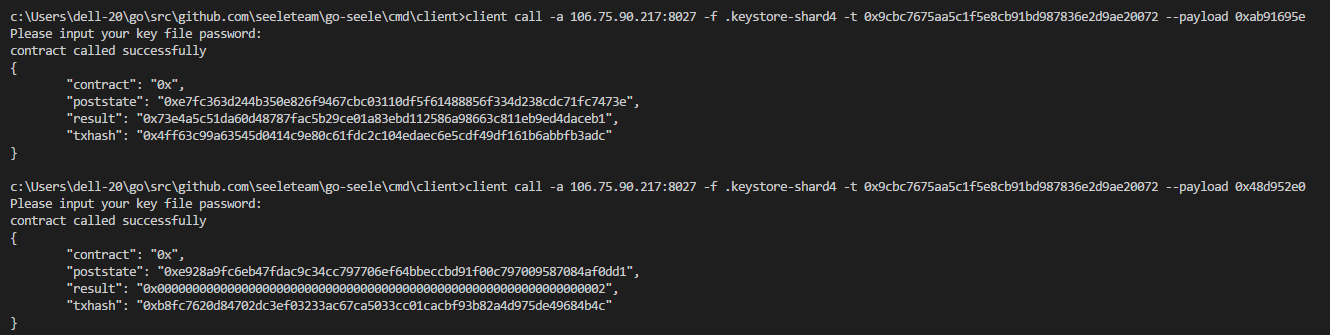
***blockHash, height, previousblockhash, txs :***

"0x73e4a5c51da60d48787fac5b29ce01a83ebd112586a98663c811eb9ed4daceb1", 2, "0x2a53daccb2587168ee58e385e7ba274de1ae37c5d21b6b709a81d019fa2a65b4", ["0xe0c8cd14193dac75548bb5d4072d5adc414c4526476999567a8e655c35212e45"]



Looking at PreBlockHash and PreBlockHeight, the expected result should be the hash and height of block two:

0x73e4a5c51da60d48787fac5b29ce01a83ebd112586a98663c811eb9ed4daceb1，2



In line with expectations, the test was successful.

* **Test getBlockHeader**

***Remarks: On this contract, all inquiries are those who need to pay the commission to the deployment contract (the initial consideration is to motivate someone to storeBlockHeader to make it autonomous, which can be discarded at this stage)***

***blockHash:***

"0x2a53daccb2587168ee58e385e7ba274de1ae37c5d21b6b709a81d019fa2a65b4"

Expected results: 1

The return value of this function is successful if and only if it is one. Same as below.



In line with expectations, the test was successful.

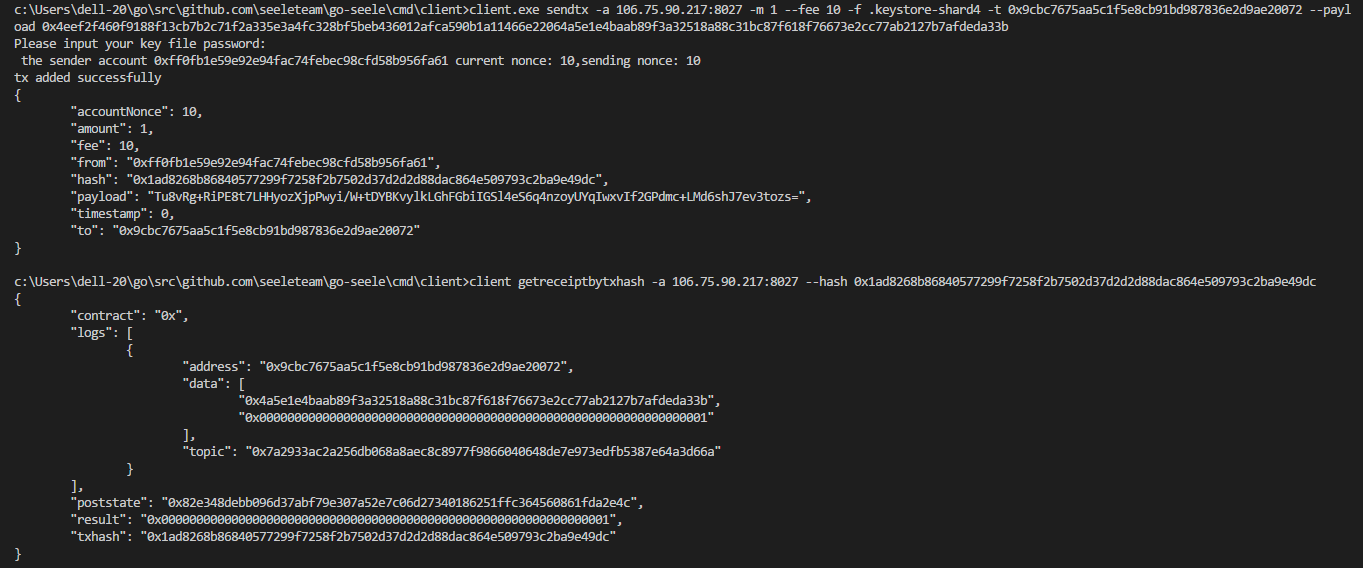
* **Test verifyTx**

This method verifies that there is this transaction on the block.

***blockHash, tx:***

"0x0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206", "0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b"

Expected results: 1

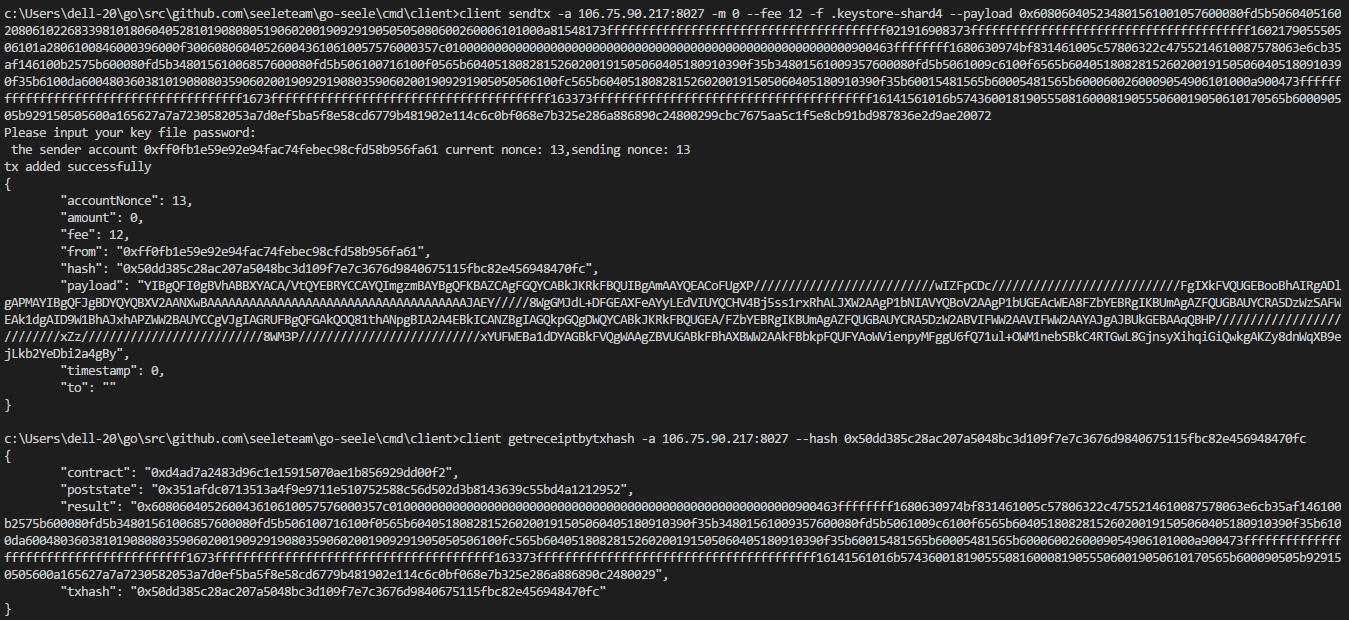


In line with expectations, the test was successful.

* **Test relayTx**

This method can forward the transaction to another contract, provided that another contract must inherit the processTransaction method that implements BitcoinProcessor.

First deploy this contract to the test network, the contract source code in the BTC-RELAY contract.

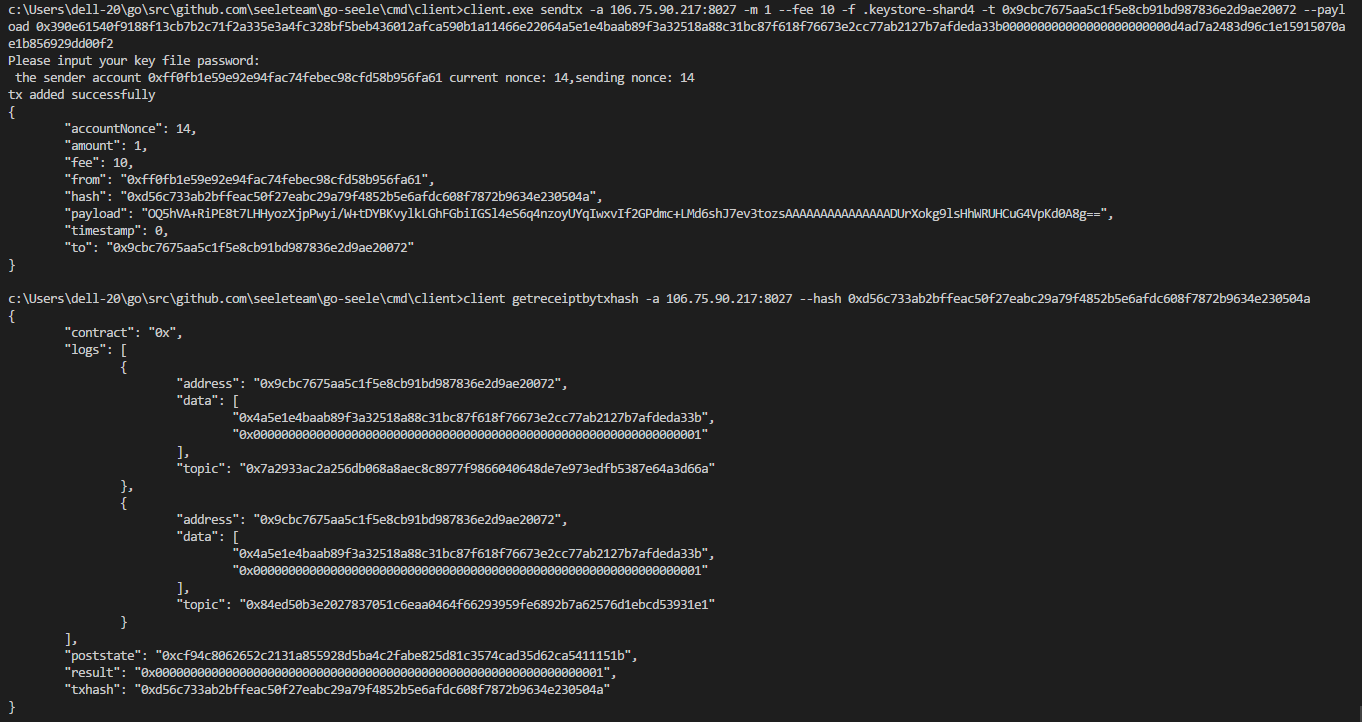


After the deployment is complete, get the address of this contract.

***blockHash, tx, contract:***

"0x0f9188f13cb7b2c71f2a335e3a4fc328bf5beb436012afca590b1a11466e2206", "0x4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b", "0xd4ad7a2483d96c1e15915070ae1b856929dd00f2"

Expected results: 1



In line with expectations, the test was successful.