**CSED 490U Blockchain & Cryptocurrency**

**Assignment 3**

****

**Submitted by- Sajan Maharjan**

**POVIS id-** [**thesajan@postech.ac.kr**](mailto:thesajan@postech.ac.kr)

**Registration Number- 20182095**

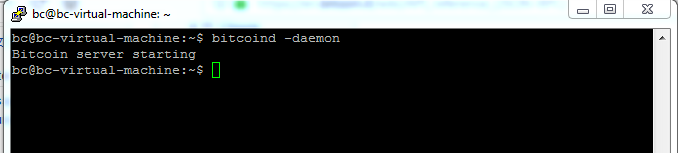
**<Step 1> Install Bitcoin Core Client**

Bitcoin PPA repository was added to the server, updated and bitcoind was installed using the commands-

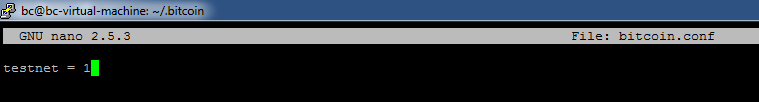
|  |
| --- |
| *sudo apt-add-repository ppa:bitcoin/bitcoin*  *sudo apt-get update*  *sudo apt-get install bitcoind* |

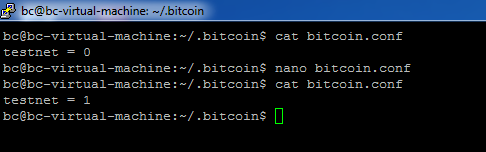
Starting the bitcoin server

|  |
| --- |
| *bitcoind -daemon* |



Editing the bitcoin.conf file to use the testnet server by setting the value testnet=1





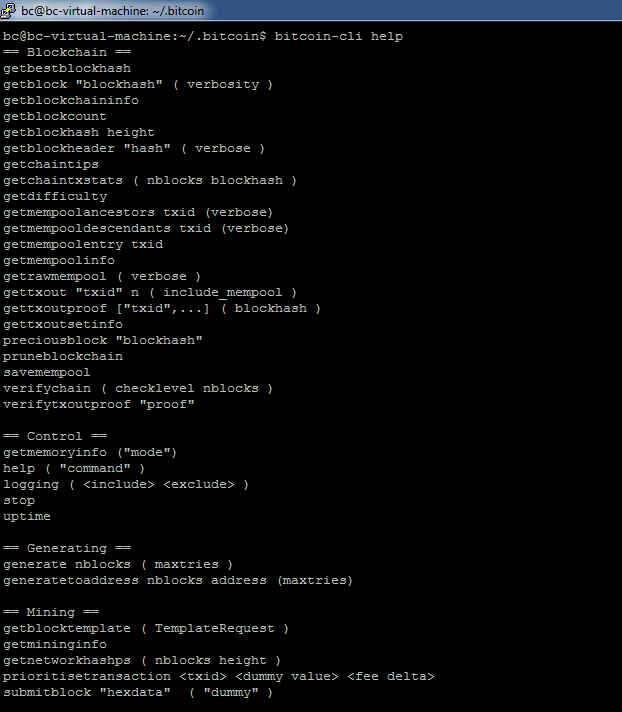
Now, it is possible to execute bitcoin command line without adding the –testnet option.

**<Step. 2> Check out the JSON-RPC APIs**

The command-

|  |
| --- |
| *bitcoin-cli help* |

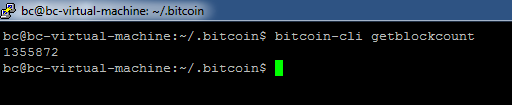
lists all the commands that can be executed in the bitcoin server using command line



For example, we can use the command

|  |
| --- |
| *bitcoin-cli getblockcount* |

to list the total number of blocks in the bitcoin network (in our case the test network)



|  |
| --- |
| *bitcoin-cli getmininginfo* |

Outputs a JSON object listing the values for different attributes of mining like current no. of blocks, difficulty setting, chain (in our case, testnet)



Also, the bitcoin core client comes with a pre-installed wallet. We can get the information regarding this wallet using the command-

|  |
| --- |
| *bitcoin-cli getwalletinfo* |

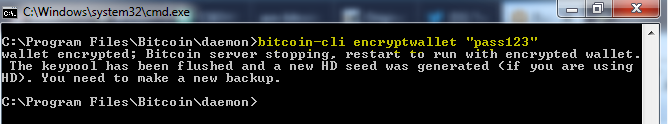


**<Step. 3> Set a password on the wallet / Unlock the password on the wallet.**

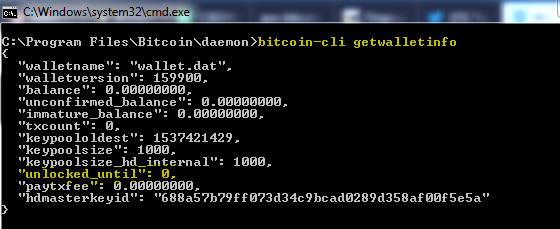
As seen from the above snapshot, the inbuilt wallet is unencrypted (we know this because the JSON object returned by *getwalletinfo*, doesn’t contain the field- *unlocked\_until*).

A wallet can be encrypted using the command- encryptwallet followed the the passphrase with which to encrypt the wallet. For example-

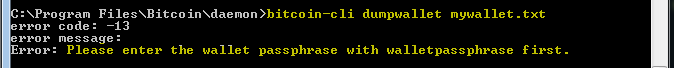
|  |
| --- |
| bitcoin-cli encryptwallet “pass123” |



Setting up encryption in your wallet requires a restart of the Bitcoin (testnet) server. After restarting the server, we can see that on executing the *bitcoin-cli getwalletinfo* command, an additional field *unlocked\_until* is now added into the JSON object.

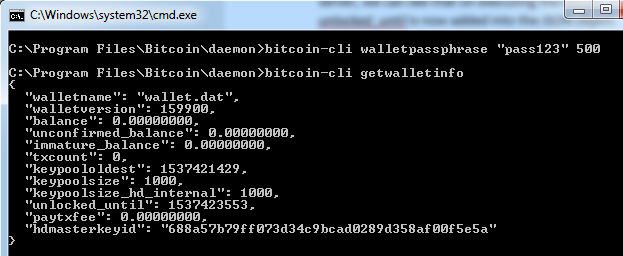


When the wallet is encrypted in this way, you are unable to perform operations concerning wallet such as dump the wallet as text file, dump the private key of the wallet, etc are all disabled as shown in the figure below-



Now, in order to continue operations on the wallet, you should unlock the wallet using the command- *bitcoin-cli walletpassphrase* followed by two parameters- first is the passphrase that was used to encrypt the wallet and the second parameter is the no. of seconds to unlock the wallet before it gets locked again.

|  |
| --- |
| *bitcoin-cli walletpassphrase “pass123” 500* |

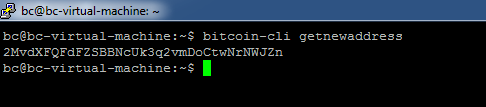


The above snapshot shows that the wallet was successfully unlocked using the command *walletpassphrase.* We can also see additional info by executing the command *getwalletinfo* that the wallet is now unlocked until a specified UNIX Epoch timestamp (no. of seconds since 1st January 1970).

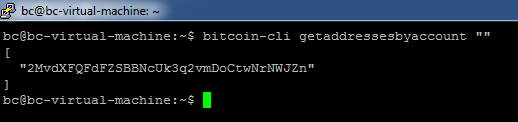
**<Step. 4> Generate an address / Get a Bitcoin / Check the Balance**

Addresses in bitcoin enable users to send/receive bitcoins. Addresses can be created in bitcoin using the command- *getnewaddress*

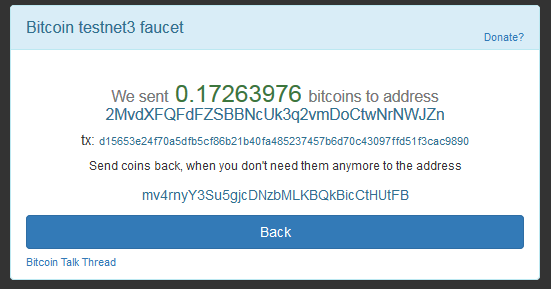
|  |
| --- |
| *bitcoin-cli getnewaddress* |



We can confirm that the address has been created for the wallet using the command- *getaddressesbyaccount* which lists all the addresses contained in the wallet i.e.

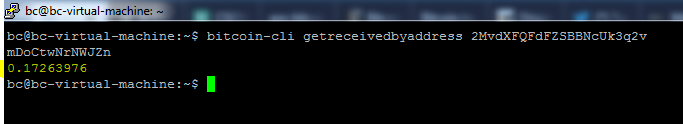


The above generated address- **2MvdXFQFdFZSBBNcUk3q2vmDoCtwNrNWJZn** was generated within the test network of bitcoin and we can get dummy bitcoins for such testnet address. The URL <https://coinfaucet.eu/en/btc-testnet/> was used to get some bitcoins for the above address as shown below



We can check the bitcoin balance available in a bitcoin address using the command- *getreceivedbyaddress* followed the address we want to check the balance for.

|  |
| --- |
| *bitcoin-cli getreceivedbyaddress 2MvdXFQFdFZSBBNcUk3q2vmDoCtwNrNWJZn* |



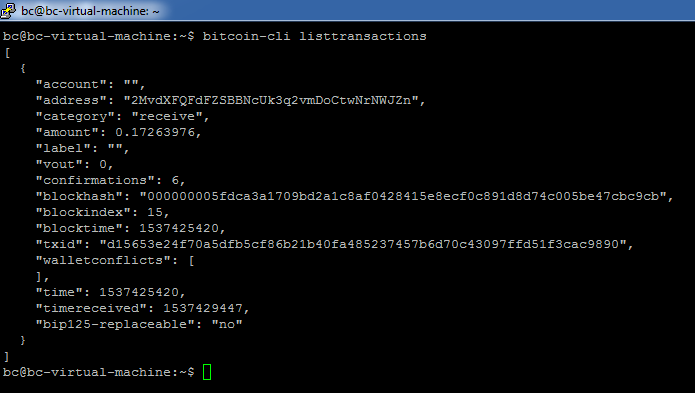
In this way, we can get bitcoins inside the test network and then check the balance in a given address.

**<Step. 5> Check the reception of BTC, examine transaction contents / investigate more detailed transaction contents**

**5-1) Verify the transaction related to BTC reception and then check the balance. You can get information such as txid, amount and recipient’s address.**

In order to view transaction and its details, we can use the commands- *listtransactions* and *gettransaction*. The *listtransactions* will output an array of JSON object for each transaction. i.e.

|  |
| --- |
| *bitcoin-cli listtransactions* |



We can see from the snapshot above that the bitcoin sent to us in the previous step is being shown here with the correct address and amount. This command will list all the transactions applicable to all the addresses in the wallet.

**5-2) Examine the transaction found in 5-1 using transaction id**

One can also view a single transaction by using the command *gettransaction* followed by the transaction id in hex code

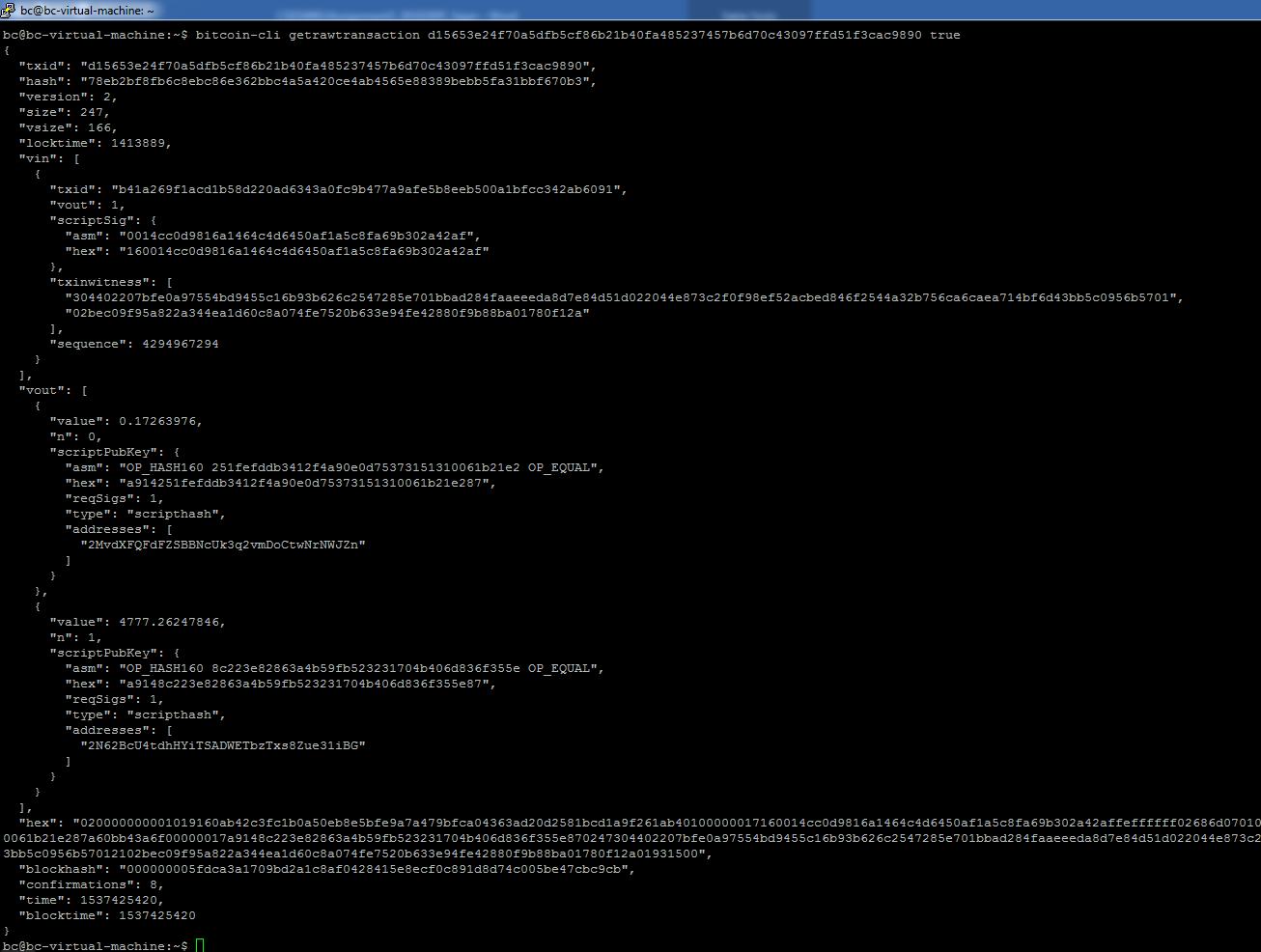
|  |
| --- |
| *bitcoin-cli gettransaction d15653e24f70a5dfb5cf86b21b40fa485237457b6d70c43097ffd51f3cac9890* |



**5-3) The results of 5-2 are simple. Print out the actual details of the transactions**

If we want to investigate into further details of the transaction, we can do so using the commands- *getrawtransaction* followed by the value true i.e.

|  |
| --- |
| *bitcoin-cli getrawtransaction d15653e24f70a5dfb5cf86b21b40fa485237457b6d70c43097ffd51f3cac9890 true* |



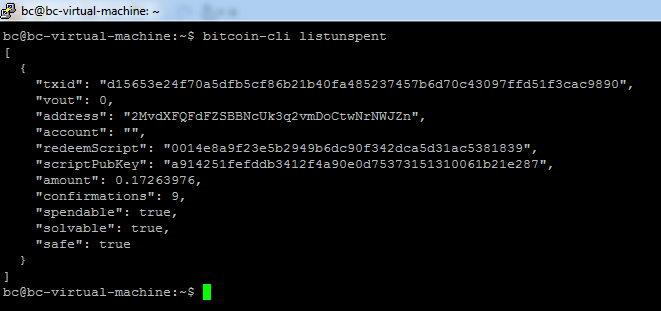
Using this command, we view the details about transactions such as- input UTXO consumed, no. of inputs and outputs, previous transaction referenced, change value created, locking script encumbered to the transaction output as such.

**<Step. 6> Create a transaction using UTXO / Sign the transaction / Send the transaction**

**6-1) Determine unspent and confirmed outputs to use as an input to a transaction**

The confirmed and unspent outputs of the transactions can be obtained using the command- *listunspent*

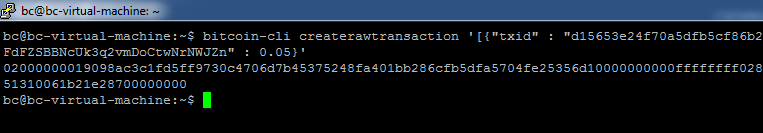
|  |
| --- |
| *bitcoin-cli listunspent* |



**6-2) Choose one of UTXOs gained from 6-1 and then create a transaction.**

Transactions in bitcoin are created using the *createrawtransaction* command which takes the unspent transaction output address as its input and two transaction outputs- bitcoin sent to the new address and change sent to the previous address i.e.

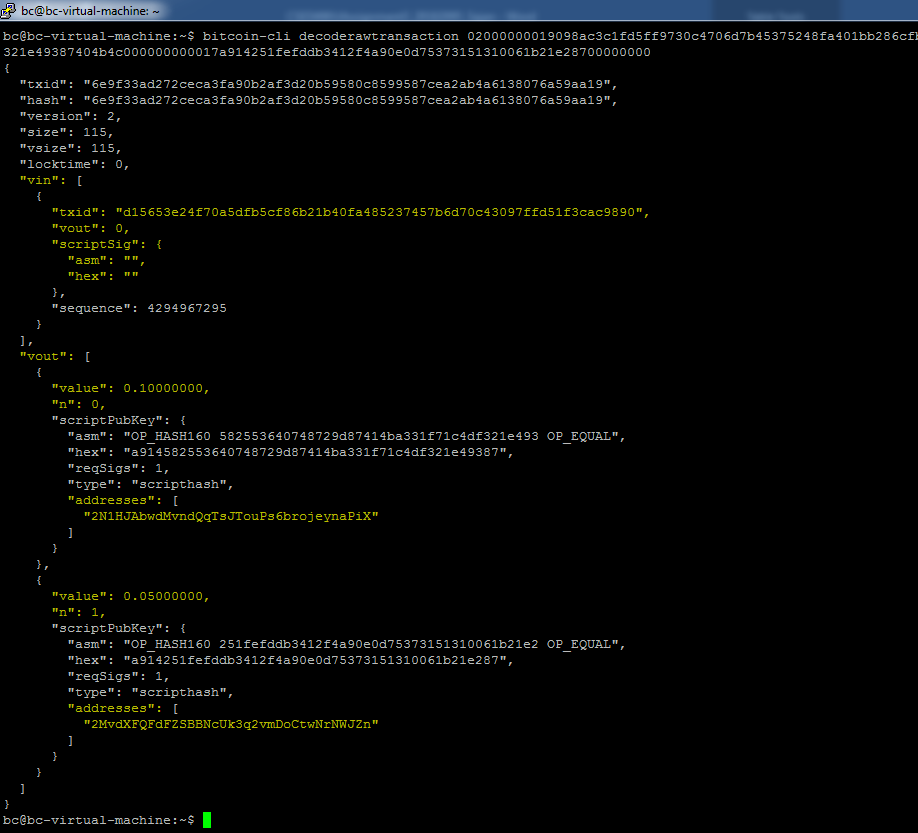
|  |
| --- |
| *bitcoin-cli createrawtransaction ‘[{“txid” : “d15653e24f70a5dfb5cf86b21b40fa485237457b6d70c43097ffd51f3cac9890”, “vout” : 0}]’ ‘{“2N1HJAbwdMvndQqTsJTouPs6brojeynaPiX” : 0.1, “2MvdXFQFdFZSBBNcUk3q2vmDoCtwNrNWJZn” : 0.05}’* |



The first argument to the createrawtransaction is the ith output produced by the previous transaction id and the second argument is the list of addresses with the values of bitcoin to send. The output of the transaction is the raw hex string of the the transaction created.

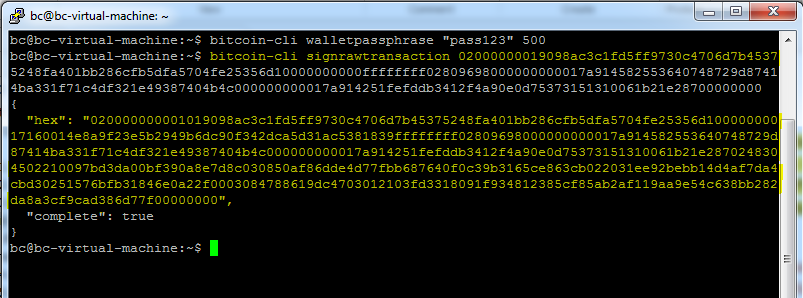
**6-3) Decode rawtransaction with hex string generated in 6-2. You can see that the 'scriptsig' field is empty. You can see that the scriptSig field is empty**

|  |
| --- |
| *bitcoin-cli decoderawtransaction 02000000019098ac3c1fd5ff9730c4706d7b45375248fa401bb286cfb5dfa5704fe25356d10000000000ffffffff02809698000000000017a914582553640748729d87414ba331f71c4df321e49387404b4c000000000017a914251fefddb3412f4a90e0d75373151310061b21e28700000000* |



**6-4) Sign the transaction so that the ‘scriptSig’ field can be filled. As a result of signing, you will obtain a hex string**

|  |
| --- |
| *bitcoin-cli signrawtransaction* **HEX\_STRING** |



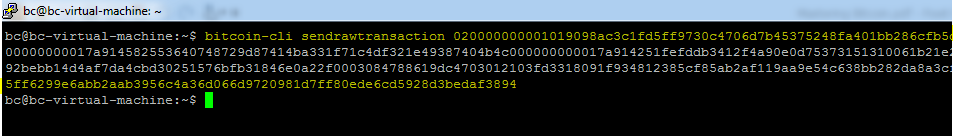
**6-5) Check if ‘scriptsig’ field of the transaction is filled or not. If you have performed everything correctly, you will be able to see the filled ‘scriptsig’ field after decoding the hex string**

|  |
| --- |
| *bitcoin-cli decoderawtransaction* **HEX\_CODE\_OBTAINED\_FROM\_6-4** |



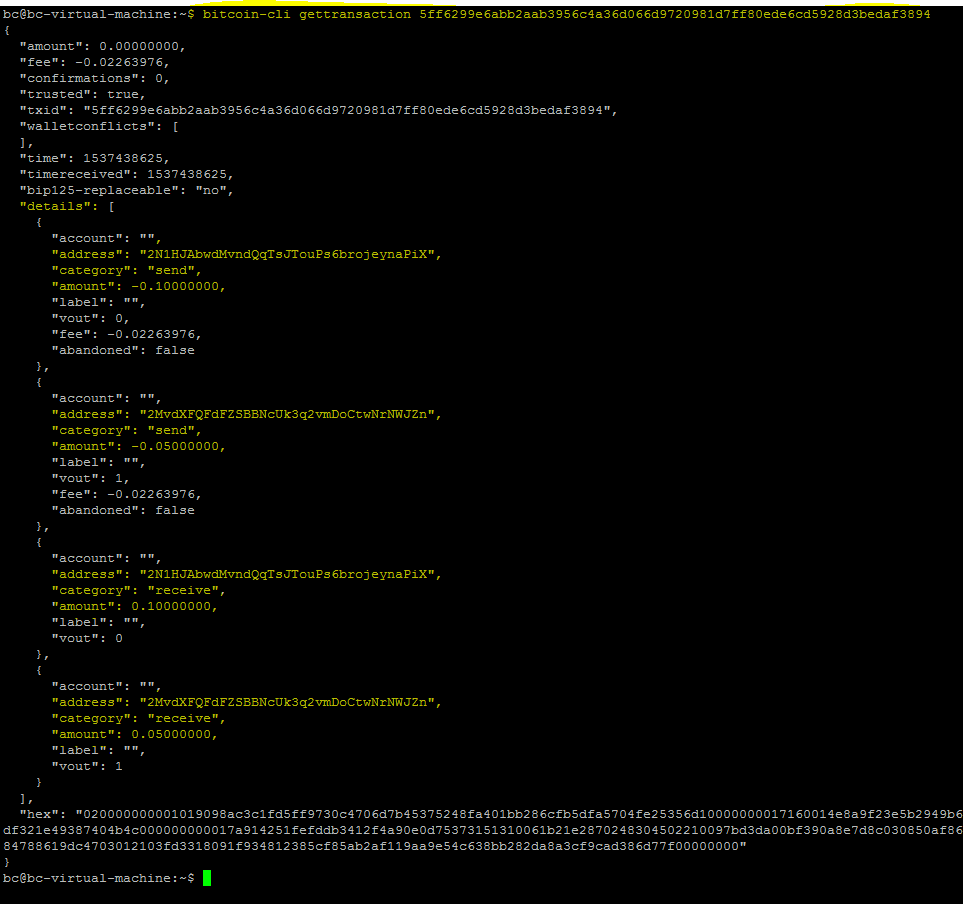
**6-6) Send the transaction to the bitcoin network**

|  |
| --- |
| *bitcoin-cli sendrawtransaction* **HEX\_STRING** |



**6-7) Examine the contents of the transaction using tx\_id obtained from 6-6**

|  |
| --- |
| *bitcoin-cli gettransaction* **TXN\_ID\_FROM\_6-6** |

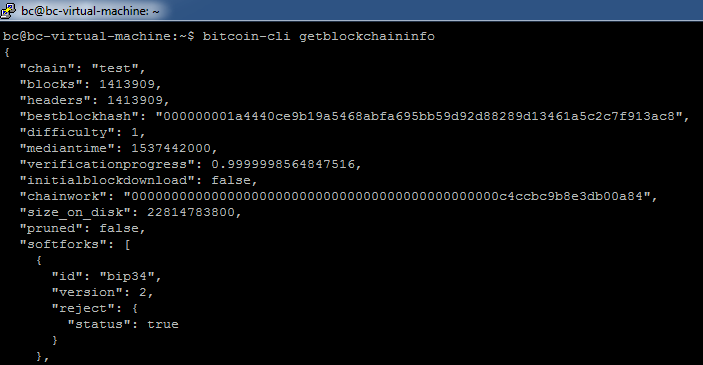


**<Step. 7> Search for blocks**

7-1) Search for blocks connected to main chain

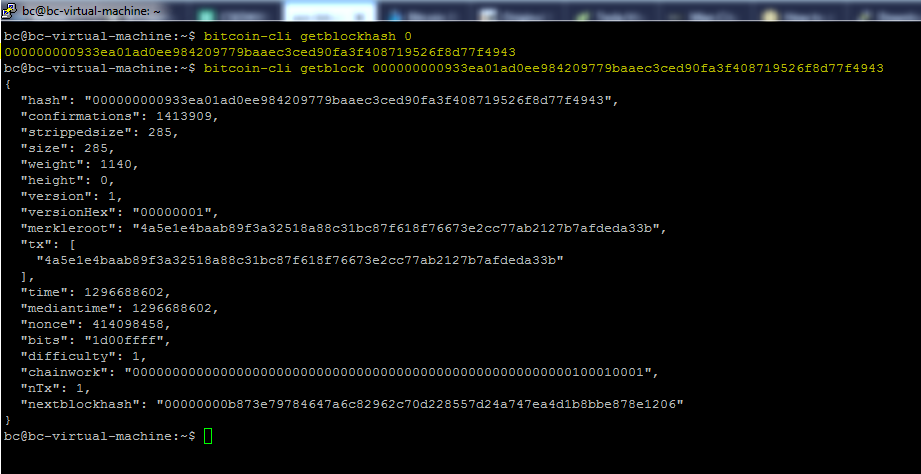
The general summary of blocks connected to the blockchain can be obtained using the command- *getblockchaininfo* i.e.

|  |
| --- |
| *bitcoin-cli getblockchaininfo* |



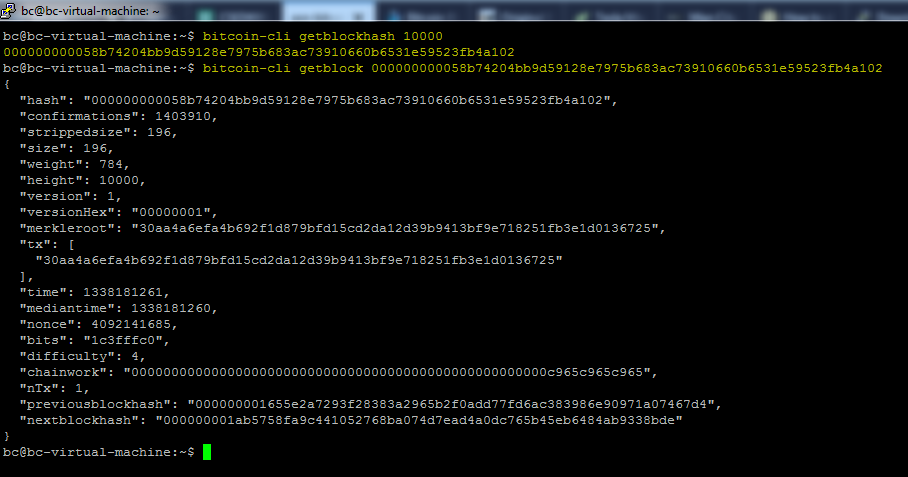
7-2) The block of height 0

|  |
| --- |
| *bitcoin-cli getblockhash 0* |
| *bitcoin-cli getblock* **HASH\_RETURNED\_BY\_BLOCK\_0** |



7-3) The block of height 10000

|  |
| --- |
| *bitcoin-cli getblockhash 10000* |
| *bitcoin-cli getblock* **HASH\_RETURNED\_BY\_BLOCK\_10000** |



7-4) Block including the transaction in which you transferred the BTC

In order to find the block containing the transaction we incurred, we have to first execute the *gettransaction* command followed by the transaction id as its argument, which also contains the *blockchainhash* field. Using this value of *blockchainhash* field as argument into the *getblock* command, we can find the block that contains the transaction we used for transferring bitcoin i.e.

|  |
| --- |
| *bitcoin-cli gettransaction 5ff6299e6abb2aab3956c4a36d066d9720981d7ff80ede6cd5928d3bedaf3894* |
| *bitcoin-cli getblock* **BLOCKCHAINHASH\_FIELD\_VALUE** |

