Most Popular Consensus Algorithms

Proof-Of-Work (2009) — nodes do complex calculations to «mine» blocks.
- Most secure, energy Inefficient.
- Bitcoin (BTC), Bitcoin Cash (BCH), Ethereum (ETH), Monero (XMR), Zcash (ZEC), Ravencoin (RVN), SUQA etc.

Proof-Of-Stake (2013) – mining based on coins posession.
- Energy efficient, full decentralization is not possible.
- Dash, Pivx, Reddcoin, Neblio, QTUM, NAV Coin, Stratis etc. Ethereum 2.0...

Delegated Proof-of-Stake (2014) — PoS variation, only elected delegate nodes create blocks.
- Scalable, Fast. Good for Enterprize usage.
BitShares, Lisk (LSK), EOS (EOS), Steem, Ark, Nano, Cardano, Tezos, TRON.

PoS Byzantine Fault Tolerance, PoS BFT (2017) — reliable to compromised nodes and connection problems.
- Reliable, Fast.
NEO, TON.

Proof of Authority — no mining.
- Very fast. Good for Enterprize usage.
Ethereum Kovan, private Ethereum networks.

Blockchain Generations

• First generation: Single-chain, PoW, no support for smart contracts.
Examples: Bitcoin (2009) and a lot of otherwise uninteresting imitators (Litecoin, Monero, . . .).

• Second generation: Single-chain, PoW, smart-contract support.
Example: Ethereum (2013; deployed in 2015), at least in its original form.

• Third generation: Single-chain, PoS, smart-contract support.
Example: future Ethereum (2018 or later).

• Alternative third (30) generation: Multi-chain, PoS, no support for smart contracts, loosely-coupled.
Example: Bitshares (2013–2014; uses DPOS).

• Fourth generation: Multi-chain, PoS, smart-contract support, Loosely-coupled.
Examples: EOS (2017; uses DPOS), PolkaDot (2016; uses BFT).

• Fifth generation: Multi-chain, PoS with BFT, smart-contract support, tightly-coupled, with sharding.
Examples: TON (2017).

Blockchain Generations

Project	Year	G.	Cons.	Sm.	Ch.	R.	Sh.	Int.
Bitcoin	2009	1	PoW	no	1			
Ethereum	2013, 2015	2	PoW	yes	1			
NXT	2014	2+	PoS	no	1			
Tezos	2017, ?	2+	PoS	yes	1			
Casper	2015, (2017)	3	PoW/PoS	yes	1			
BitShares	2013, 2014	3′	DPoS	no	m	ht.	no	L
EOS	2016, (2018)	4	DPoS	yes	m	ht.	no	L
PolkaDot	2016, (2019)	4	PoS BFT	yes	m	ht.	no	L
Cosmos	2017, ?	4	PoS BFT	yes	m	ht.	no	L
TON	2017, (2018)	5	PoS BFT	yes	m	mix	dyn.	T

Table 1: A summary of some notable blockchain projects.

The columns are:
Project– project name
Year– year announced and year deployed
G.– generation
Cons.– consensus algorithm
Sm.– support for arbitrary code (smart contracts)
Ch.– single/multipleblockchain system
R.– heterogeneous/homogeneous multichain systems (all blocks bloks have the same structure)
Sh.–sharding support
Int.– interaction between blockchains, (L)oose or (T)ight

Gathered $1.7B dollars from 170 investors wrong! Minted Gram coin

Pavel Durov told he will not support TON and pass it to the community due to USA restrictions

FreeTON => Everscale from Ton LABS in 2021. Won RustCup in 2021 with 55k tps = fastest blockchain. EVER coin (EVER)

The Open Network (TON) got ton.org and github project from Durov brothers. TON Coin (TON)

TON Components: fast multiblockchain

A flexible multi-blockchain platform:
- capable of processing millions of transactions per second,
- with Turing-complete smart contracts,
- upgradable formal blockchain specifications,
- multi-cryptocurrency value transfer,
- support for micropayment channels and off-chain payment networks.

New and unique features:
- “self-healing” vertical blockchain
- mechanism and Instant Hypercube Routing which enable it to be fast, reliable, scalable and self-consistent at the same time.

TON Components: P2P network

A peer-to-peer network, used for accessing the TON Blockchain,
sending transaction candidates,
and receiving updates about only those parts of the blockchain a client is interested in (e.g., those related to the client's accounts and smart contracts),
but also able to support arbitrary distributed services, blockchain-related or not.

TON Components: distributed storage

A distributed file storage technology, accessible through TON Network,
used by the TON Blockchain to store archive copies of blocks and status data (snapshots),
but also available for storing arbitrary files for users or other services running on the platform,
with torrent-like access technology.

TON Components: proxy/anonymizer layer

A network proxy/anonymizer layer, similar to the I2P (Invisible Internet Project), used to hide the identity and IP addresses of TON Network nodes if necessary (e.g., nodes committing transactions from accounts with large amounts of cryptocurrency,
or high-stake blockchain validator nodes who wish to hide their exact IP address and geographical location as a measure against
DDoS attacks).

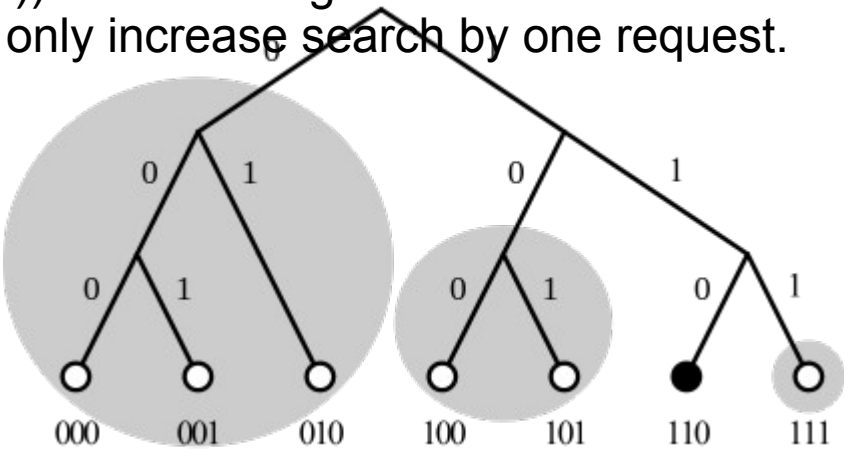
TON Components: DHT

A Kademlia-like distributed hash table, used as a “torrent tracker” for TON Storage, as an “input tunnel locator” for TON Proxy,
and as a service locator for TON Services.

Kademlia (DHT) is a distributed hash table for decentralized peer-to-peer computer networks designed by Petar Maymounkov and David Mazières in 2002. It specifies the structure of the network and the exchange of information through node lookups. Kademlia nodes communicate among themselves using UDP. (in contrast Bitcoin use TCP persistent connections). A virtual or overlay network is formed by the participant nodes. Each node is identified by a number or node ID. The node ID serves not only as identification, but the Kademlia algorithm uses the node ID to locate values (usually file hashes or keywords). In fact, the node ID provides a direct map to file hashes and that node stores information on where to obtain the file or resource.

Like many other DHTs, Kademlia contacts only O(log(n)) nodes during the search out of a total of n nodes in the system. Doubling networks size only increase search by one request.

Kademlia uses a "distance" calculation between two nodes. This distance is computed as the exclusive or (XOR) of the two node IDs, taking the result as an integer number. Keys and Node IDs have the same format and length, so distance can be calculated among them in exactly the same way. The node ID is typically a large random number that is chosen with the goal of being unique for a particular node. It can and does happen that geographically widely separated nodes—from Germany and Australia, for instance—can be "neighbors" if they have chosen similar random node IDs.



Kademlia

Public networks using the Kademlia algorithm (these networks are incompatible with one another):

I2P – an anonymous network built on top of the internet.

Kad Network – developed originally by the eMule community to replace the server-based architecture of the eDonkey2000 network.

Overnet network: With KadC a C library for handling its Kademlia is available. (development of Overnet is discontinued)

BitTorrent Uses a DHT based on an implementation of the Kademlia algorithm, for trackerless torrents.

Osiris sps (all version): used to manage distributed and anonymous web portal.

Retroshare – F2F decentralised communication platform with secure VOIP, instant messaging, file transfer etc.

Tox – A fully distributed messaging, VoIP and video chat platform

Gnutella DHT – Originally by LimeWire to augment the Gnutella protocol for finding alternate file locations, now in use by other gnutella clients.

IPFS − A peer-to-peer distributed filesystem (The InterPlanetary File System, supported by Microsoft Asure).

TeleHash is a mesh networking protocol that uses Kademlia to resolve direct connections between parties.

TON Components: Services

A platform for arbitrary services,

residing in and available through TON Network and TON Proxy,

with formalized interfaces enabling browser-like or smartphone application interaction.

These formal interfaces and persistent service entry points can be published in the TON Blockchain;

actual nodes providing service at any given moment can be looked up through the TON DHT

starting from information published in the TON Blockchain.

Services may create smart contracts in the TON Blockchain to offer some guarantees to their clients.

TON Components: DNS, Payments, Integrations

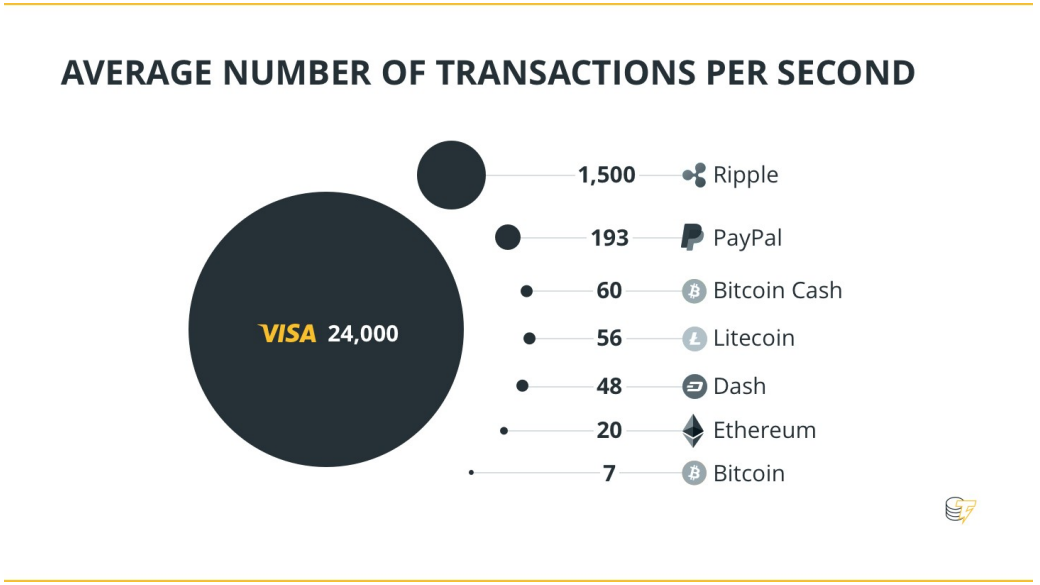
- TON DNS, a service for assigning human-readable names to accounts, smart contracts, services and network nodes.

- TON Payments, a platform for micropayments, micropayment channels and a micropayment channel network. It can be used for fast off-chain value transfers, and for paying for services powered by TON Services.

- TON will allow easy integration with third-party messaging and social networking applications (Telegram Messenger), thus making blockchain technologies and distributed services finally available and accessible to ordinary users, rather than just to a handful of early cryptocurrency adopters.

TON Blockchain as a Collection of 2-Blockchains

The TON Blockchain is actually a collection of blockchains (even a collection of blockchains of blockchains, or 2-blockchains),
because no single blockchain project is capable of achieving
our goal of processing millions of transactions per second,
as opposed to the now-standard dozens of transactions per second.



60 000 tps according to
Visa and Mastercard

Source https://cointelegraph.com/news/ethereum-blockchain-solutions-for-enterprise-business-pros-and-cons

TON: Shardchains

Each workchain is in turn subdivided into up to 2^60 shard blockchains (shardchains),

having the same rules and block format as the workchain itself,

but responsible only for a subset of accounts,

depending on several first (most significant) bits of the account address.

This form of sharding is built into the system.

Because all these shardchains share a common block format and rules,

the TON Blockchain is homogeneous in this respect,

similarly to what has been discussed in one of Ethereum scaling proposals.

TON: 2D Blockchain

Each block in a shardchain (and in the masterchain) is actually not just a block, but a small blockchain. Normally, this “block blockchain” or “vertical blockchain” consists of exactly one block, and then we might think this is just the corresponding block of the shardchain (also called “horizontal blockchain” in this situation).

However, if it becomes necessary to fix incorrect shardchain blocks, a new block is committed into the “vertical blockchain”, containing either the replacement for the invalid “horizontal blockchain” block, or a “block difference”, containing only a description of those parts of the previous version of this block that need to be changed.

This is a TON-specific mechanism to replace detected invalid blocks without making a true fork of all shardchains involved.

Each shardchain (and the masterchain) is not a conventional blockchain, but a blockchain of blockchains, or 2D-blockchain, or just a 2-blockchain.

TON: Infinite Sharding Paradigm

Almost all blockchain sharding proposals are “top-down”: one first imagines a single blockchain, and then discusses how to split it into several interacting shardchains to improve performance and achieve scalability.

The TON approach to sharding is “bottom-up”:

Imagine that sharding has been taken to its extreme, so that exactly one account or smart contract remains in each shardchain. Then we have a huge number of “account-chains”, each describing the state and state transitions of only one account, and sending value-bearing messages to each other to transfer value and information.

Of course, it is impractical to have hundreds of millions of blockchains, with updates (i.e., new blocks) usually appearing quite rarely in each of them. In order to implement them more efficiently, we group these “account-chains” into “shardchains”, so that each block of the shardchain is essentially a collection of blocks of account-chains that have been assigned to this shard. Thus the “account-chains” have only a purely virtual or logical existence inside the “shardchains”.

We call this perspective the Infinite Sharding Paradigm. It explains many of the design decisions for the TON Blockchain.

TON: Messages

The Infinite Sharding Paradigm instructs us to regard each account (or smart contract) as if it were in its own shardchain by itself.

Then the only way one account might affect the state of another is by sending a message to it.

Therefore, a system of messages between accounts (and shardchains, because the source and destination accounts are, generally speaking, located in different shardchains) is of paramount importance to a scalable system such as the TON Blockchain. In fact, a novel feature of the TON Blockchain, called Instant Hypercube Routing, enables it to deliver and process message created in a block of one shardchain into the very next block of the destination shardchain, regardless of the total number of shardchains in the system.

Each account-chain (i.e., each account) has its own output queue, consisting of all messages it has generated, but not yet delivered to their recipients. Of course, account-chains have only a virtual existence; they are grouped into shardchains, and a shardchain has an output “queue”, consisting of the union of the output queues of all accounts belonging to the shardchain.

This shardchain output “queue” imposes only partial order on its member messages. Namely, a **message generated in a preceding block must be delivered before any message generated in a subsequent block**, and any messages generated by the same account and having the same destination must be delivered in the order of their generation.

«Slow» and «Fast» algorithms are used to deliver messages.

TON: Slow Reliable Message Delivery

The TON Blockchain uses “hypercube routing” as a slow, but safe and reliable way of delivering messages from one shardchain to another, using several intermediate shardchains for transit if necessary. Otherwise, the validators of any given shardchain would need to keep track of the state of (the output queues of) all other shardchains, which would require prohibitive amounts of computing power and network bandwidth as the total quantity of shardchains grows, thus limiting the scalability of the system. Therefore, it is not possible to deliver messages directly from any shard to every other.

Instead, each shard is “connected” only to shards differing in exactly one hexadecimal digit of their (w, s) shard identifiers. In this way, all shardchains constitute a “hypercube” graph, and messages travel along the edges of this hypercube. If a message is sent to a shard different from the current one, one of the hexadecimal digits (chosen deterministically) of the current shard identifier is replaced by the corresponding digit of the target shard, and the resulting identifier is used as the proximate target to forward the message to.

TON: Instant Hypercube Routing.

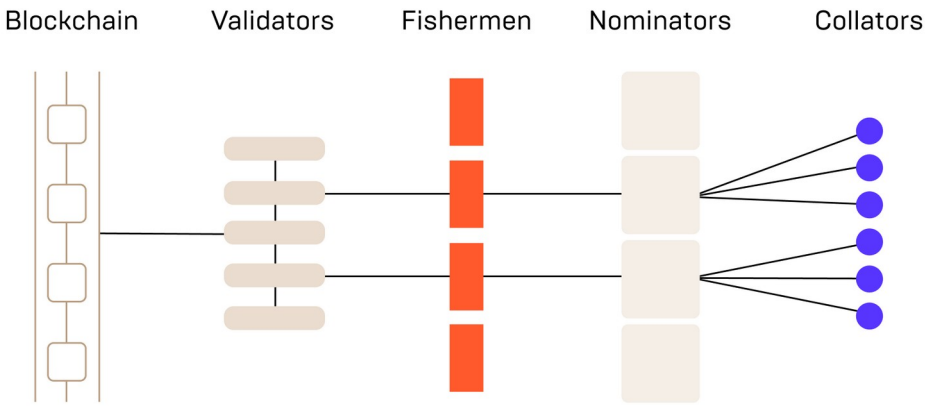
The idea is as follows. During the “slow” hypercube routing, the message travels (in the network) along the edges of the hypercube, but it is delayed (for approximately five seconds) at each intermediate vertex to be committed into the corresponding shardchain before continuing its voyage.

To avoid unnecessary delays, one might instead relay the message along with a suitable Merkle proof along the edges of the hypercube, without waiting to commit it into the intermediate shardchains. In fact, the network message should be forwarded from the validators of the “task group” of the original shard to the designated block producer of the “task group” of the destination shard; this might be done directly without going along the edges of the hypercube. When this message with the Merkle proof reaches the validators (more precisely, the collators) of the destination shardchain, they can commit it into a new block immediately, without waiting for the message to complete its travel along the “slow path”. Then confirmation of delivery along with a suitable Merkle proof is sent back along the hypercube edges, and it may be used to stop the travel of the message along the “slow path”, by committing a special transaction.

Note that this “instant delivery” mechanism does not replace the “slow” but failproof mechanism. The “slow path” is still needed because the validators cannot be punished for losing or simply deciding not to commit the “fast path” messages into new blocks of their blockchains.

Therefore, both message forwarding methods are run in parallel, and the “slow” mechanism is aborted only if a proof of success of the “fast” mechanism is committed into an intermediate shardchain.

TON: Consensus



The TON consensus network includes nodes of different types:

The validators are the PoS nodes and block producers.

The fishermen can that the consensus network looking for errors or trying to detect a presumably malicious node and in case the fisherman clearly confirms that there is such node, it will obtain a reward in the form of confiscation of a part of the validator's stake.

The collators are tasked with preparing the shardchain blocks and submitting them to validation by the PoS nodes for which they obtain their fraction of reward for creating a block. In such case, the collators are basically additional participants of the consensus as the validators almost always generate the blocks on their own.

The nominators lend their assets (Gram tokens) to the validators in order to get a profit. The nominators do not practically enter the infrastructure of the validators and only distribute their initial large stake in the asset among them in exchange for a proportionate percentage of the entire reward.

Source: https://hackernoon.com/telegram-open-network-insights-by-the-network-validator-cg2732gj

TON: History

https://test.ton.org/

Original distribution:
- Lite-Client command line node and client in C++.
- Validator node.
- Fift to compile, execute, and debug your smart contracts locally (Forth based stack machine language).

15.11.2019 testnet2 launch.

06.07.2020: We are discontinuing our support of the test network of the TON Blockchain. Our remaining validators will be switched off not later than 1.08.2020. Please save all relevant data and terminate your testing process.

24.05.2020: The original TON development team is discontinuing its active involvement with the TON project.

15.05.2021 Testnet2 renamed to mainnet. NEWTON community renamed to TON Foundation.

29.06.2021 ton.org domain name and GitHub repo passed to TON community by request.

9.08.2021 TONCOIN listed in EXMO exchange.

In 2024 Ton Wallet integrated to Telegram, TON endorsed by Telegram, used exclusively to pay for Ads. The TON Foundation's president is Steve Yun, the body's website says. Andrew Rogozov, former CEO of Russian social media site VK, is a founding member of the foundation, according to his LinkedIn profile. He appeared on stage with Telegram co-founder Pavel Durov and Tether CEO Paolo Ardoino at a conference in Dubai where the partnership to issue the stablecoin on the TON blockchain was announced. Pavel Durov was arrested in France.

Free TON / Everscale

Ton Open Network
Original testnet server owners
PoW Giver Smart contracts
1.5kk accounts
238 validators
Pavel Durov transfered rights to community:
https://ton.org/
https://github.com/ton-blockchain

https://freeton.org/
https://tonlabs.io/products
https://docs.ton.dev/

Node JS based smart contract development tools.

Development node.

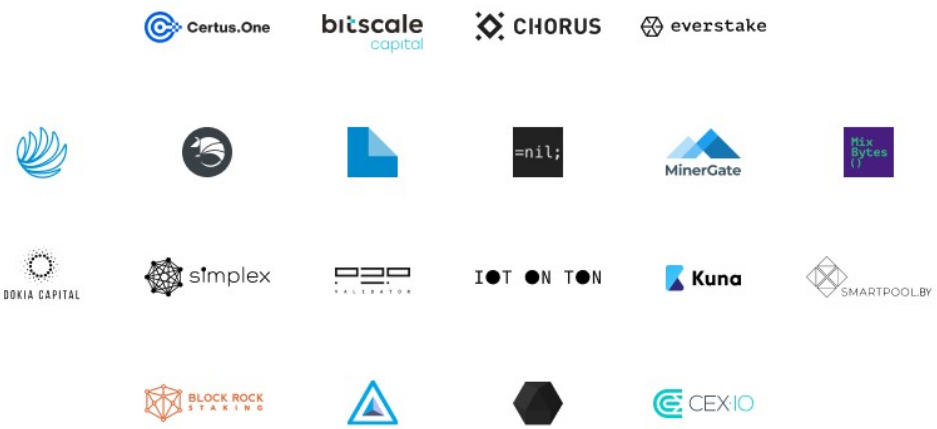
Solidity to Fift compiler.

TON OS Startup Edition - a local blockchain for development and testing.

TON OS SE Components:

- TON Labs implementation of TON VM written in Rust
- ArangoDB database,
- GraphQL endpoint with web playground.
- Pre-deployed Giver

Telegram:
@freetondevru (1400+ members)
@freetondev_ru (2000+ members)



TON Hello World

```
sin@thinkpad:~$ cd Univer/Blockchain/
sin@thinkpad:~/Univer/Blockchain$ mkdir TON
sin@thinkpad:~/Univer/Blockchain$ cd TON/
sin@thinkpad:~/Univer/Blockchain/TON$ npm create ton@latest
Need to install the following packages:
  create-ton@latest
Ok to proceed? (y) Y

? Project name Counter
? First created contract name (PascalCase) Counter
? Choose the project template A simple counter contract (FunC)

[1/3] Copying files...
[2/3] Installing dependencies...

(([redacted])) ⁝ idealTree:Counter: sill idealTree buildDeps
```

Shell

TON : npm create ton@ - Drop-Down Terminal

data between calls from different users, for example the value of our *counter* variable. To write this data to state storage, we will need a write/encode function and to read this data back from state storage, we will need a read/decode function.

The **messages** section deals with messages sent to our contract. The main form of interaction with contracts on TON Blockchain is by sending them messages. We mentioned before that our contract will need to support a variety of actions like *increment*, *deposit*, *withdraw* and *transfer ownership*. All of these operations are performed by users as transactions. These operations are not read-only because they change something in the contract's persistent state.

The **getters** section deals with read-only interactions that don't change state. For example, we would want to allow users to query the value of our *counter*, so we can implement a getter for that. We've also mentioned that the contract has a special *owner*, so what about a getter to query that. Since our contract can hold money (TON coins), another useful getter could be to query the current balance.

Step 5: Implement the Counter contract

03. Building your first web client

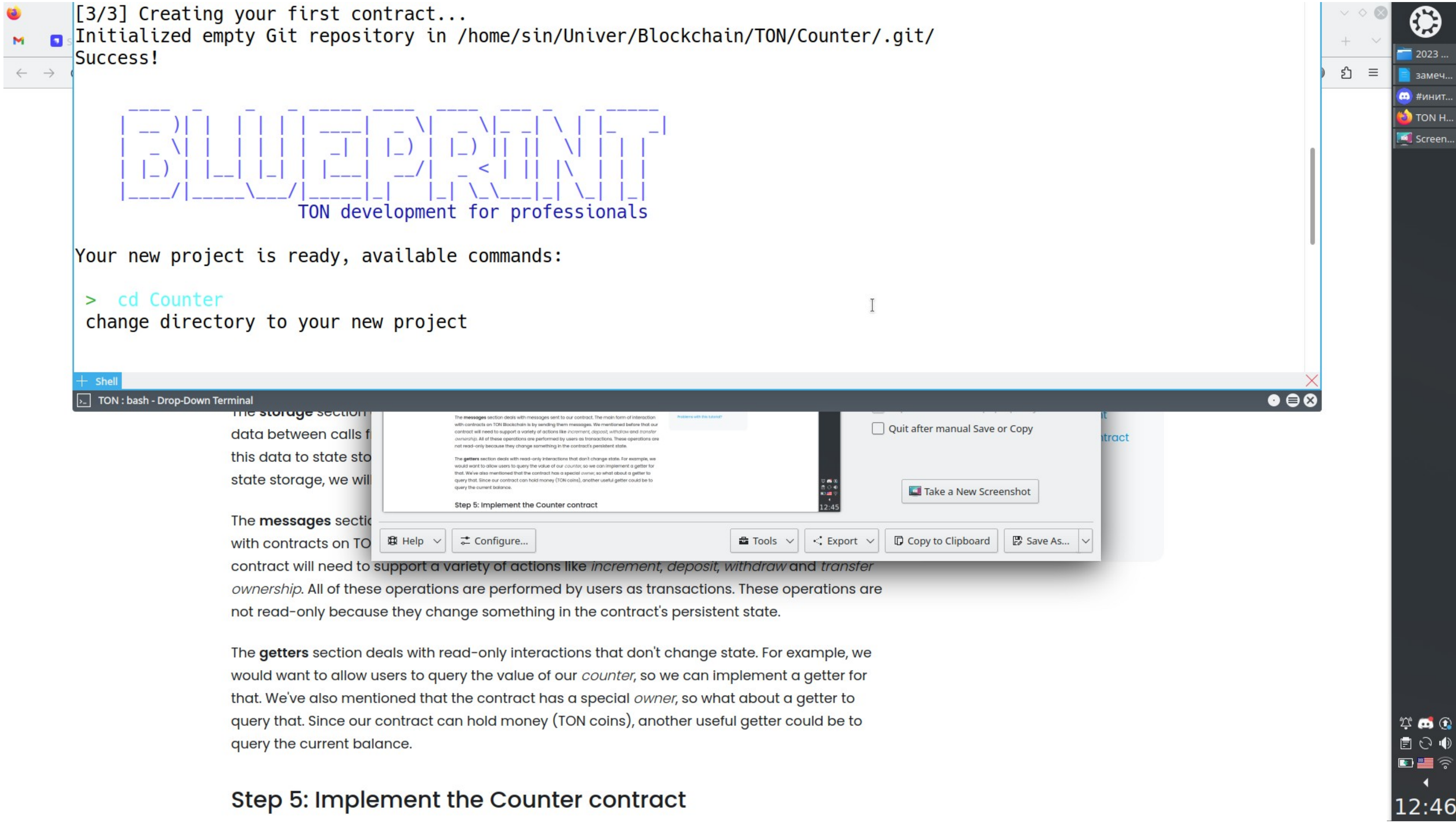
04. Testing your first smart contract

Help

Problems with this tutorial?

https://tonhelloworld.com/02-contract/

Project commands

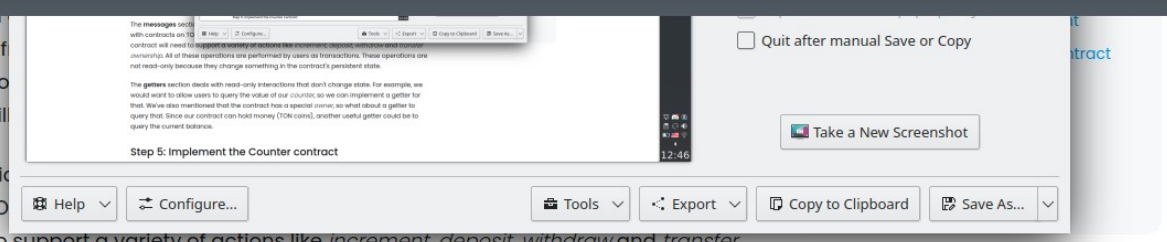


Project commands

```
Your new project is ready, available commands:

>  cd Counter
change directory to your new project

>  npx blueprint build
choose a smart contract and build it

>  npx blueprint test
run the default project test suite

>  npx blueprint run
choose a script and run it (eg. a deploy script)

>  npx blueprint create AnotherContract
create all the necessary files for another new contract
```

Shell

TON : bash - Drop-Down Terminal



data between calls f
this data to state sto
state storage, we wil

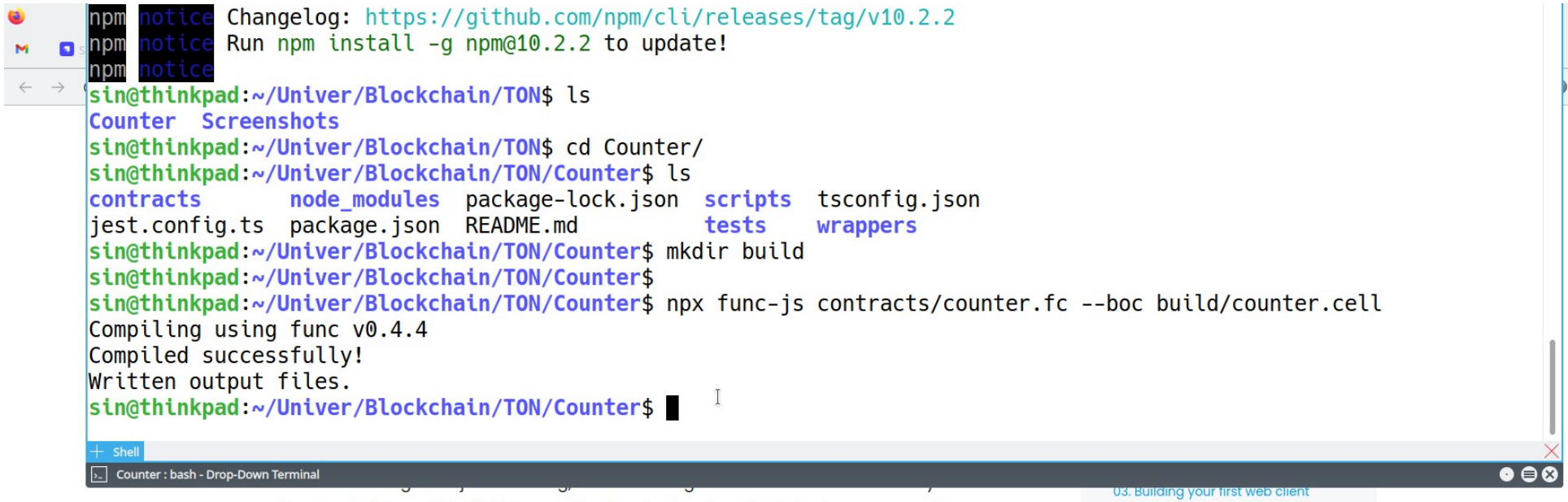
The **messages** secti
with contracts on TO
contract will need to support a variety of actions like *increment*, *deposit*, *withdraw* and *transfer ownership*. All of these operations are performed by users as transactions. These operations are not read-only because they change something in the contract's persistent state.

The **getters** section deals with read-only interactions that don't change state. For example, we would want to allow users to query the value of our *counter*, so we can implement a getter for that. We've also mentioned that the contract has a special *owner*, so what about a getter to query that. Since our contract can hold money (TON coins), another useful getter could be to query the current balance.

Step 5: Implement the Counter contract

12:47

Compile to bytecode



store the build result. To do this, open the terminal and run the following command:

```
mkdir build
```

The func-js package contains everything we need to compile our contract to bytecode. To use it, open terminal in the project directory and run the following:

```
npx func-js contracts/counter.fc --boc build/counter.cell
```

The build should succeed, with the output of this command being a new file - `counter.cell`. This is a binary file that finally contains the TVM bytecode in cell format that is ready to be deployed on-chain. This will actually be the only file we need for deployment moving forward (we won't need the FunC source file).

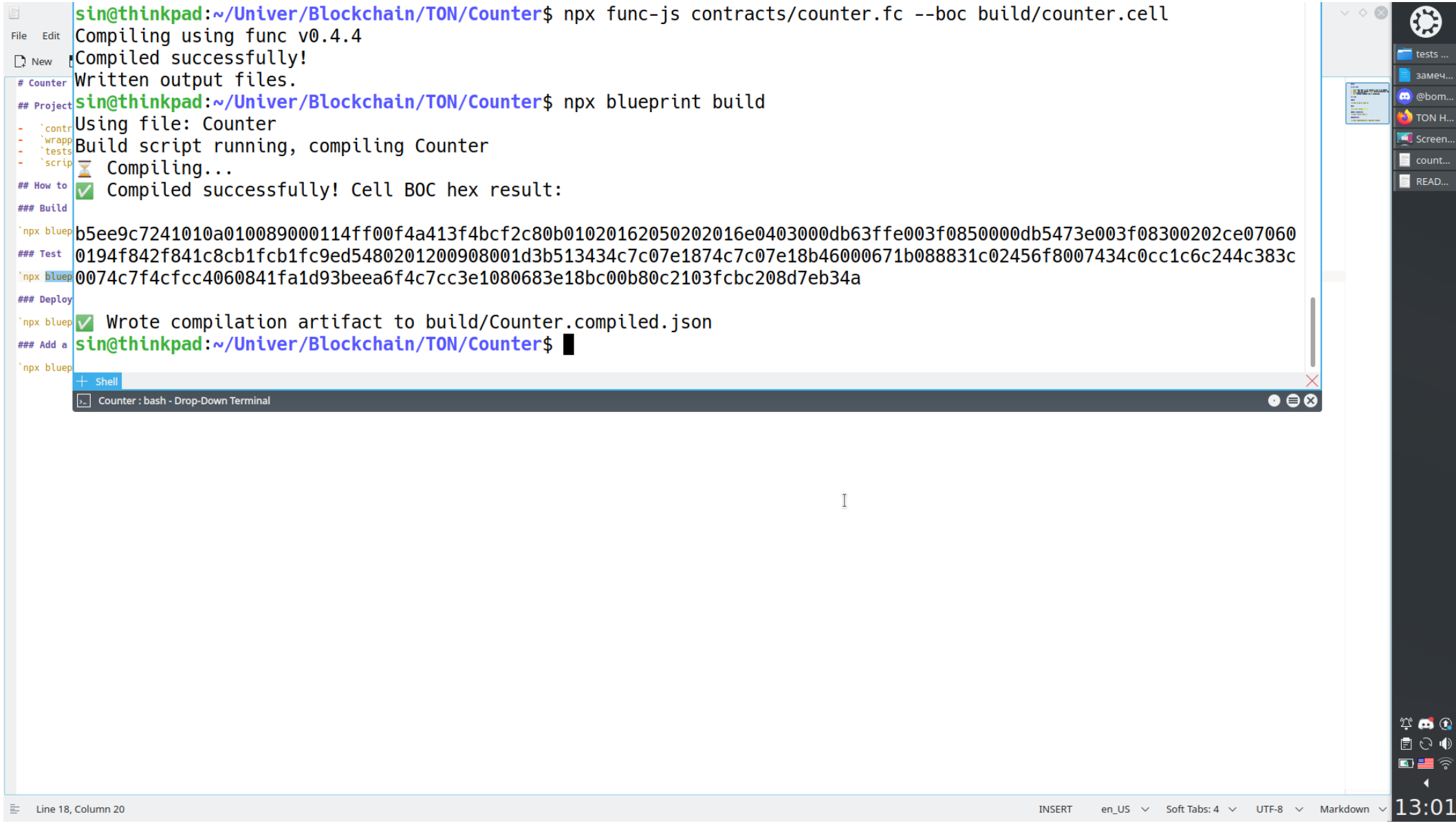
Step 7: Prepare init data for deploying on-chain

03. Building your first web client
04. Testing your first smart contract

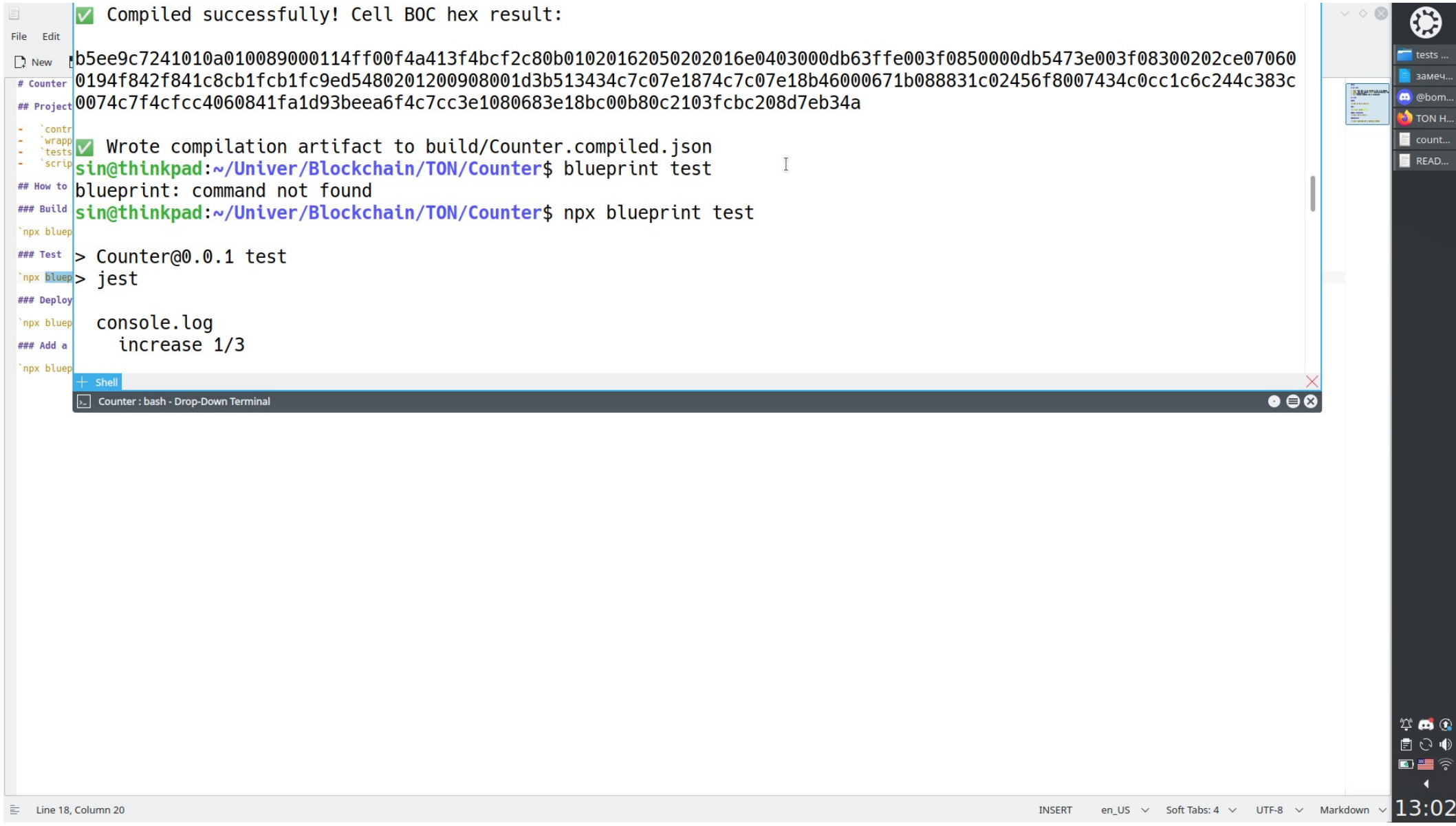
Help

Problems with this tutorial?

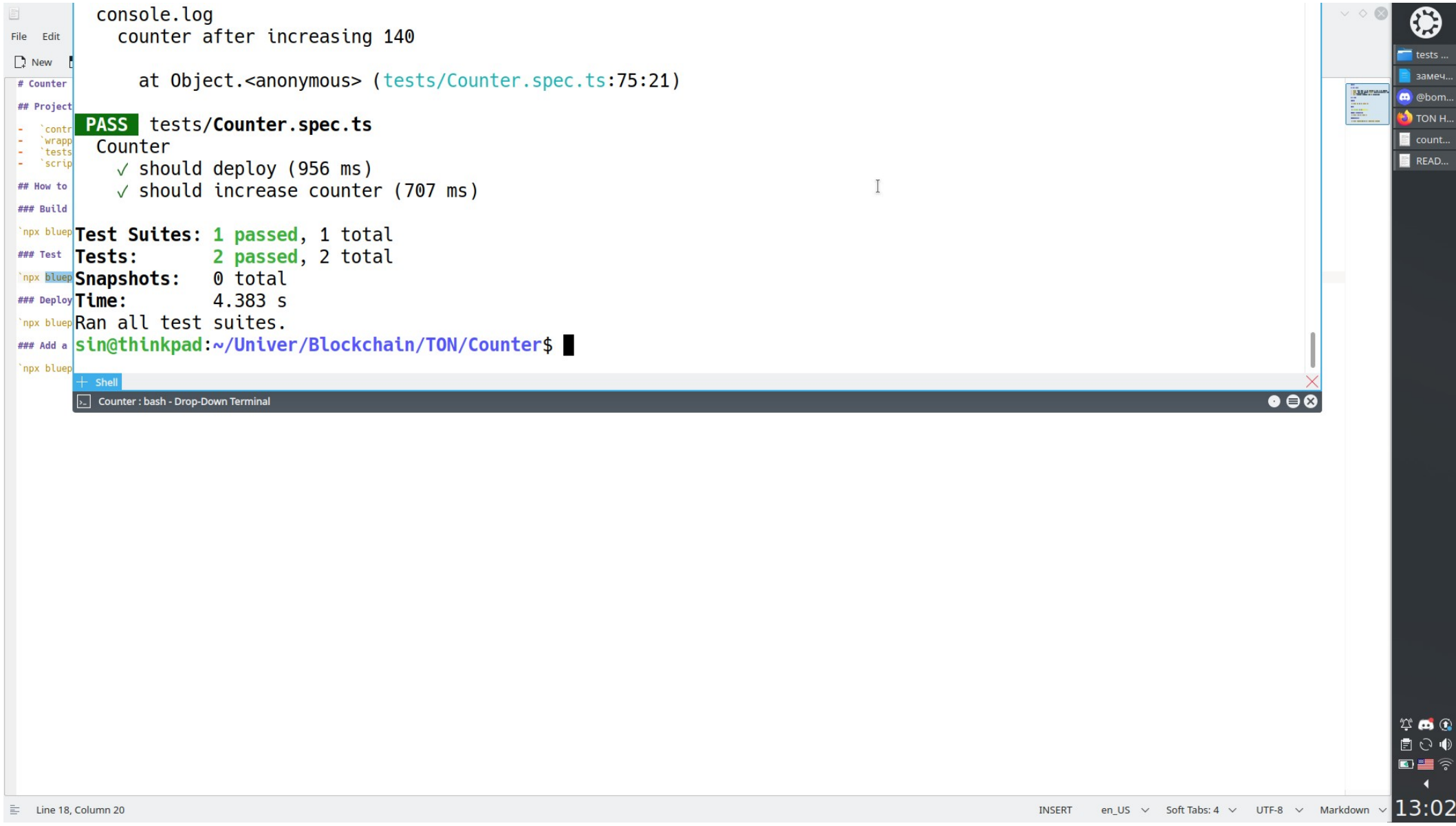
Build smart contracts



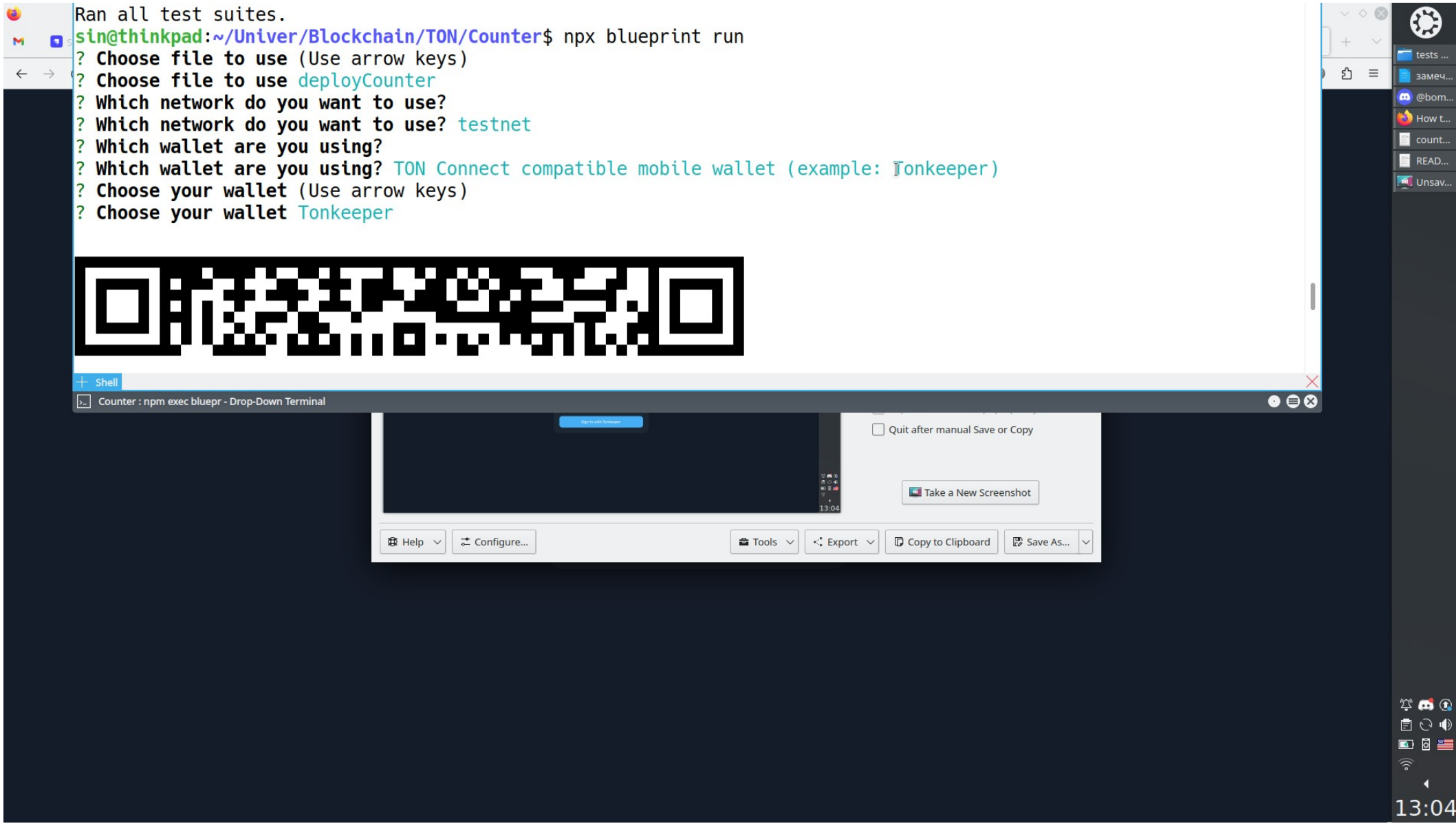
Run Tests


✅ Compiled successfully! Cell BOC hex result:
b5ee9c7241010a010089000114ff00f4a413f4bcf2c80b0102016205020201 6e0403000db63ffe003f0850000db5473e003f08300202ce07060
0194f842f841c8cb1fcb1fc9ed5480201200908001d3b513434c7c07e1874c7c07e18b46000671b088831c02456f8007434c0cc1c6c244c383c
0074c7f4cfcc4060841fa1d93beea6f4c7cc3e1080683e18bc00b80c2103fcbc208d7eb34a
✅ Wrote compilation artifact to build/Counter.compiled.json
sin@thinkpad:~/Univer/Blockchain/TON/Counter$ blueprint test
blueprint: command not found
sin@thinkpad:~/Univer/Blockchain/TON/Counter$ npx blueprint test
> Counter@0.0.1 test
> jest
console.log
increase 1/3
Shell
Counter : bash - Drop-Down Terminal

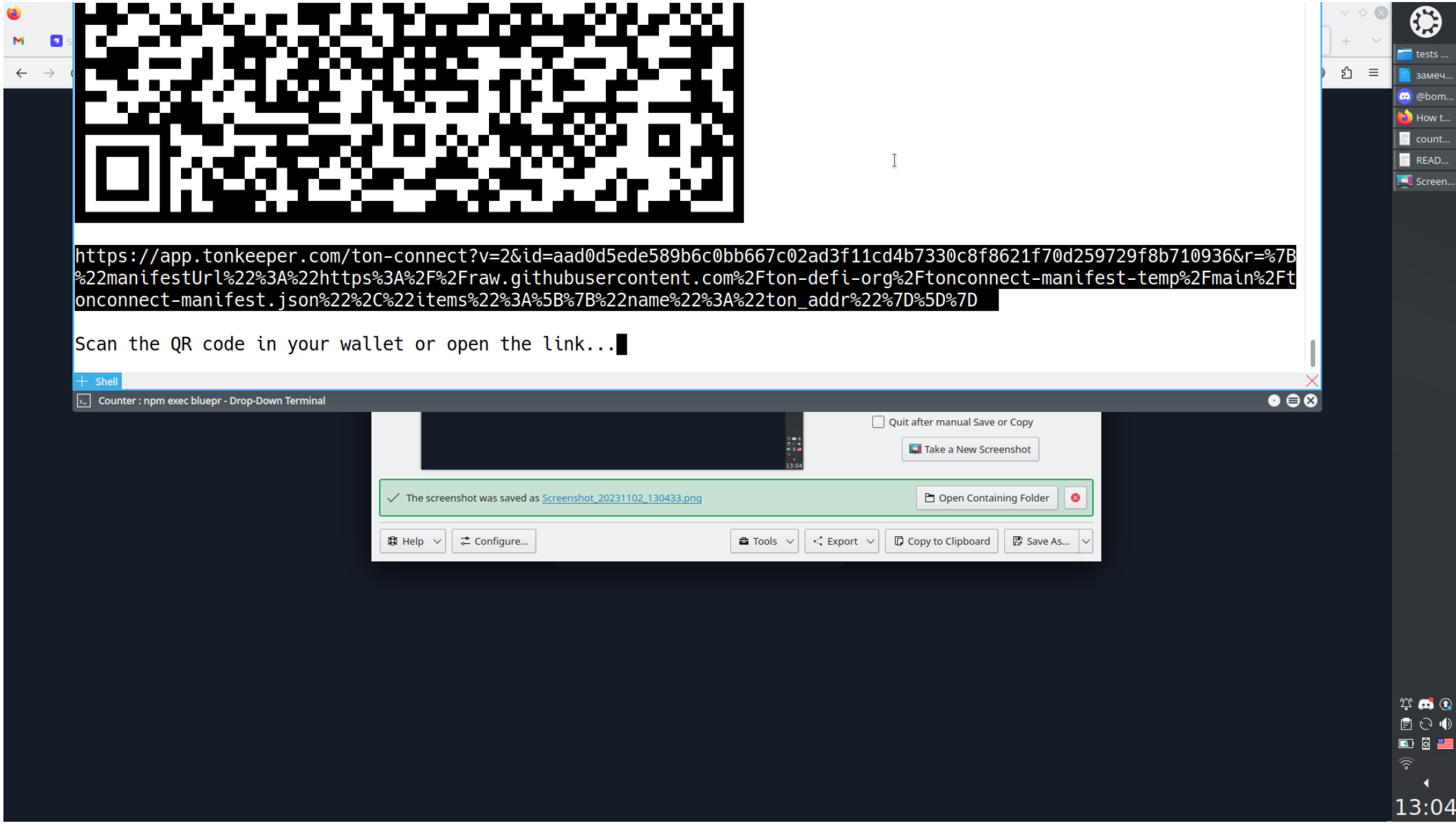
Run Tests


console.log
counter after increasing 140
at Object.<anonymous> (tests/Counter.spec.ts:75:21)
PASS tests/Counter.spec.ts
Counter
✓ should deploy (956 ms)
✓ should increase counter (707 ms)
Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 4.383 s
Ran all test suites.
sin@thinkpad:~/Univer/Blockchain/TON/Counter$
Shell
Counter : bash - Drop-Down Terminal
File Edit
New
Line 18, Column 20
INSERT
en_US
Soft Tabs: 4
UTF-8
Markdown
13:02

Deploy to testnet



Pay from TonKeeper wallet in dev mode



Pay from TonKeeper wallet in dev mode

https://app.tonkeeper.com/ton-connect?v=2&id=aad0d5ede589b6c0bb667c02ad3f11cd4b7330c8f8621f70d259729f8b710936&r=%7B%22manifestUrl%22%3A%22https%3A%2F%2Fraw.githubusercontent.com%2Fton-defi-org%2Ftonconnect-manifest-temp%2Fmain%2Ft

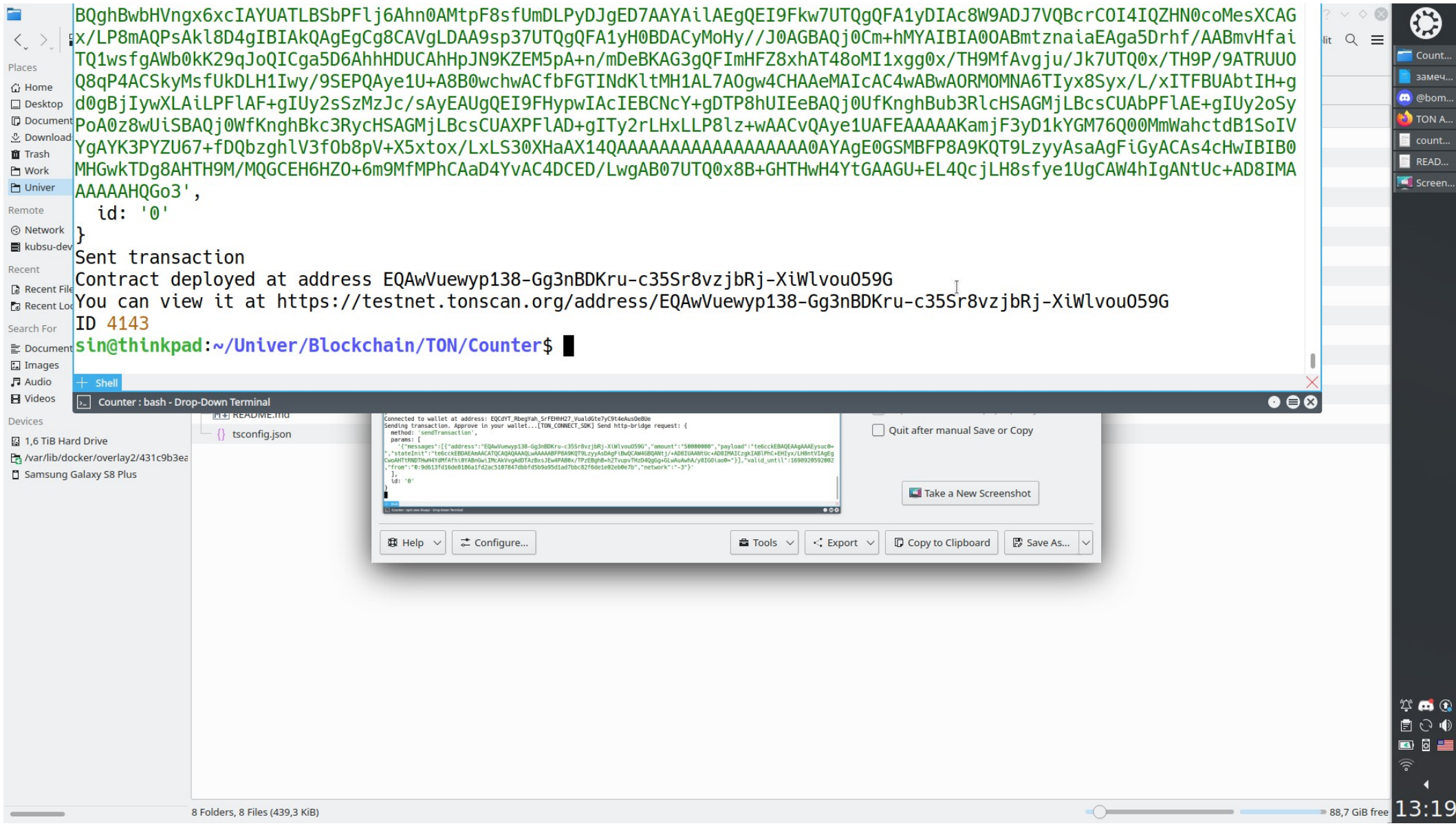
Approve in TonKeeper

```
https://app.tonkeeper.com/ton-connect?v=2&id=f9e71d59f9ce04f7c938b493ea52e258ac4fccf36dbbad6e0acd9c8785d7e241&r=%7B
%22manifestUrl%22%3A%22https%3A%2F%2Fraw.githubusercontent.com%2Fton-defi-org%2Ftonconnect-manifest-temp%2Fmain%2Ft
onconnect-manifest.json%22%2C%22items%22%3A%5B%7B%22name%22%3A%22ton_addr%22%7D%5D%7D

Scan the QR code in your wallet or open the link...[TON_CONNECT_SDK] Wallet message received: {
  id: 10,
  event: 'connect',
  payload: {
    items: [ [Object] ],
    device: {
      platform: 'android',
      appName: 'Tonkeeper',
      appVersion: '3.4.4.393',
      maxProtocolVersion: 2,
      features: [Array]
    }
  }
}
Connected to wallet at address: EQCdYT_RbegYah_SrFEHhH27_VualdGte7yC9t4eAusOe8Ue
Sending transaction. Approve in your wallet...[TON_CONNECT_SDK] Send http-bridge request: {
  method: 'sendTransaction',
  params: [
    '{"messages":[{"address":"EQAwVuewyp138-Gg3nBDKru-c35Sr8vzjbRj-XiWlvouO59G","amount":"50000000","payload":"te6cckEBAQEAAgAAAEysuc0=
","stateInit":"te6cckEBDAEAmAACATQCAQAQAAAQLwAAAAABFP8A9KQT9LzyyAsDAgFiBwQCAW4GBQANtj/+AD8IUAANtUc+AD8IMAICzgkIABlPhC+EHIyx/LH8ntVIAgEg
CwoAHTtRNDTHwH4YdMfAfhi0YABnGwiIMcAkVvgAdDTAzBxsJEw4PAB0x/TPzEBghB+h2TvupvTHzD4QgGg+GLwAuAwhA/y8IGOiao0="}],"valid_until":1698920592002
,"from":"0:9d613fd16de8186a1fd2ac5107847dbbfd5b9a95d1ad7bbc82f6de1e02eb0e7b","network":"-3"}'
  ],
  id: '0'
}
```

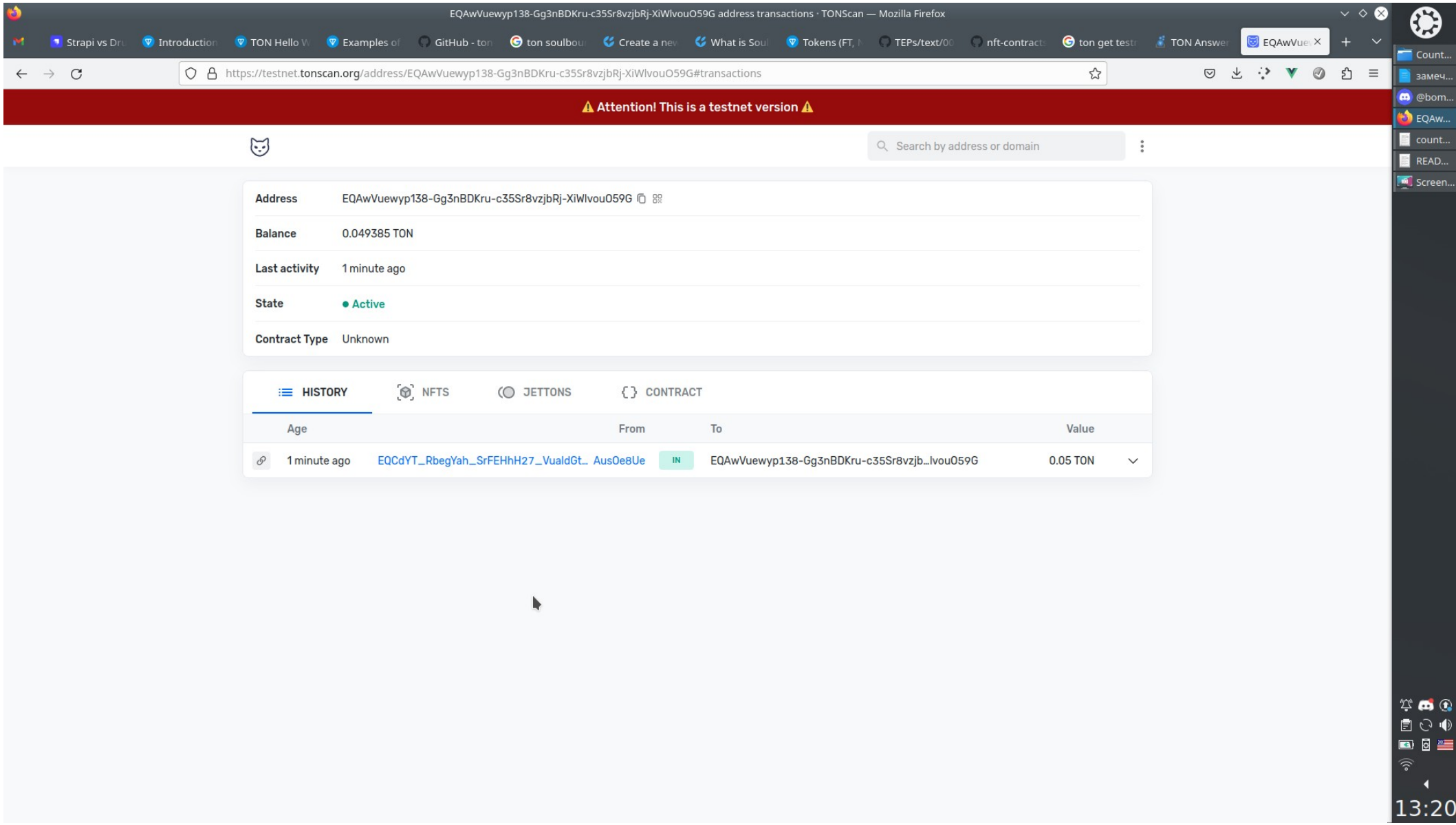
Shell

Counter : npm exec bluepr - Drop-Down Terminal

Contract Deployed!



Check in Blockchain Explorer



Read Counter

readCounter.ts — KWrite

File Edit View Bookmarks Tools Settings Help

New Open... Save Save As... Close Undo Redo

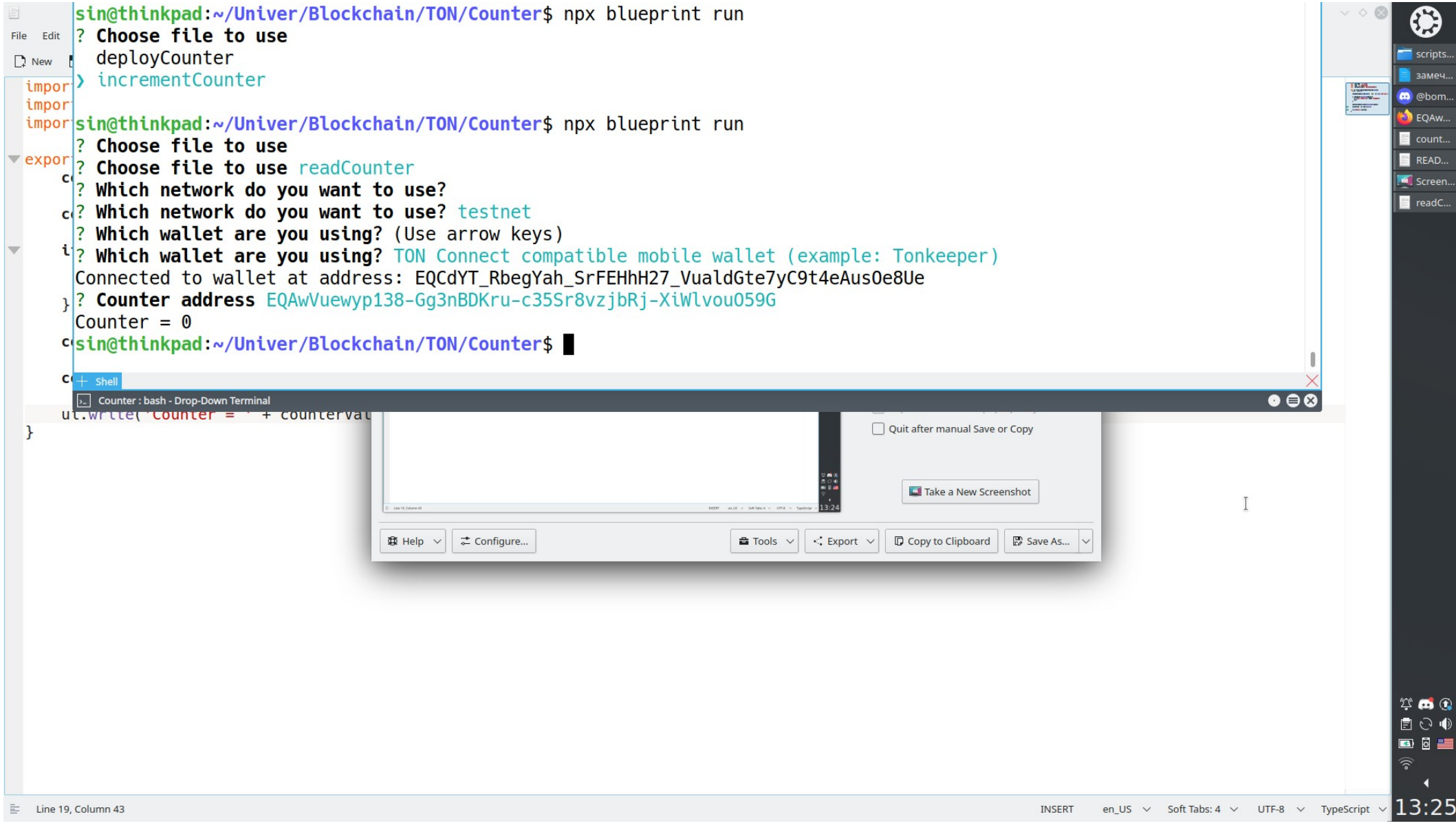
```ts
import { Address, toNano } from 'ton-core';
import { Counter } from '../wrappers/Counter';
import { NetworkProvider, sleep } from '@ton-community/blueprint';

export async function run(provider: NetworkProvider, args: string[]) {
    const ui = provider.ui();

    const address = Address.parse(args.length > 0 ? args[0] : await ui.input('Counter address'));

    if (!(await provider.isContractDeployed(address))) {
        ui.write(`Error: Contract at address ${address} is not deployed!`);
        return;
    }

    const counter = provider.open(Counter.createFromAddress(address));

    const counterValue = await counter.getCounter();

    ui.write('Counter = ' + counterValue);
}
```

Line 19, Column 43 INSERT en_US Soft Tabs: 4 UTF-8 TypeScript

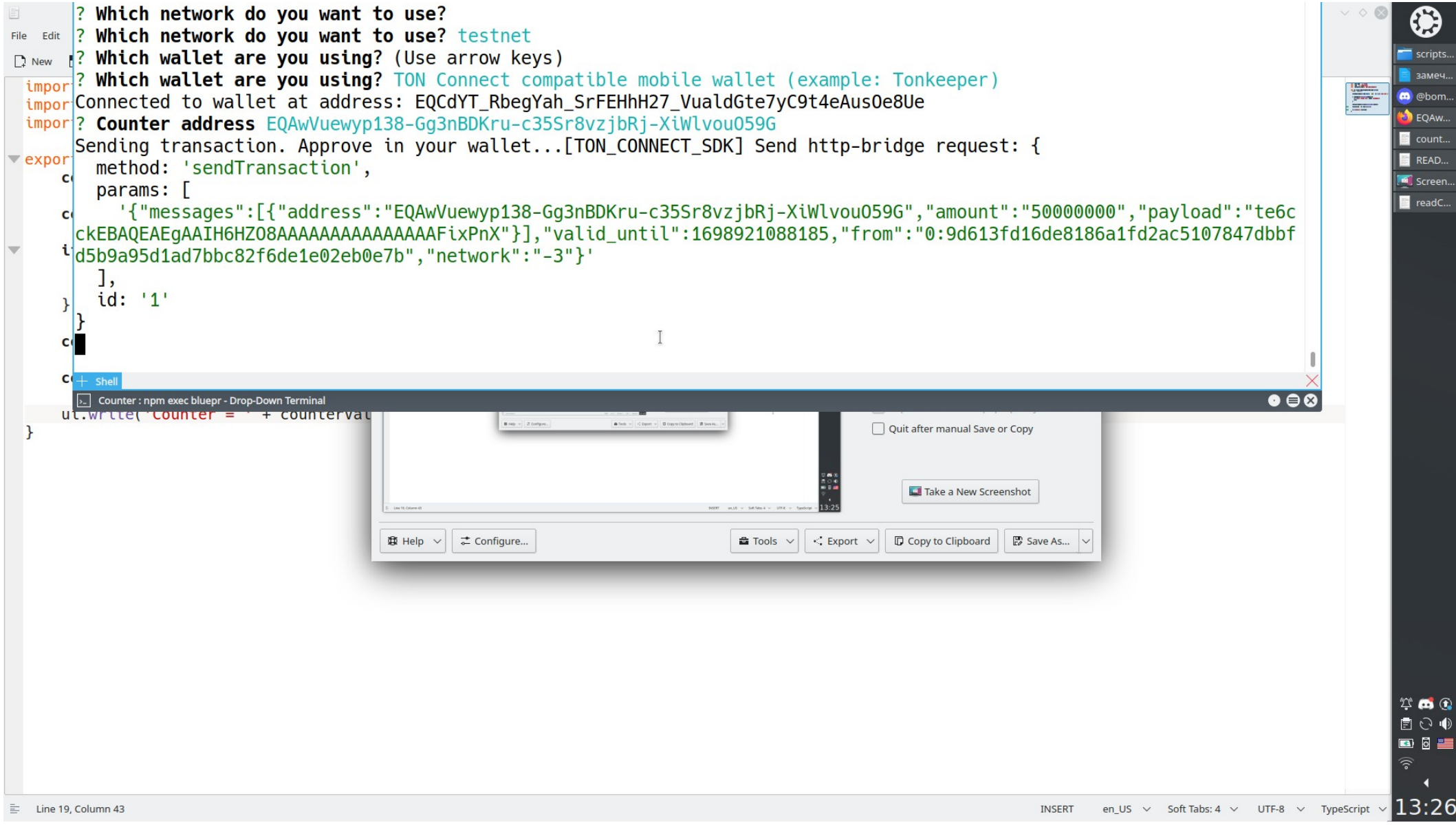
scripts... замеч... @bom... EQAw... count... READ... Screen... readC...

13:24

Run readCounter



Approve in TonKeeper



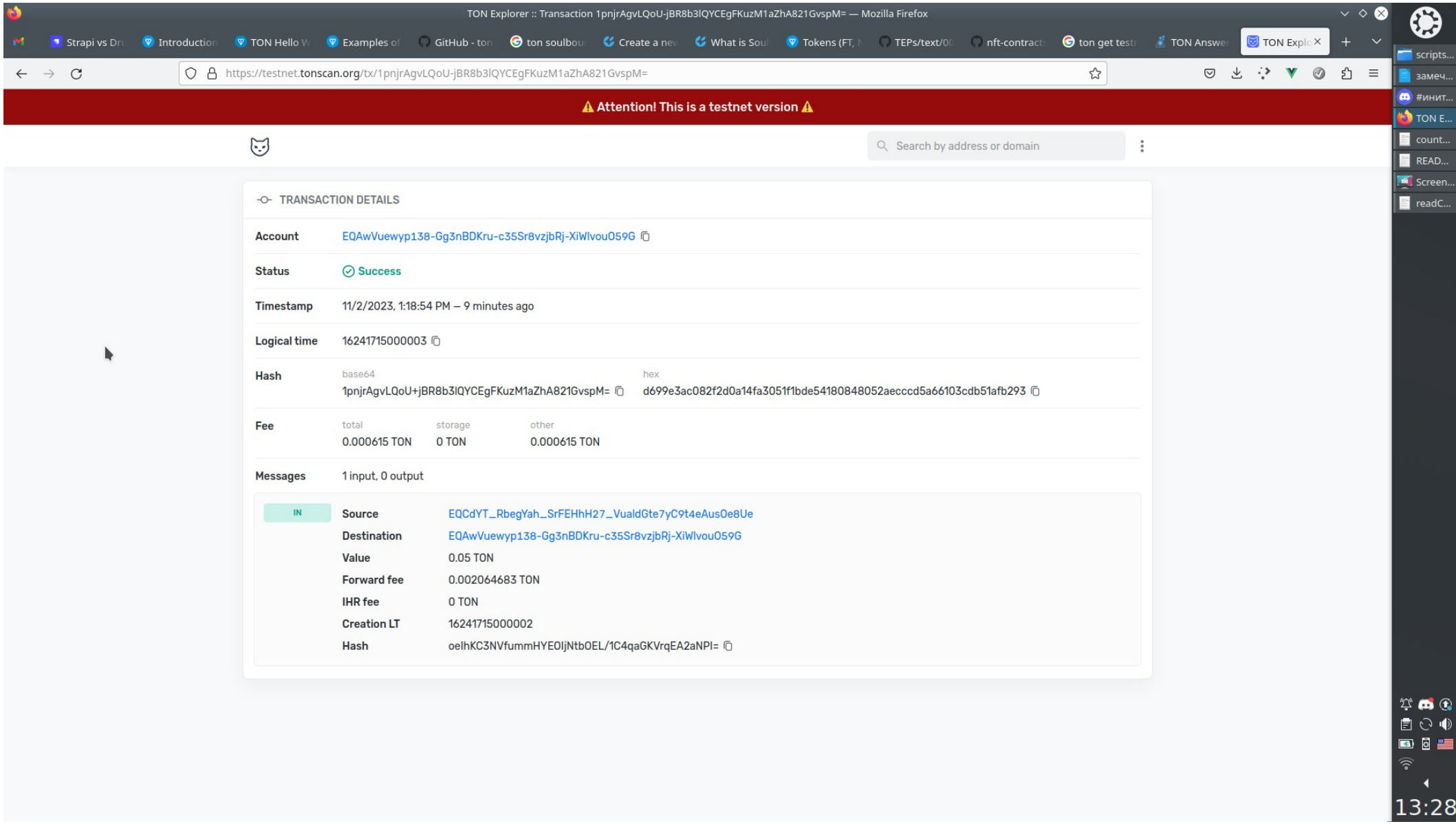
Run incrementCounter

```
Counter = 0
sin@thinkpad:~/Univer/Blockchain/TON/Counter$ npx blueprint run
? Choose file to use
? Choose file to use incrementCounter
? Which network do you want to use?
? Which network do you want to use? testnet
? Which wallet are you using? (Use arrow keys)
? Which wallet are you using? TON Connect compatible mobile wallet (example: Tonkeeper)
Connected to wallet at address: EQCdYT_RbegYah_SrFEHhH27_VualdGte7yC9t4eAusOe8Ue
? Counter address EQAwVuewyp138-Gg3nBDKru-c35Sr8vzjbRj-XiWlvouO59G
Sending transaction. Approve in your wallet...[TON_CONNECT_SDK] Send http-bridge request: {
  method: 'sendTransaction',
  params: [
    '{"messages":[{"address":"EQAwVuewyp138-Gg3nBDKru-c35Sr8vzjbRj-XiWlvouO59G","amount":"50000000","payload":"te6c
ckEBAQEAEgAAIH6HZO8AAAAAAAAAAAAAAAFixPnX"}],"valid_until":1698921088185,"from":"0:9d613fd16de8186a1fd2ac5107847dbbf
d5b9a95d1ad7bbc82f6de1e02eb0e7b","network":"-3"}'
  ],
  id: '1'
}
[TON_CONNECT_SDK] Wallet message received: {
  result: 'te6cckEBAgEAugAB4YgBOsJ/otvQMNQ/pViiDwj7d/q3NSujWvd5Be28PAXWHPYAaug1bxxMUyGcontxJdnyZxgKEugjDcftiSE9n7OC
C8C6u1ItDBFzXgufCKnQJqjrystxumdJCWF6P2Pjff5wAU1NGLsqG81oAAAACAAcAQCIYgAYK3PYZU67+fDQbzghlV3fOb8pV+X5xtox/LxLS30XHaA
X14QAAAAAAAAAAAAAAAAAAH6HZO8AAAAAAAAAAAAAAFeajFi',
  id: '1'
}
Sent transaction
Waiting for counter to increase...
Counter increased successfully!
sin@thinkpad:~/Univer/Blockchain/TON/Counter$
```

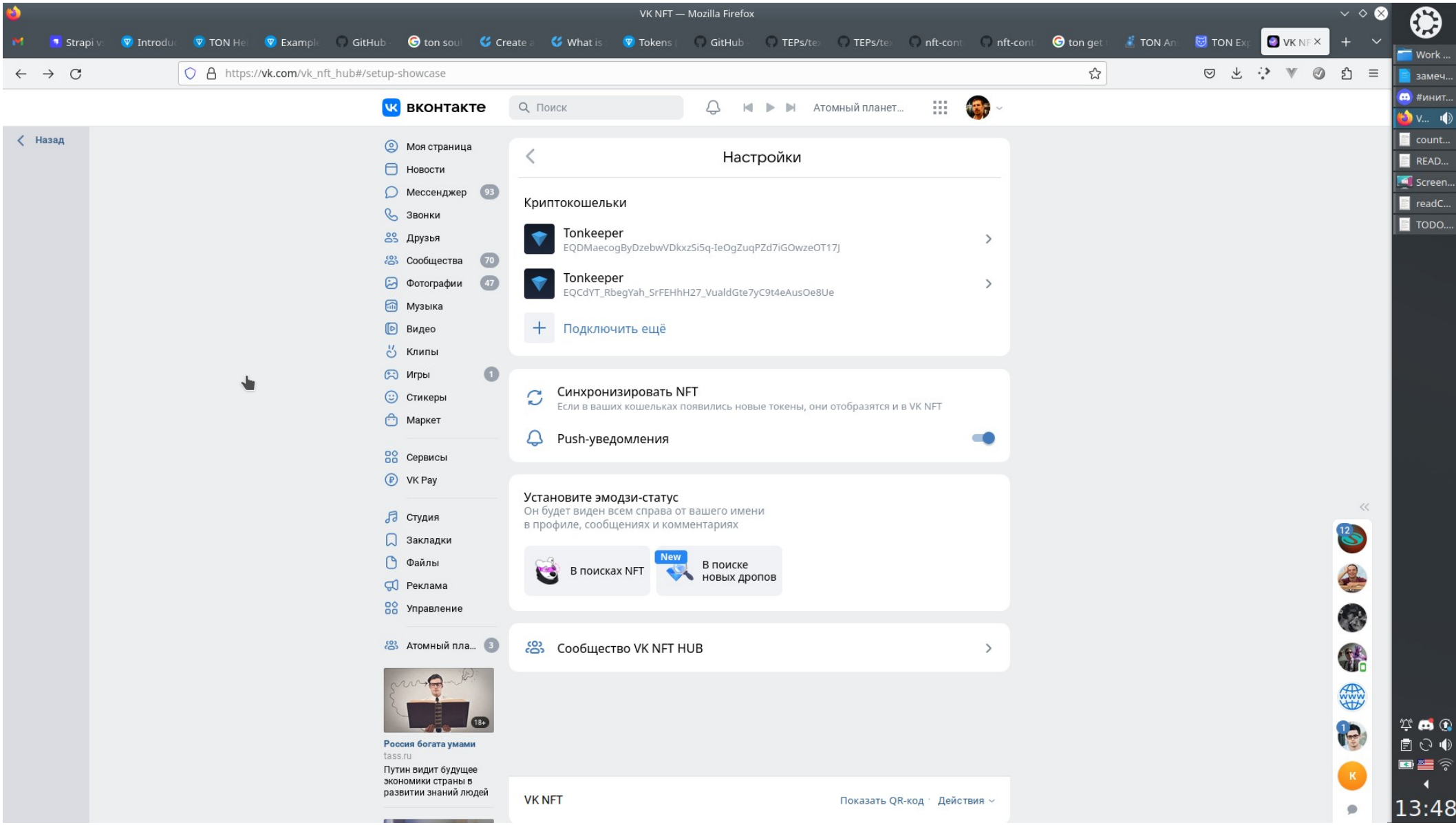
Shell

Counter : bash - Drop-Down Terminal

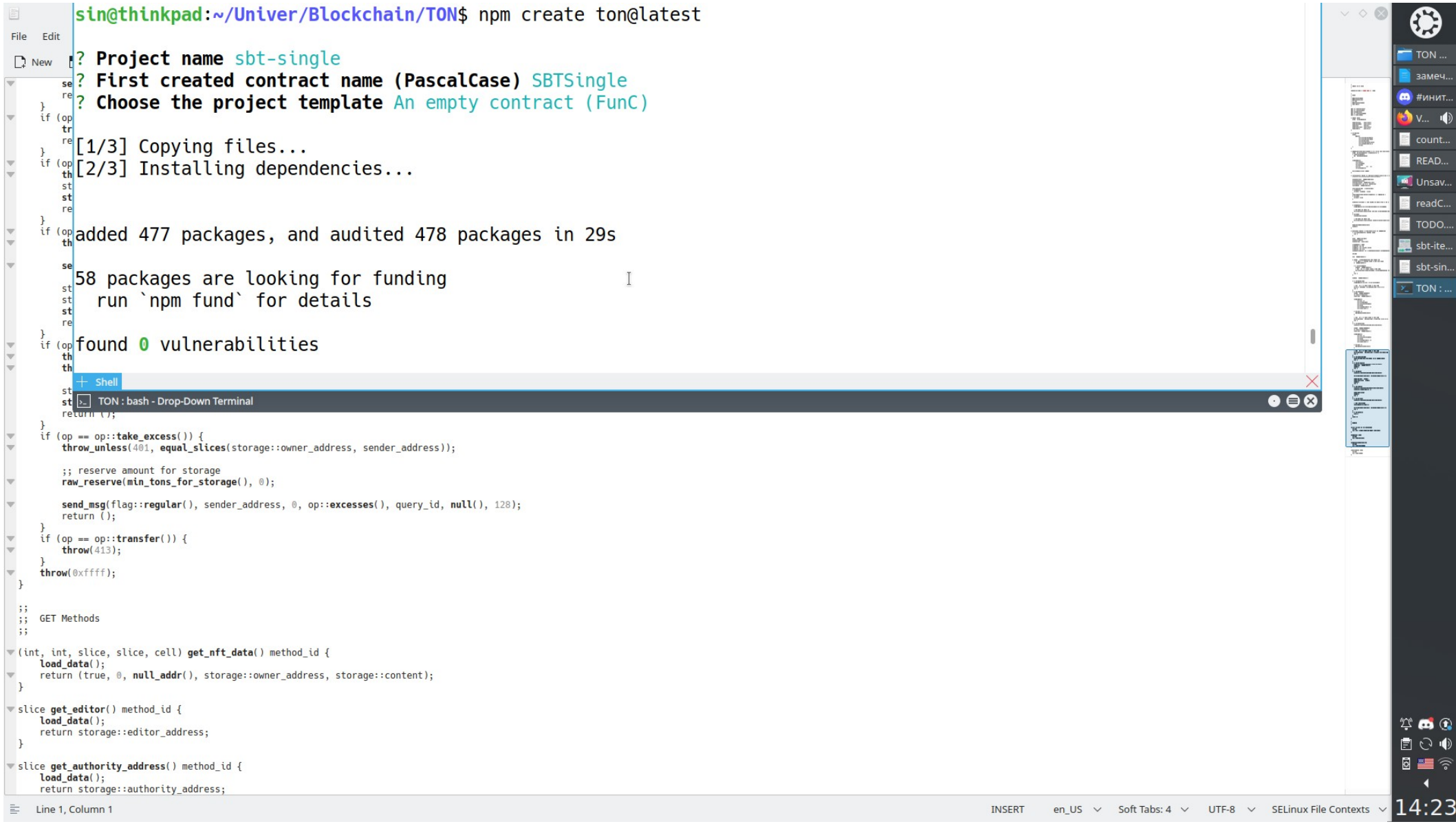
Check in Blockchain Explorer



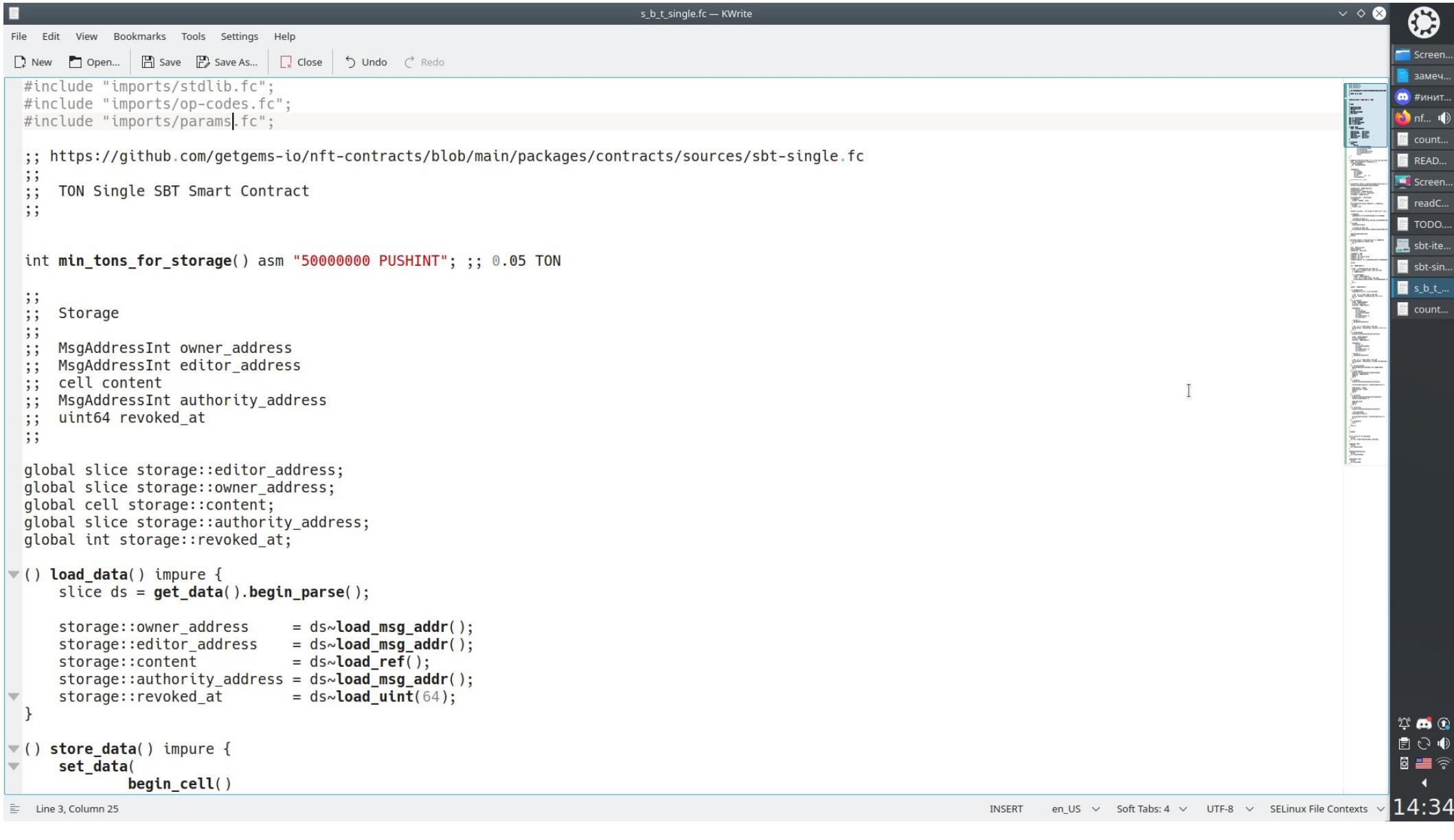
Add Address to VK



Create soulbound NFT Token



Steal code from GetGems

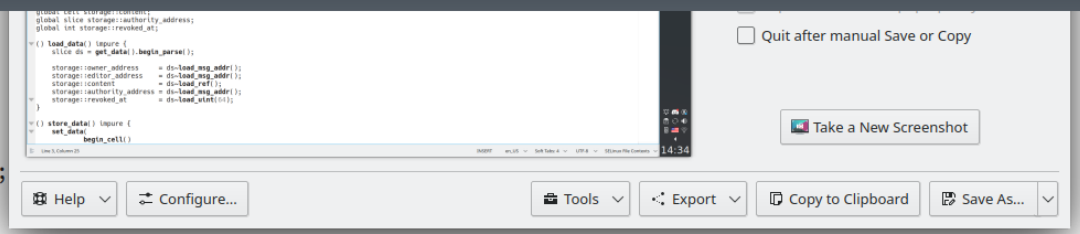


Build

```
sin@thinkpad:~/Univer/Blockchain/TON/sbt-single$ npx blueprint build
Using file: SBTSingle
Build script running, compiling SBTSingle
⌛ Compiling...
✅ Compiled successfully! Cell BOC hex result:

b5ee9c7241021901000418000114ff00f4a413f4bcf2c80b0102016209020201200403001 9bc7e7f8013fb845817c217c21c020158080502012
00706000db360fc00be1160000db01d3c00be1060000db5631e005f08900202ce0d0a0201200c0b002f3e117e10f23e10b3c5be1073c5b33e11
33c5b2cff27b552000373b51343e90007e18be90007e1875007e18fe90007e1934cfcc3e19600201200f0e00113e910c1c2ebcb8536001ed0c8
871c02497c0f83434c0fe903e900c7e800c5c75c87e800c7e800c3c0080f4c7c0dc6c2393040917c134c7e084014931eb84aea38e74cfcc2004
3e109c20843063a1b49540db601000f232c7c4b2cfc85bace4c073c5e44c78b25c417232c1540133c5963e8084f2dab332407ec0244c38b800b
4cfe01004fc82102fcb26a25240ba8e40306c223270c8cbff8b02cf1680107082108b771735405503804003c8cb1f12cb3f216eb39301cf1791
31e2c97105c8cb055004cf1658fa0213cb6accc901fb00e08210d0c3bfea5240bae302821004ded1485240bae30282101c04412a5240ba8e853
34003db3ce0343482101a0b9d515220ba1817151103fe8e103132f84112c705f2e19ad430f863f003e03282101f04537a5210ba8e4730f84221
c705f2e1918010708210d53276db41046d830603c8cb1f12cb3f216eb39301cf179131e2c97105c8cb055004cf1658fa0213cb6accc901fb008
b02f8628b02f864f003e082106f89f5e35210bae3028210d136d3b35210bae3026c21141312002082105fcc3d14ba93f2c19dde840ff2f0008a
```

Shell

sbt-single : bash - Drop-Down Terminal




```
;; uint64 revoked_at
;;

global slice storage::editor_address;
global slice storage::owner_address;
global cell storage::content;
global slice storage::authority_address;
global int storage::revoked_at;

() load_data() impure {
    slice ds = get_data().begin_parse();

    storage::owner_address      = ds~load_msg_addr();
    storage::editor_address     = ds~load_msg_addr();
    storage::content            = ds~load_ref();
    storage::authority_address  = ds~load_msg_addr();
    storage::revoked_at         = ds~load_uint(64);
}

() store_data() impure {
    set_data(
            begin_cell()
```

Line 3, Column 25 INSERT en_US Soft Tabs: 4 UTF-8 SELinux File Contexts

14:35

Deploy

```
sin@thinkpad:~/Univer/Blockchain/TON/sbt-single$ npx blueprint run
Using file: deploySBTSingle
? Which network do you want to use? testnet
? Which wallet are you using? TON Connect compatible mobile wallet (example: Tonkeeper)
? Choose your wallet Tonkeeper
```

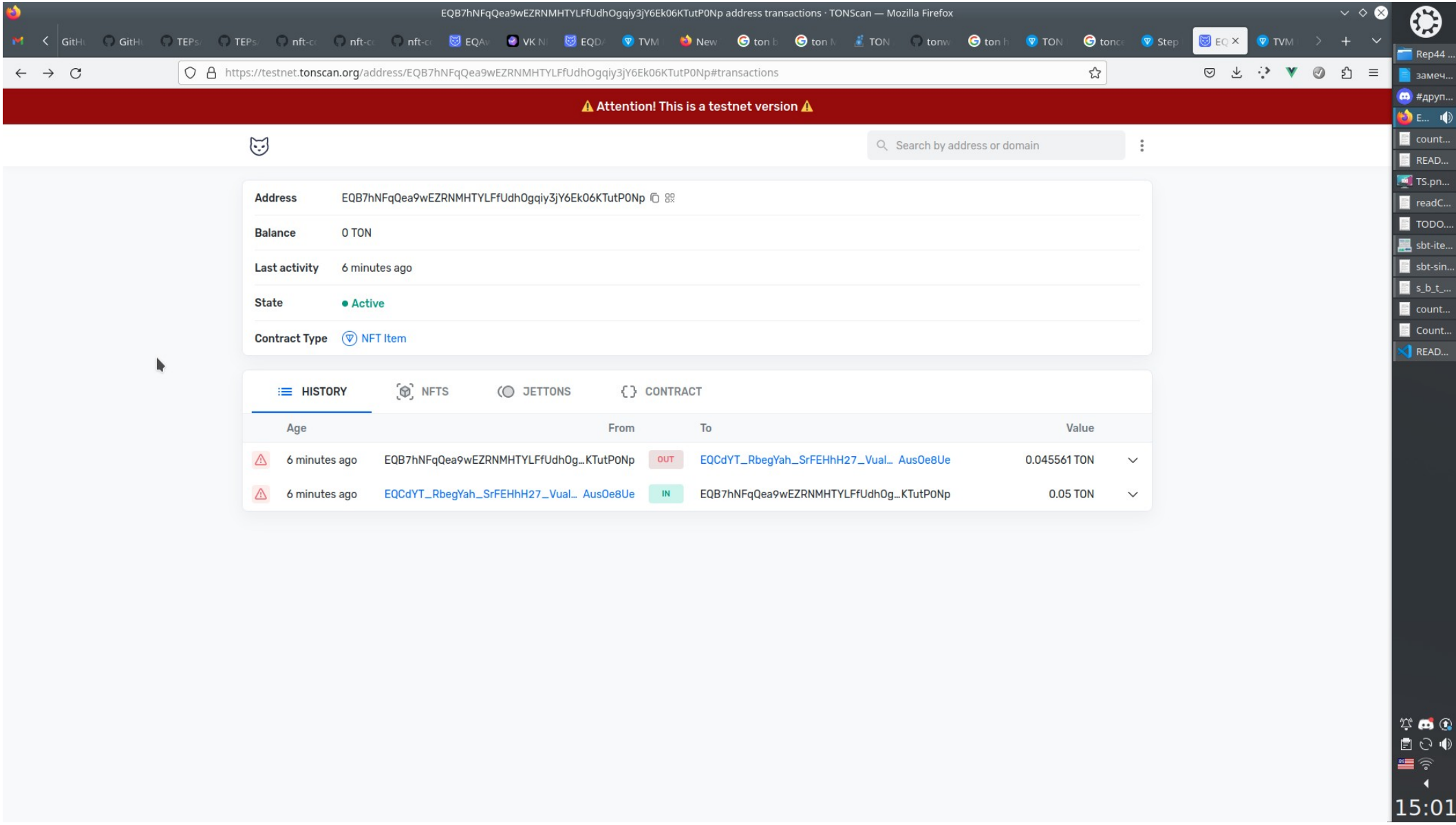
Deploy Result


```
","stateInit":"te6cckECGwEABB8AAgE0AgEAAAEU/wD0pBP0vPLICwMCAWILBAIBIAYFABm8fn+AE/uEWBfCF8IcAgFYCgcCASAJCAANs2D8AL4RYAANsB08AL4QYAANtWMe
AF8IkAICzg8MAgEgDg0ALz4RfhDyPhCzxb4Qc8WzPhEzxbLP8ntVIAA3O1E0PpAAfhi+kAB+GHUAfhj+kAB+GTTPzD4ZYAIBIBEQABE+kQwcLry4U2AB7QyIccAkl8D4NDTA/pA
+kAx+gAxcdch+gAx+gAw8AID0x8DcbCOTBAkXwTTH4IQBSTHrhK6jjnTPzCAEPhCcIIQwY6G0lUDbYBAA8jLHxLLPyFus5MBzxeRMeLJcQXIywVQBM8WWPoCE8tqzMkB+wCRMOL
gAtM/gEgT8ghAvyyaiUkC6jkAwbCIycMjL/4sCzxaAEHCCEIt3FzVAVQOAQAPIyx8Syz8hbrOTAc8XkTHiyXEFyMsFUATPFlj6AhPLaszJAfsA4IIQ0MO/6lJAuuMCghAE3tFIU
kC64wKCEBwEQSpSQLqOhTNAA9s84DQ0ghAaC51RUiC6GhkXEwP+jhAxMvhBEscF8uGa1DD4Y/AD4DKCEB8EU3pSELqORzD4QiHHBfLhkYAQcIIQ1TJ220EEbYMGA8jLHxLLPyFu
s5MBzxeRMeLJcQXIywVQBM8WWPoCE8tqzMkB+wCLAvhiiwL4ZPAD4IIQb4n141IQuuMCghDRNtOzUhC64wJsIRYVFAAgghBfzD0UupPywZ3ehA/y8ACKMPhCIccF8uGRggr68IB
w+wKAEHCCENUydttBBG2DBgPIyx8Syz8hbrOTAc8XkTHiyXEFyMsFUATPFlj6AhPLaszJAfsAAC4wMfhEAccF8uGR+EXAAPLhk/gj+GXwAwH0+EEUxwXy4ZH6QCHwAfpA0gAx+g
CCCvrwgBehIZRTFaCh3iLXCwHDACCSBqGRNuIgwv/y4ZIhjj3I+EHPFlAHzxaAEIIQURpEYxNxJlRIUAPIyx8Syz8hbrOTAc8XkTHiyXEFyMsFUATPFlj6AhPLaszJAfsAkjYw4
gMYAICONiLwAYAQghDVMnbbFEUDbXEDyMsfEss/IW6zkwHPF5Ex4slxBcjLBVAEzxZY+gITy2rMyQH7AJJsMeL4YfADAMhsM/hCUAPHBfLhkQH6QNTTADD4RXDIy//4Qs8WE8wS
yz9SEMsAAcMAlPhDAczegBB4sXCCEAUkx65AVQOAQAPIyx8Syz8hbrOTAc8XkTHiyXEFyMsFUATPFlj6AhPLaszJAfsAAMBsM/pA1NMAMPhFcMjL/1AGzxb4Qs8WEswUyz9SMMs
AA8MAlvhDUAPMAt6AEHixcIIQDdYH40A1FIBAA8jLHxLLPyFus5MBzxeRMeLJcQXIywVQBM8WWPoCE8tqzMkB+wAuz+Qv"}],"valid_until":1698925354501,"from":"0:
9d613fd16de8186a1fd2ac5107847dbbfd5b9a95d1ad7bbc82f6de1e02eb0e7b","network":"-3"}'
  ],
  id: '0'
}
[TON_CONNECT_SDK] Wallet message received: {
  result: 'te6cckECHQEABMsAAeGIATrCf6Lb0DDUP6VYog8I+3f6tzUro1r3eQXtvDwF1hz2ABmyqf3y3ohB2Ee0mR9X/OC052juJ7JOD7wwACYXCXWQG4nK/o8AeqLvayLi
T+C7mfahypCCSBQv6O8Y/T8ekBFNTRi7KhxSQAAAABAAHAEBaWIAYCmIkH4eympyxzinBnoX5OjNbIpfuXdr1UA5UqPkiZ4gF9eEAAAAAAAAAAAAAAAAAAAAAANAAgIBNAMcART/APS
kE/S88sgLBAIBYgUVAgLOBhICASAHEQHtDIhxwCSXwPg0NMD+kD6QDH6ADFx1yH6ADH6ADDwAgPTHwNxsI5MECRfBNMfghAFJMeuErqOOdM/MIAQ+EJwghDBjobSVQNtgEADyMs
fEss/IW6zkwHPF5Ex4slxBcjLBVAEzxZY+gITy2rMyQH7AJEw4uAC0z+AIBPyCEC/LJqJSQLqOQDBsIjJwyMv/iwLPFoAQcIIQi3cXNUBVA4BAA8jLHxLLPyFus5MBzxeRMeLJc
QXIywVQBM8WWPoCE8tqzMkB+wDgghDQw7/qUkC64wKCEATe0UhSQLrjAoIQHARBKlJAuo6FM0AD2zzgNDSCEBoLnVFSILoJCgsNAMBsM/pA1NMAMPhFcMjL/1AGzxb4Qs8WEswU
yz9SMMsAA8MAlvhDUAPMAt6AEHixcIIQDdYH40A1FIBAA8jLHxLLPyFus5MBzxeRMeLJcQXIywVQBM8WWPoCE8tqzMkB+wAAyGwz+EJQA8cF8uGRAfpA1NMAMPhFcMjL//hCzxY
TzBLLP1IQywABwwCU+EMBzN6AEHixcIIQBSTHrkBVA4BAA8jLHxLLPyFus5MBzxeRMeLJcQXIywVQBM8WWPoCE8tqzMkB+wAB9PhBFMcF8uGR+kAh8AH6QNIAMfoAggr68IAXoS
GUUxWgod4i1wsBwwAgkgahkTbiIML/8uGSIY49yPhBzxZQB88WgBCCEFEaRGMTcSZUSFADyMsfEss/IW6zkwHPF5Ex4slxBcjLBVAEzxZY+gITy2rMyQH7AJI2MOIDDACAjjYi8
AGAEIIQ1TJ22xRFA21xA8jLHxLLPyFus5MBzxeRMeLJcQXIywVQBM8WWPoCE8tqzMkB+wCSbDHi+GHwAwP+jhAxMvhBEscF8uGa1DD4Y/AD4DKCEB8EU3pSELqORzD4QiHHBfLh
kYAQcIIQ1TJ220EEbYMGA8jLHxLLPyFus5MBzxeRMeLJcQXIywVQBM8WWPoCE8tqzMkB+wCLAvhiiwL4ZPAD4IIQb4n141IQuuMCghDRNtOzUhC64wJsIQ4PEAAuMDH4RAHHBfL
hkfhFwADy4ZP4I/hl8AMAijD4QiHHBfLhkYIK+vCAcPsCgBBwghDVMnbbQQRtgwYDyMsfEss/IW6zkwHPF5Ex4slxBcjLBVAEzxZY+gITy2rMyQH7AAAgghBfzD0UupPywZ3ehA
/y8AARPpEMHC68uFNgAgEgExQANztRND6QAH4YvpAAfhh1AH4Y/pAAfhk0z8w+GWAALz4RfhDyPhCzxb4Qc8WzPhEzxbLP8ntVIAIBIBIBYbAgFYFxgADbVjHgBfCJACASAZGgANs
B08AL4QYAANs2D8AL4RYAAZvH5/gBP7hFgXwhfCHAAACuamtQ==',
  id: '0'
}
Sent transaction
Contract deployed at address EQDAUxEg_D2U1OWOcU4M9C_J0ZrZFL9y7teqgHKlR8kTPEeg
You can view it at https://testnet.tonscan.org/address/EQDAUxEg_D2U1OWOcU4M9C_J0ZrZFL9y7teqgHKlR8kTPEeg
sin@thinkpad:~/Univer/Blockchain/TON/sbt-single$
```

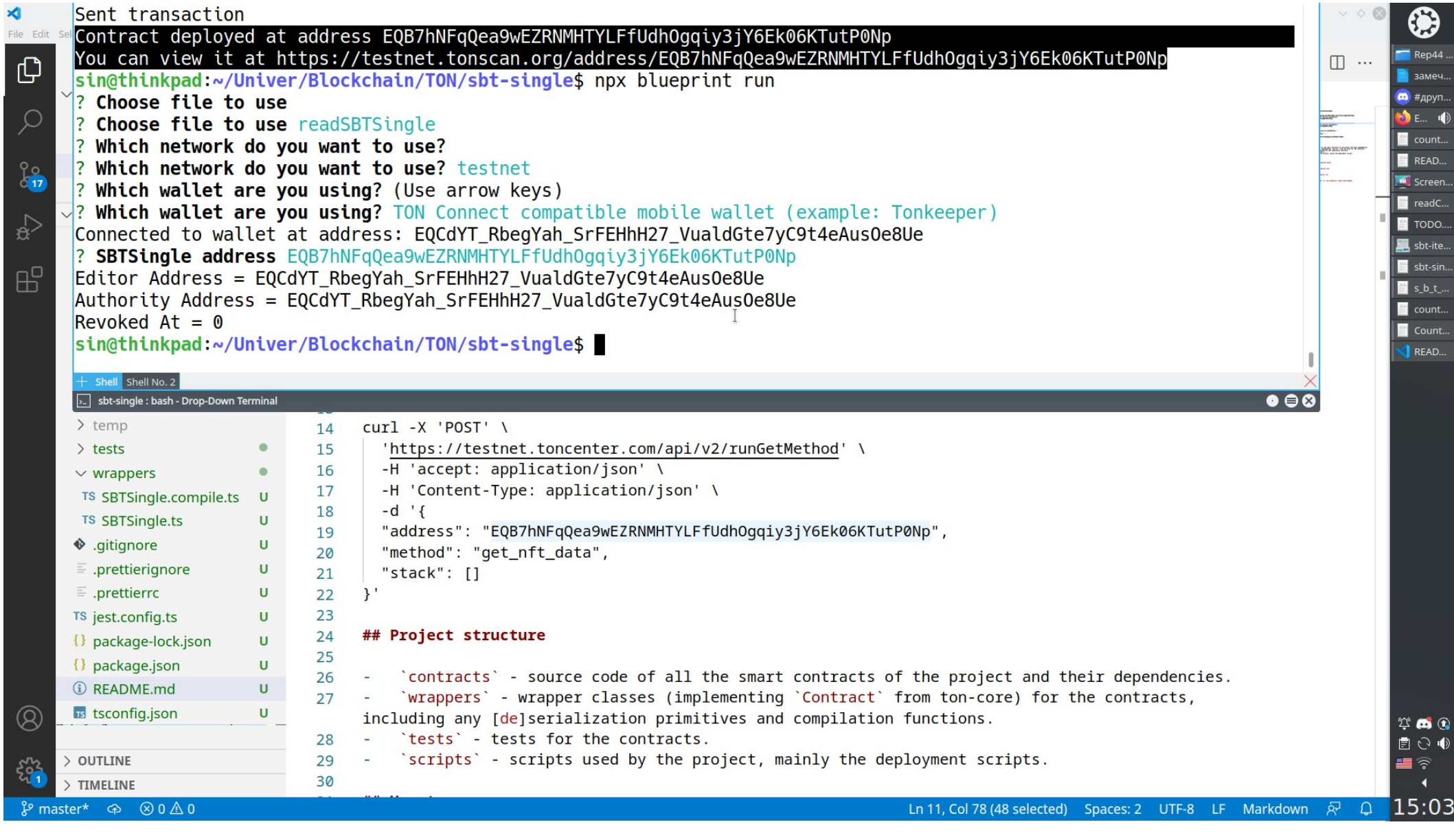

Shell

sbt-single : bash - Drop-Down Terminal

Check in Blockchain Explorer



Read Token Data



Additional Links

TON Description:
https://test.ton.org/ton.pdf

TON Blockchain Whitepaper:
https://test.ton.org/tblkch.pdf