Menu

**Join our Open Slack Community to get the latest updates from the RSK Ecosystem!**

# Create your first token

In this tutorial I will show you step-by-step how to create a token with less than 10 lines of code, using Truffle plus Open Zeppelin smart contracts, and deploy it to the RSK testnet.

## Overview

Here is a summary of the steps to be taken to build our token:

1. Initialize a project using Truffle;
2. Install Open Zeppelin smart contracts in the project folder;
3. Create a wallet mnemonic;
4. Configure Truffle to connect to RSK testnet;
5. Get some tRBTC from a faucet;
6. Create smart contract of token;
7. Create deploy file at Truffle;
8. Deploy a smart contract on RSK Testnet using Truffle;
9. Interact with the smart contract at Truffle console.

## Requirements

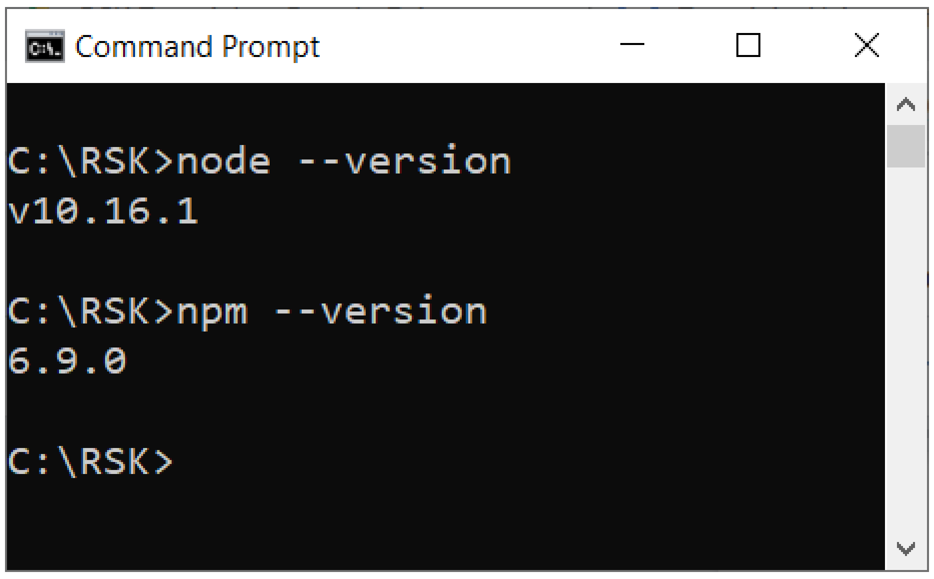
- Node.js and NPM (Node Package Manager)
- Visual Studio Code (VSCode) or any other editor of your choice
- Truffle

### Node.js and NPM

Node.js and NPM are needed, though both are usually installed at once.

NB: To check if Node.js and NPM is already installed, input the following commands in the terminal:

```
node --version
npm --version
```



```
C:\RSK>node --version
v10.16.1

C:\RSK>npm --version
6.9.0

C:\RSK>
```

Go to **Node.js** if you need to install it.

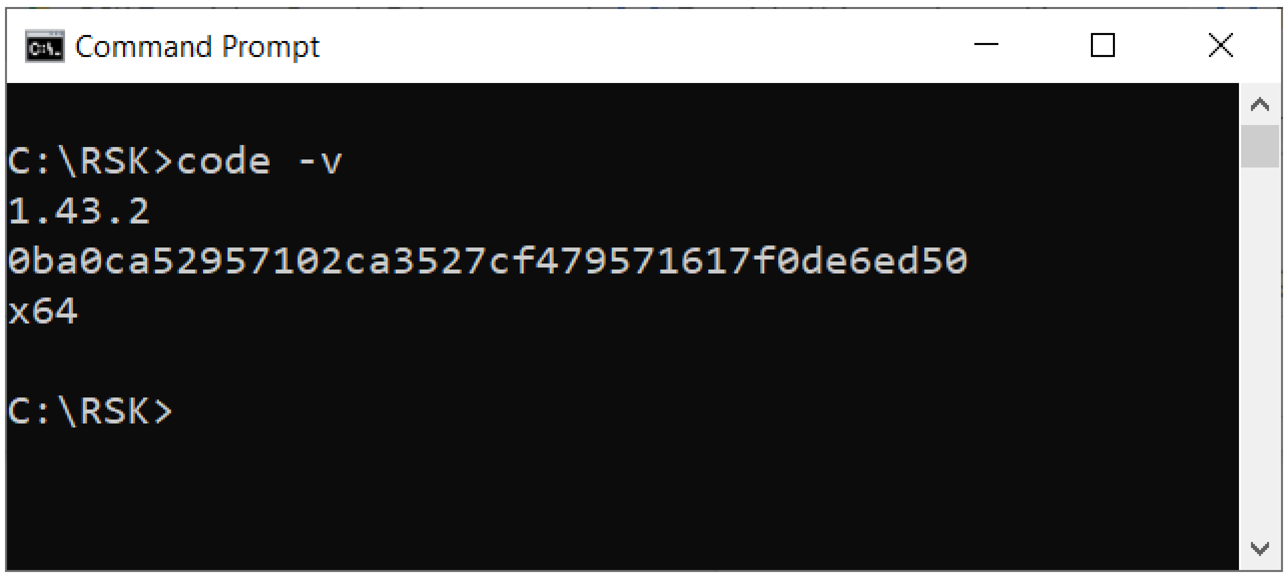
### Visual Studio Code (VSCode)

In this tutorial, we will use VSCode to create and deploy our project. Feel free to use any other code editor of your choice.

To use VSCode **download it here**.

Verify if your VS code installation was successful by typing the following command into the terminal:

```
code -v
```



```
C:\RSK>code -v
1.43.2
0ba0ca52957102ca3527cf479571617f0de6ed50
x64

C:\RSK>
```

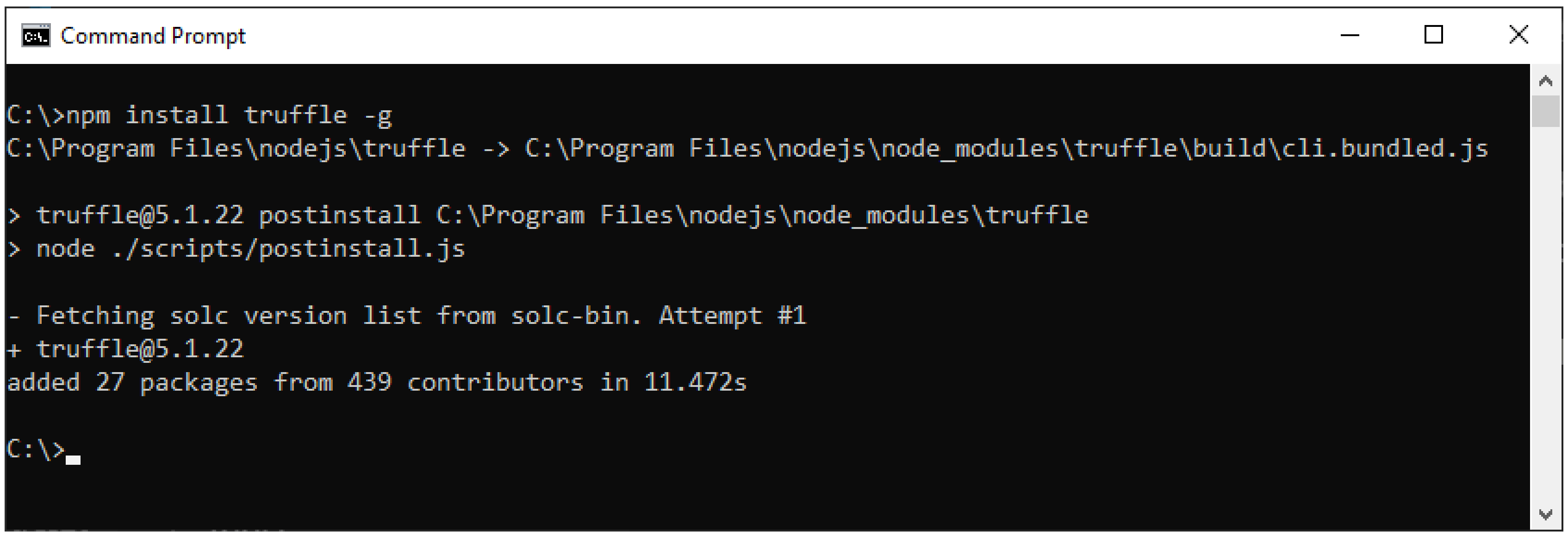
**Truffle**

Truffle is a popular development framework for smart contract developers with a mission to make your work a whole lot easier. Among its features, it has smart contract lifecycle management, scriptable deployment & migrations, automated contract testing and simple network management.

Truffle makes it easy to configure a connection to the RSK network.

To install Truffle, input the command below into the terminal and press `enter`:

```
npm install truffle -g
```

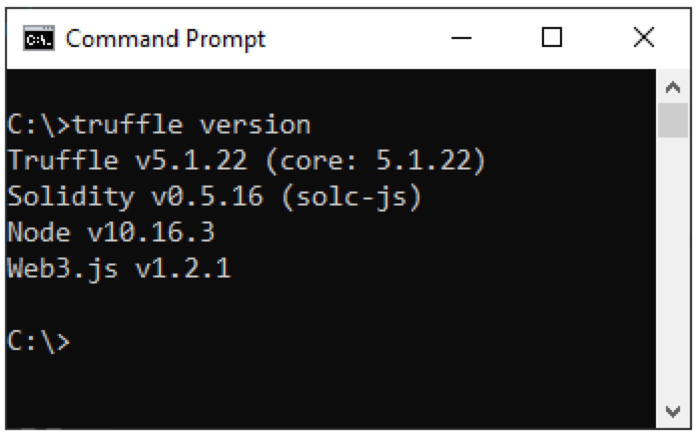


```
C:\>npm install truffle -g
C:\Program Files\nodejs\truffle -> C:\Program Files\nodejs\node_modules\truffle\build\cli.bundled.js

> truffle@5.1.22 postinstall C:\Program Files\nodejs\node_modules\truffle
> node ./scripts/postinstall.js

- Fetching solc version list from solc-bin. Attempt #1
+ truffle@5.1.22
added 27 packages from 439 contributors in 11.472s

C:\>_
```

When the installation is finished, close the terminal, open it again and check the Truffle version:

```
truffle version
```

Go to top



```
C:\>truffle version
Truffle v5.1.22 (core: 5.1.22)
Solidity v0.5.16 (solc-js)
Node v10.16.3
Web3.js v1.2.1

C:\>
```

More info:

**trufflesuite.com/truffle**

## Initialize a Truffle project

Create a new folder named `token`:

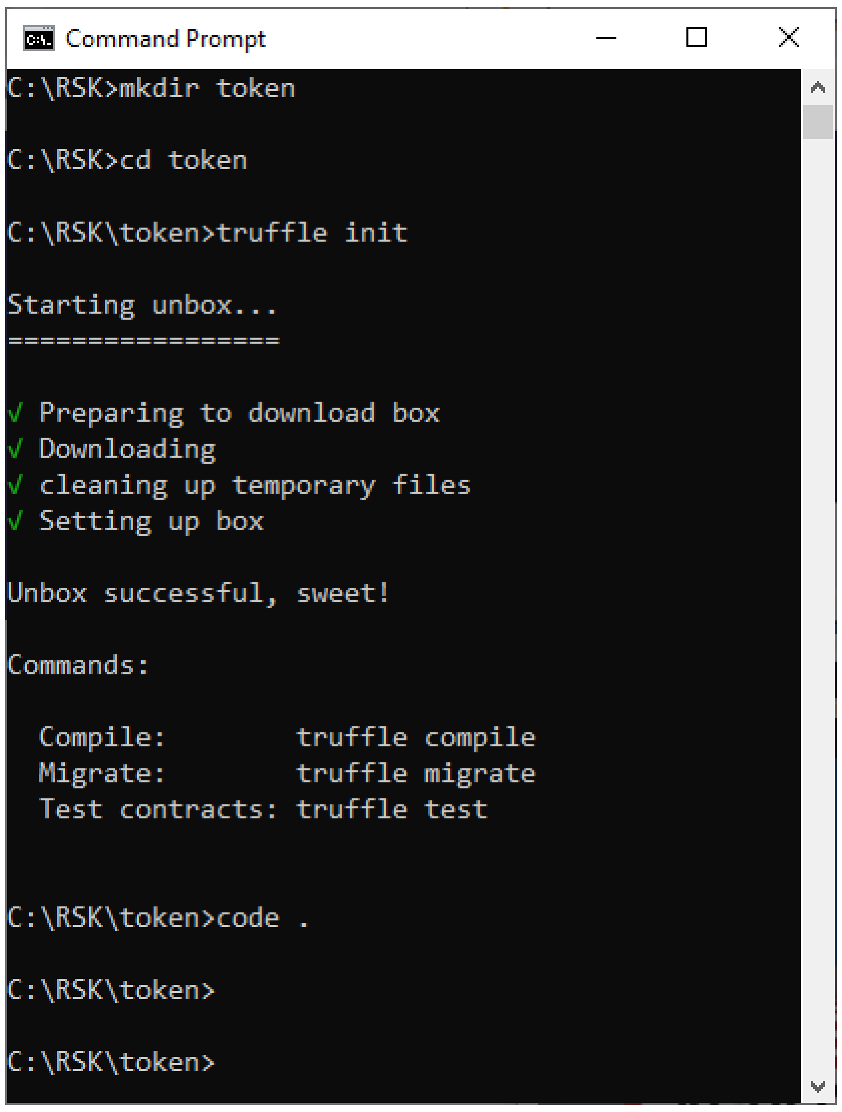
```
mkdir token
cd token
```

Start an Truffle project in the token folder by typing the following command below into the terminal:

```
truffle init
```

For example, I will create a folder at this location - `C:\RSK\` (I'm using windows OS).

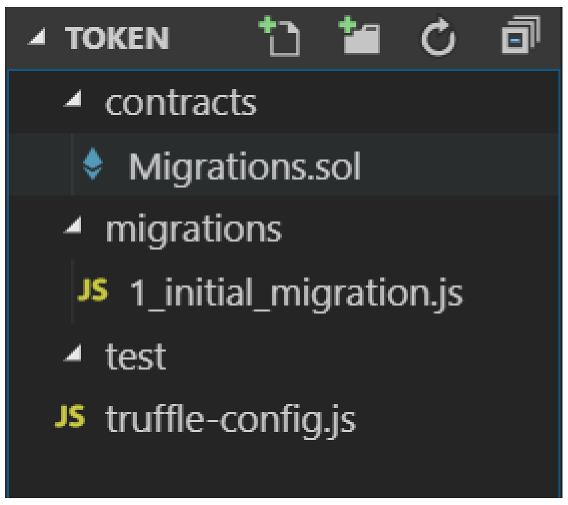
My project can be located in the folder `C:\RSK\token`.



```
C:\RSK>mkdir token

C:\RSK>cd token

C:\RSK\token>truffle init

Starting unbox...
=================

√ Preparing to download box
√ Downloading
√ cleaning up temporary files
√ Setting up box

Unbox successful, sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test


C:\RSK\token>code .

C:\RSK\token>

C:\RSK\token>
```

Open the folder in VSCode. Then you can see the file structure like this:

Go to top



```
TOKEN
  contracts
    Migrations.sol
  migrations
    1_initial_migration.js
  test
  truffle-config.js
```

- `./contracts`: All our smart contracts will be stored in this folder.
- `./migrations`: Deployment scripts will be stored in this folder.
- `./test`: Test scripts will be stored in this folder.
- `./truffle-config.js`: This is Truffle's configuration file. We'll be able to configure networks, including RSK networks.

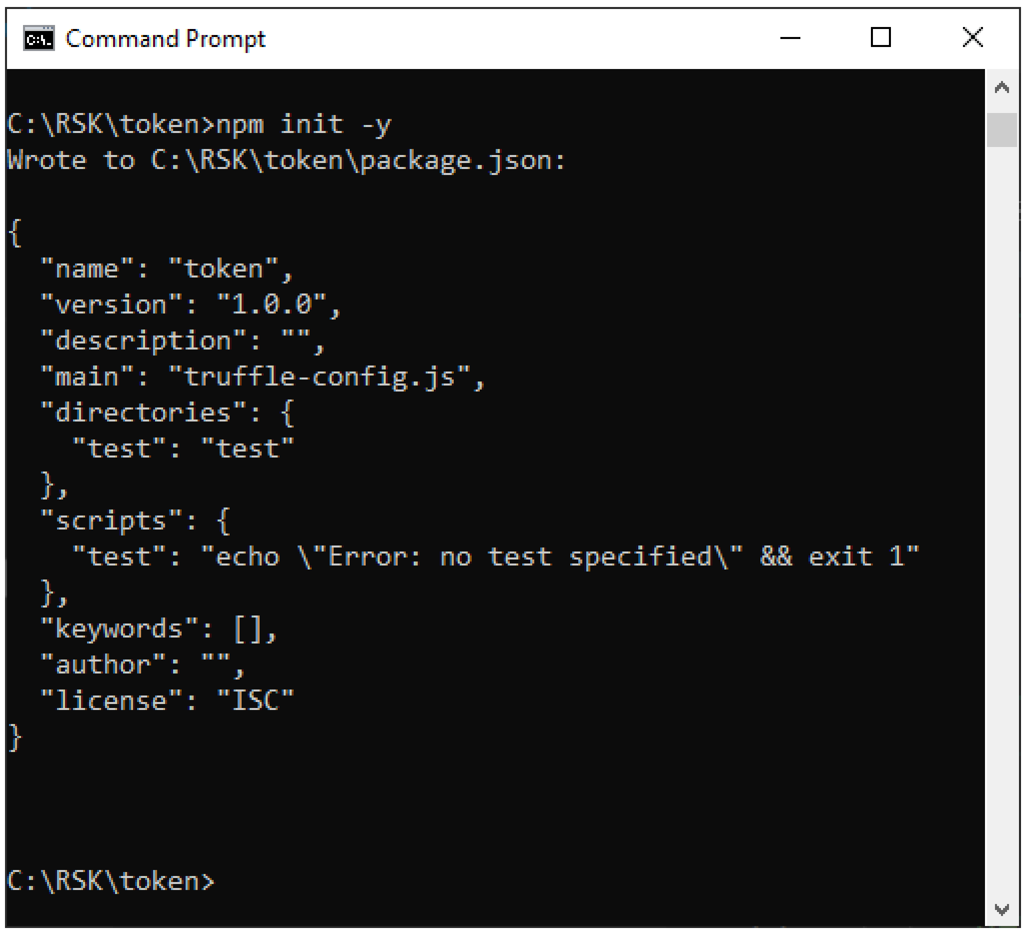
Note that the following files were also created:

- `Migrations.sol`: Keeps track of which migrations were done on the current network.
- `1_initial_migration.js`: Deployment for `Migrations.sol`.

### Initialize an npm project

Start an npm project in the token folder by typing the following command below into the terminal:

```
npm init -y
```



```
C:\RSK\token>npm init -y
Wrote to C:\RSK\token\package.json:

{
  "name": "token",
  "version": "1.0.0",
  "description": "",
  "main": "truffle-config.js",
  "directories": {
    "test": "test"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}


C:\RSK\token>
```

### Install Open Zeppelin

OpenZeppelin Contracts is a set of smart contract libraries for Ethereum. They work well with other compatible blockchains, including **RSK**. These libraries will install not only the main libraries of our token but also libraries for ownership, safe math, and many other utilities. It's worth mentioning that these libraries have been both peer reviewed and audited to accomplish high standards of security so contracts that depend on them are less susceptible to hacking when used correctly.

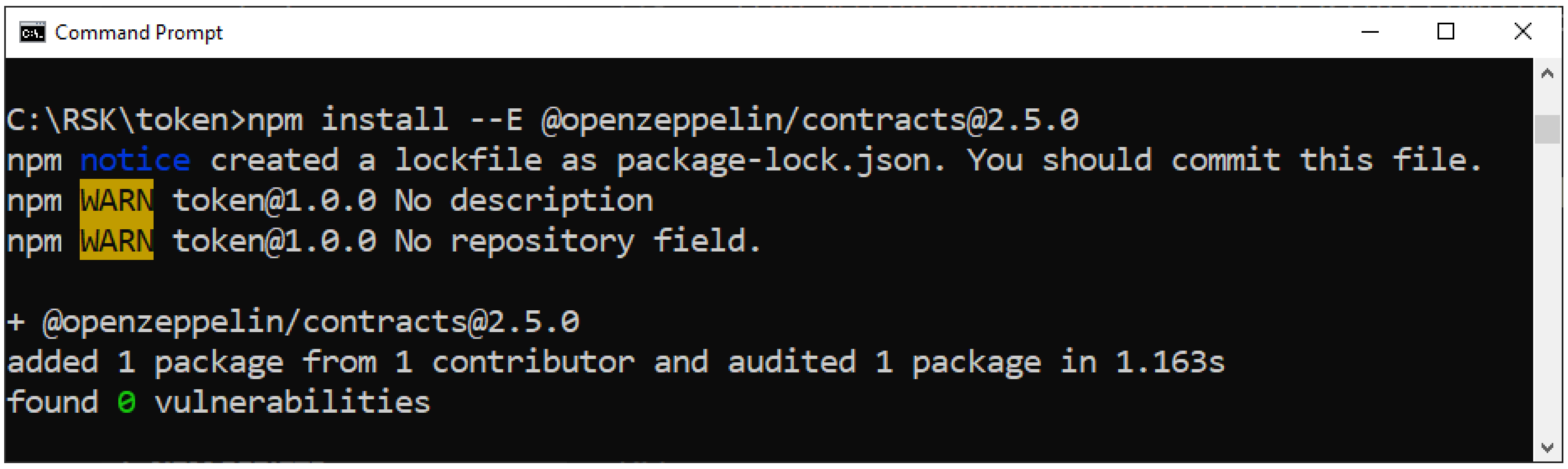
In the terminal, inside the folder token, install OpenZeppelin libraries with this command:

Go to top

```
npm install -E @openzeppelin/contracts@2.5.0
```

The option `-E` is to save dependencies with an exact version rather than using npm's default.

> *Some contracts may change over time, so it is important to set the version. This tutorial was written using the specific version gotten when we ran the `truffle version` command above.*



Command Prompt

```
C:\RSK\token>npm install --E @openzeppelin/contracts@2.5.0
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN token@1.0.0 No description
npm WARN token@1.0.0 No repository field.

+ @openzeppelin/contracts@2.5.0
added 1 package from 1 contributor and audited 1 package in 1.163s
found 0 vulnerabilities
```

More info:

[openzeppelin.com/contracts](openzeppelin.com/contracts)

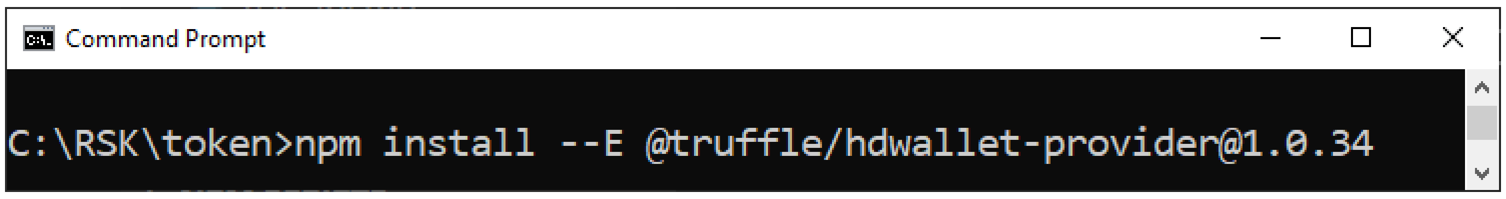
**Install HD wallet provider**

To connect to the RSK network, we are going to use a provider that allows us to connect to any network unlocking an account locally. We are going to use `@truffle/hdwallet-provider`. It can be used to sign transactions for addresses derived from a 12 or 24 word mnemonic.

> *You need to have installed Node >= 7.6.*

In the terminal, inside the folder token, install it with this command:

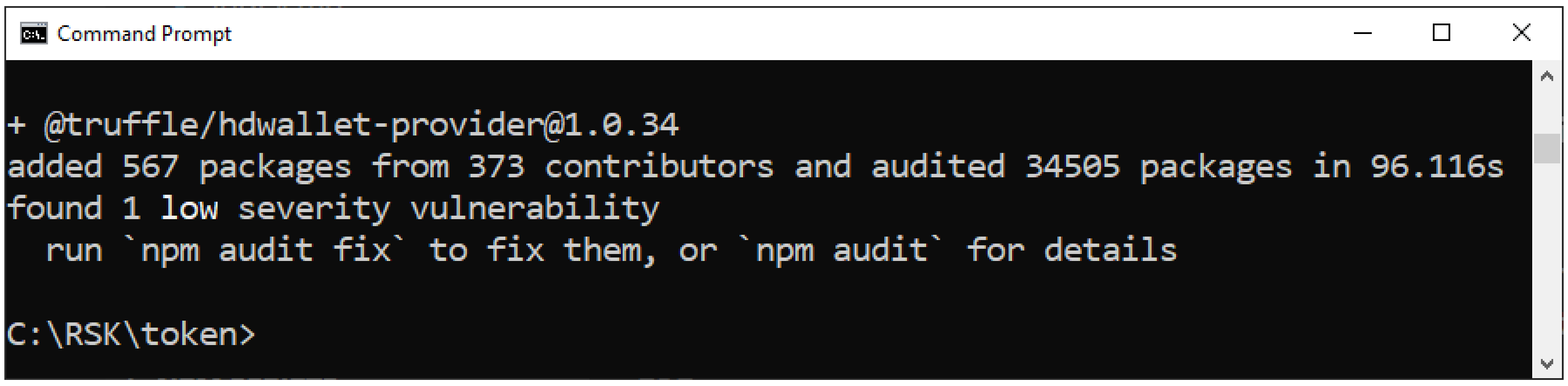
```
npm install -E @truffle/hdwallet-provider@1.0.34
```



Command Prompt

```
C:\RSK\token>npm install --E @truffle/hdwallet-provider@1.0.34
```

This `truffle` package comes with so many dependencies. A successful installation message is shown if everything works fine.



Command Prompt

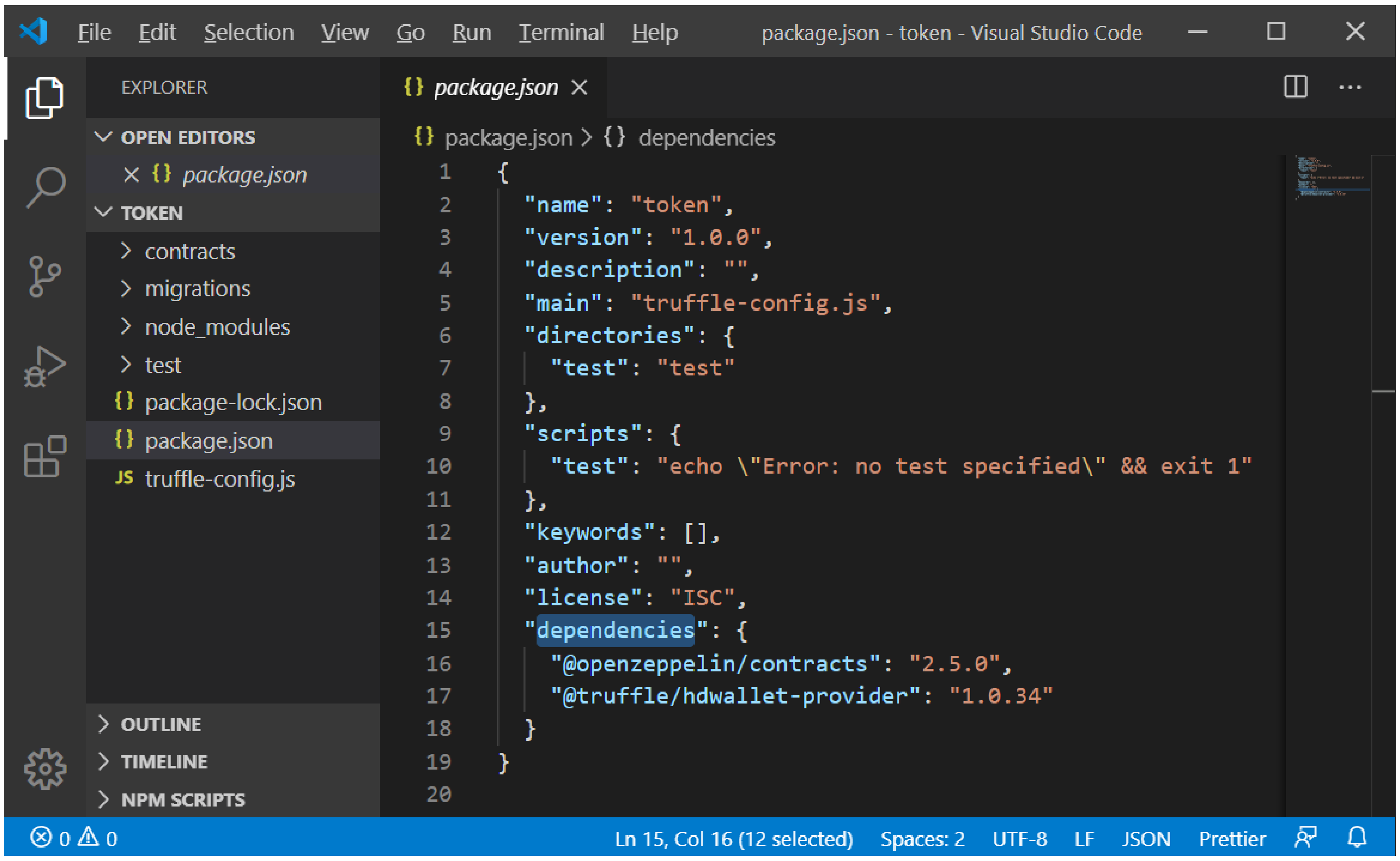
```
+ @truffle/hdwallet-provider@1.0.34
added 567 packages from 373 contributors and audited 34505 packages in 96.116s
found 1 low severity vulnerability
  run `npm audit fix` to fix them, or `npm audit` for details

C:\RSK\token>
```

**Check package.json**

Go to top

`package.json` is a file created by npm with some configurations, including the packages which we installed before using the command `npm init -y`.

After the installation, I will open the project folder named `Token` in VSCode and verify the `package.json` file. Let's take a look at the dependencies in the file:


File Edit Selection View Go Run Terminal Help
package.json - token - Visual Studio Code
EXPLORER
OPEN EDITORS
package.json
TOKEN
contracts
migrations
node_modules
test
package-lock.json
package.json
truffle-config.js
OUTLINE
TIMELINE
NPM SCRIPTS
package.json > {} dependencies
Ln 15, Col 16 (12 selected) Spaces: 2 UTF-8 LF JSON Prettier

```json
{
  "name": "token",
  "version": "1.0.0",
  "description": "",
  "main": "truffle-config.js",
  "directories": {
    "test": "test"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@openzeppelin/contracts": "2.5.0",
    "@truffle/hdwallet-provider": "1.0.34"
  }
}
```

## Verify that you can connect to RSK Testnet

Enter the following command into your terminal.

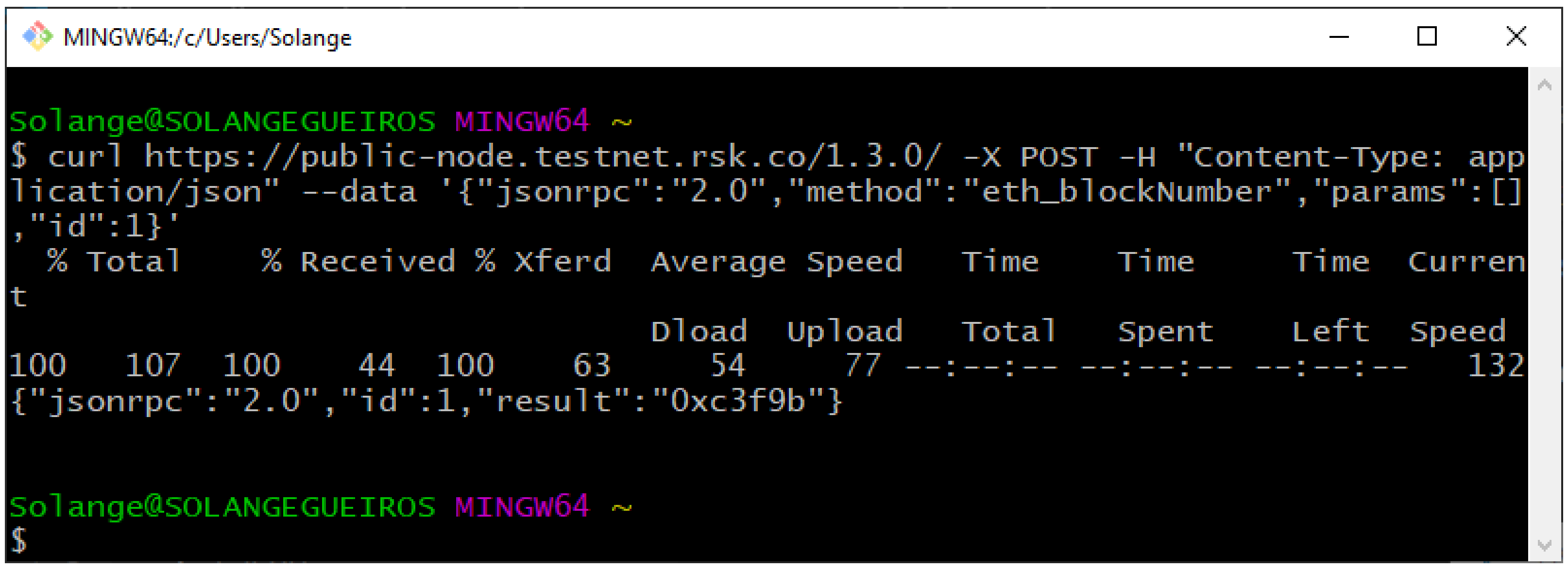
If you are using a Windows OS, I suggest to use the Git Bash terminal. Download the installer from the **Git official site**.

```
curl https://public-node.testnet.rsk.co/ \
  -X POST -H "Content-Type: application/json" \
  --data '{"jsonrpc":"2.0","method":"eth_blockNumber","params":[],"id":1}'
```

This is a command to query the network for the latest block number.

If all goes well, you'll see an output similar to the following:

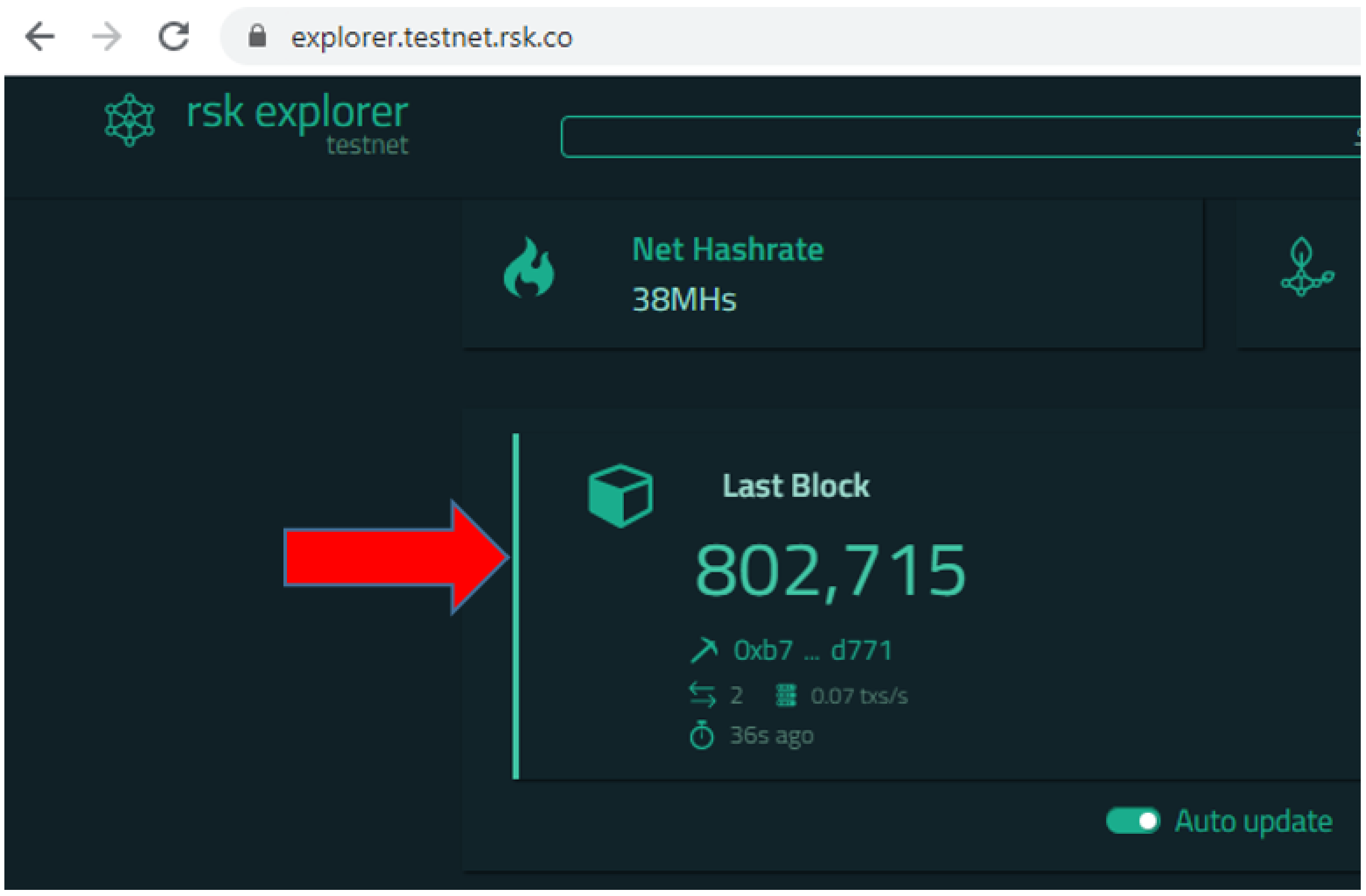
```
{"jsonrpc":"2.0","id":1,"result":"0xc3f9b"}
```



```
Solange@SOLANGEGUEIROS MINGW64 ~
$ curl https://public-node.testnet.rsk.co/1.3.0/ -X POST -H "Content-Type: app
lication/json" --data '{"jsonrpc":"2.0","method":"eth_blockNumber","params":[]
,"id":1}'
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
t
                                 Dload  Upload   Total   Spent    Left  Speed
100   107  100    44  100    63     54     77 --:--:-- --:--:-- --:--:--   132
{"jsonrpc":"2.0","id":1,"result":"0xc3f9b"}

Solange@SOLANGEGUEIROS MINGW64 ~
$
```

Go to top

The `result` value is presented in hexadecimal. `0xc3f9b` is the block number in hexadecimal, the corresponding decimal is: `802715`. To verify the block number, visit [explorer.testnet.rsk.co](explorer.testnet.rsk.co).



## Create a mnemonic

Let's create a mnemonic in order to generate some accounts.

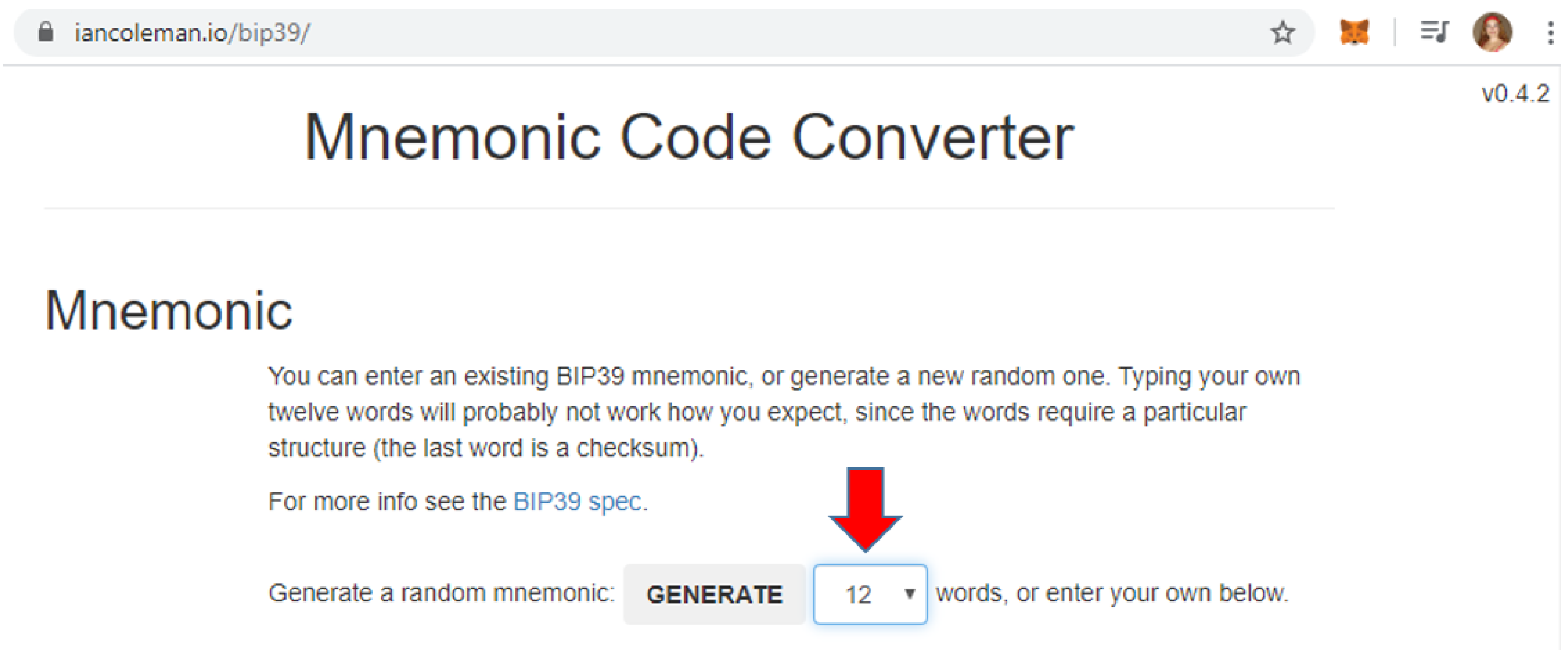
To learn more about it: **BIP39**

We are going to use this web app:

**iancoleman.io/bip39**

> *The way we are creating the mnemonic is not recommended to be used for any 'real' wallet because it's not secure generate a private key in a website, however we will use this here for learning purposes, and since we're using the Testnet anyway.*

In the `Generate a random mnemonic` field, select `12 words` and click on the `generate` button.

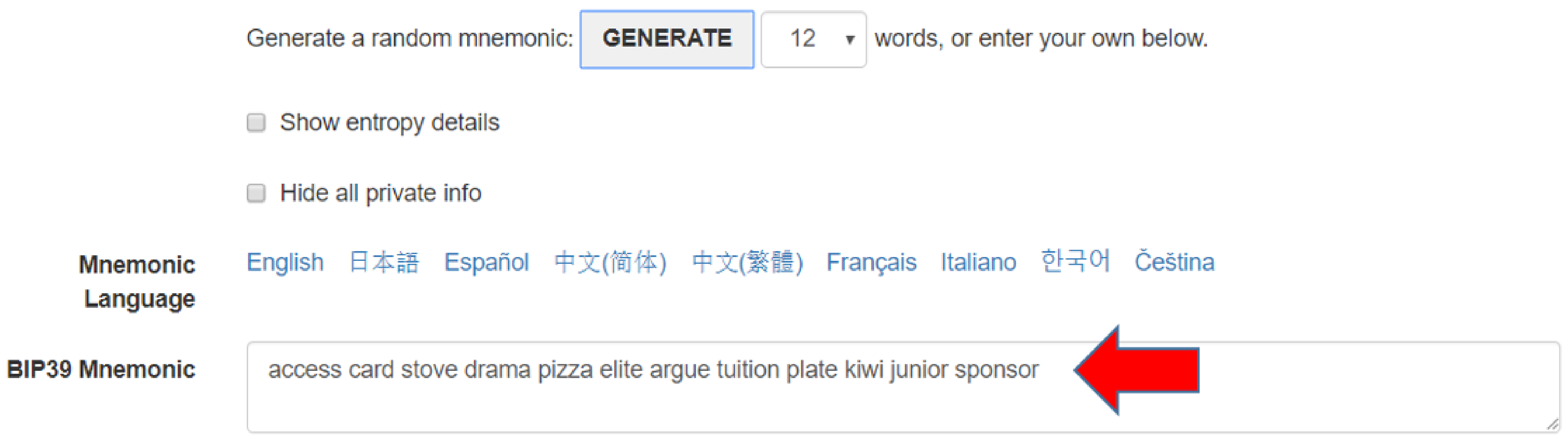


You can enter an existing BIP39 mnemonic, or generate a new random one. Typing your own twelve words will probably not work how you expect, since the words require a particular structure (the last word is a checksum).

For more info see the BIP39 spec.

Generate a random mnemonic: GENERATE 12 words, or enter your own below.

The result appears in the `BIP39 Mnemonic` field. They should be 12 random words like the words in the image:


Generate a random mnemonic: GENERATE 12 words, or enter your own below.

Show entropy details

Hide all private info

Mnemonic Language: English 日本語 Español 中文(简体) 中文(繁體) Français Italiano 한국어 Čeština

BIP39 Mnemonic: access card stove drama pizza elite argue tuition plate kiwi junior sponsor

My mnemonic is:

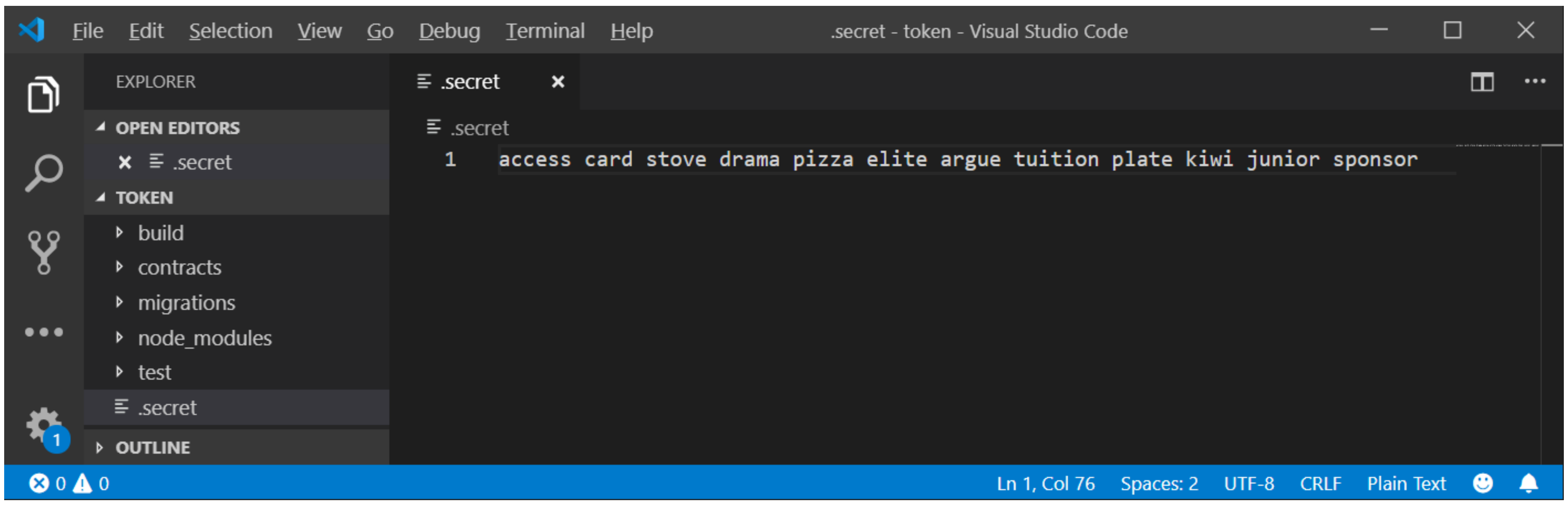
```
access card stove drama pizza elite argue tuition plate kiwi junior sponsor
```

I will copy these 12 words to use it later.

**File .secret**

Inside the `token` folder, create a file named `.secret`.

Paste your mnemonic in this file and save it.


File Edit Selection View Go Debug Terminal Help
.secret - token - Visual Studio Code
EXPLORER
OPEN EDITORS
.secret
TOKEN
build
contracts
migrations
node_modules
test
.secret
OUTLINE
Ln 1, Col 76 Spaces: 2 UTF-8 CRLF Plain Text

```
access card stove drama pizza elite argue tuition plate kiwi junior sponsor
```

## Configure Truffle to connect to RSK testnet

Open `truffle-config.js` file in your Truffle project and overwrite it with the following code:

Go to top

```
const HDWalletProvider = require('@truffle/hdwallet-provider');

const fs = require('fs');
const mnemonic = fs.readFileSync(".secret").toString().trim();

module.exports = {
  networks: {
    testnet: {
      provider: () => new HDWalletProvider(mnemonic, 'https://public-node.testnet.rsk.co/'),
      network_id: 31,
      gasPrice: 0x387EE40,
      networkCheckTimeout: 1000000000
    }
  },
  compilers: {
    solc: {
      version: "0.5.2",
    }
  }
}
```

## Truffle Console connect to RSK local network

> *You can use the Truffle console to run commands.*

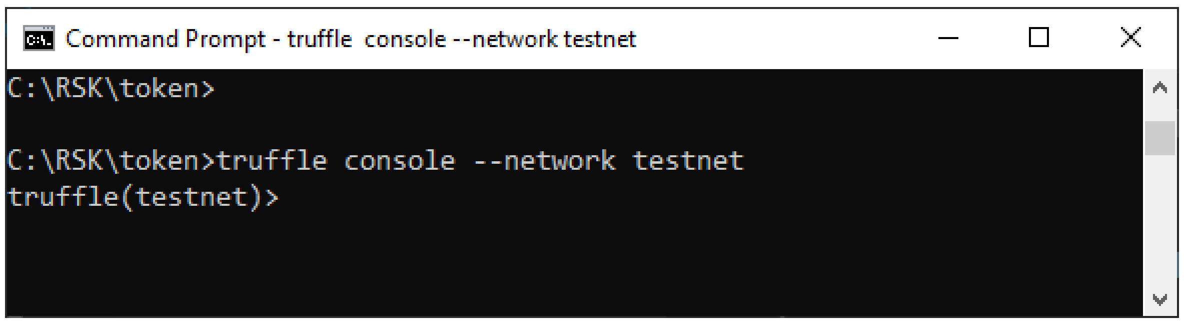
**Verify the connection**

Open a Truffle console to verify the connection.

At terminal, inside the folder token, run this command:

```
truffle console --network testnet
```

And you go to a new console:


Command Prompt - truffle console --network testnet

```
C:\RSK\token>

C:\RSK\token>truffle console --network testnet
truffle(testnet)>
```

This action instructs Truffle to connect to an RSK public testnet node and grants it permission to control the newly created account.

**To get our address**

Enter the command below into the Truffle console to save the first address generated with our mnemonic into an `account` variable:

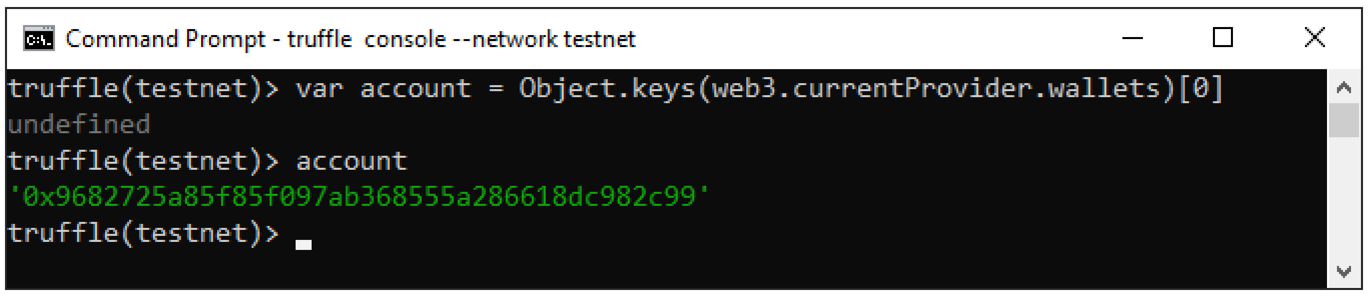
```
var account = Object.keys(web3.currentProvider.wallets)[0]
```

Go to top

The output is your account address. Enter this command to view it:

```
account
```

In my example, the output is `0x9682725a85f85f097ab368555a286618dc982c99`. Copy this address.



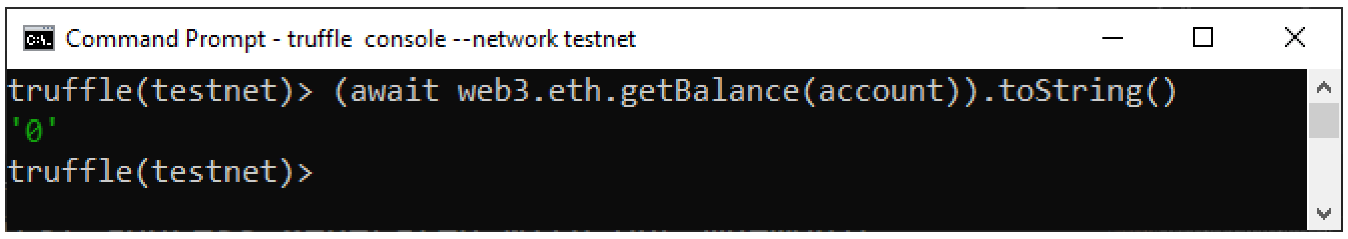
```
truffle(testnet)> var account = Object.keys(web3.currentProvider.wallets)[0]
undefined
truffle(testnet)> account
'0x9682725a85f85f097ab368555a286618dc982c99'
truffle(testnet)>
```

### Check balance

To check the balance, run this command in Truffle console:

```
(await web3.eth.getBalance(account)).toString()
```

The balance is 0 and we need some to pay for gas fees. We shall get some tRBTC in the next step.

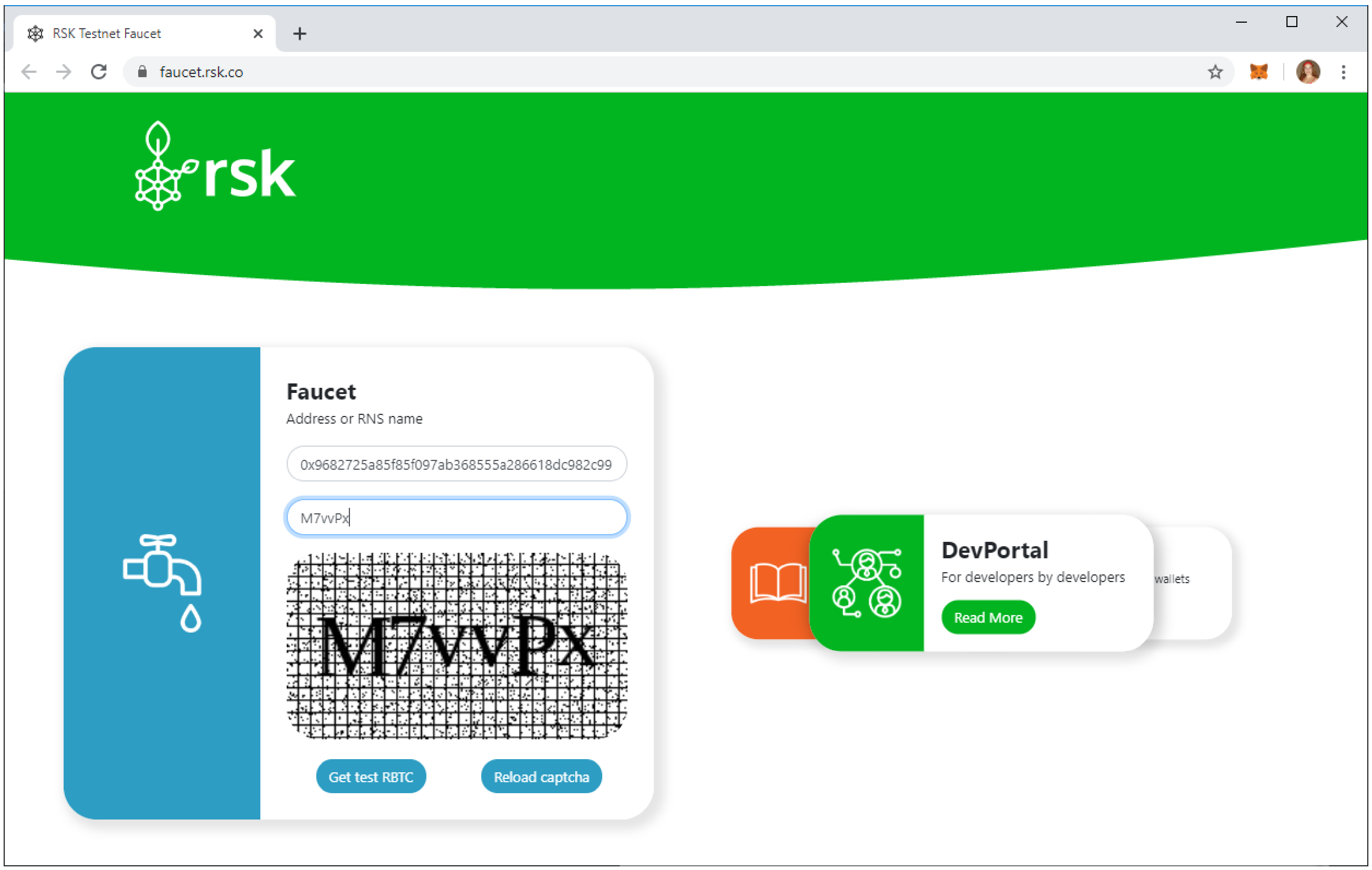


```
truffle(testnet)> (await web3.eth.getBalance(account)).toString()
'0'
truffle(testnet)>
```

## Testnet Faucet

You can get some tRBTC from the RSK Testnet faucet.

faucet.testnet.rsk.co



RSK Testnet Faucet

faucet.rsk.co

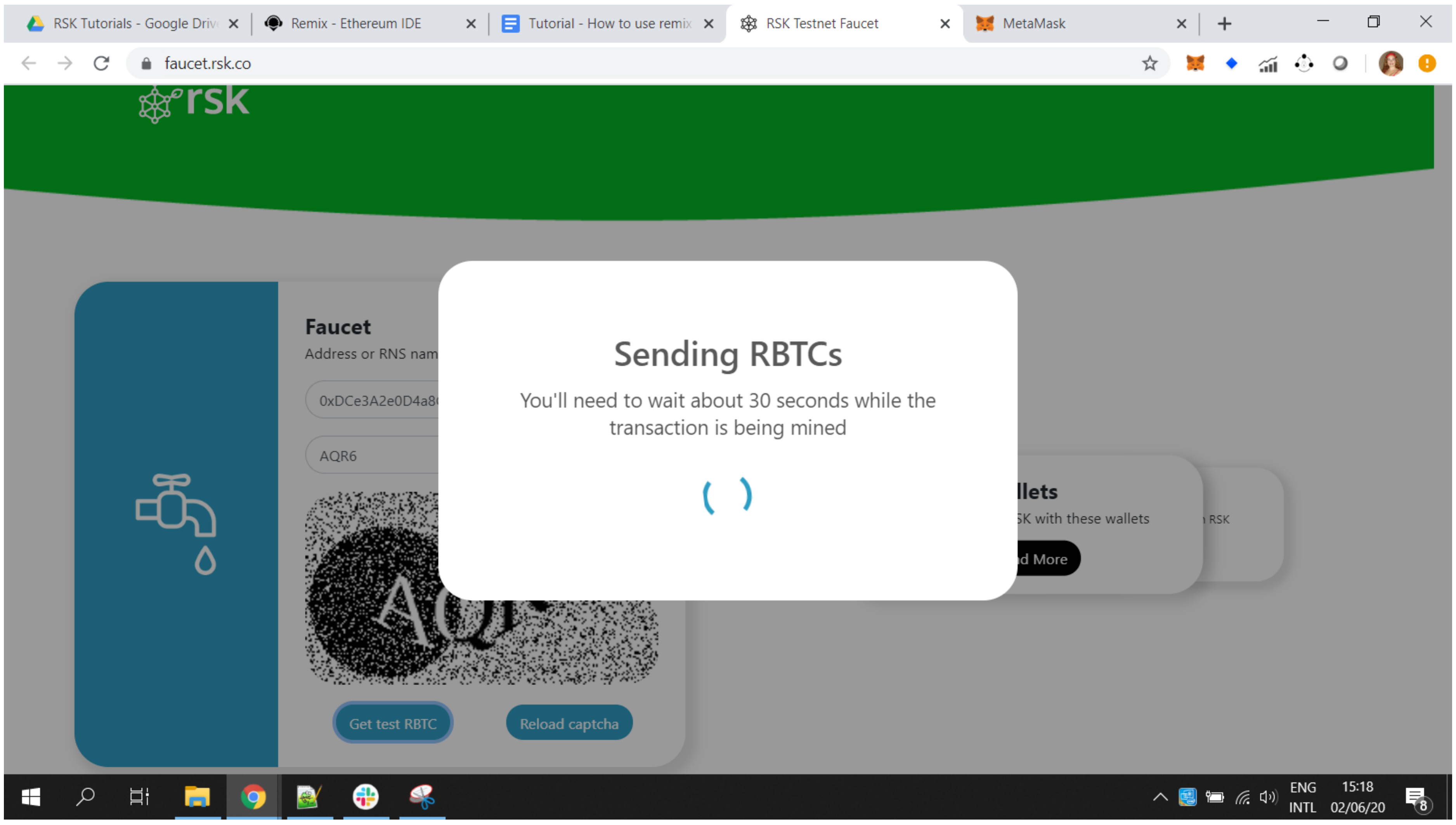
Faucet

Address or RNS name

0x9682725a85f85f097ab368555a286618dc982c99

M7vvPx

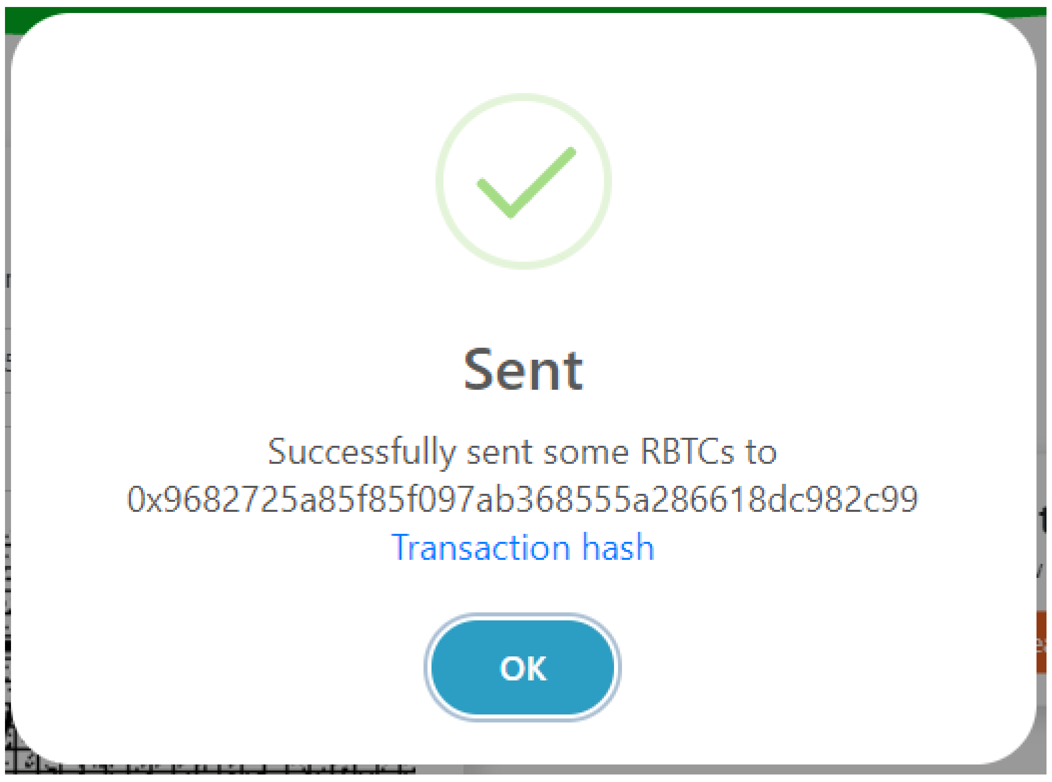
Enter your wallet address that you copied in the last step, and complete the CAPTCHA.



Sending RBTCs

You'll need to wait about 30 seconds while the transaction is being mined

Wait a few seconds...



Sent

Successfully sent some RBTCs to 0x9682725a85f85f097ab368555a286618dc982c99

Transaction hash

You can see the transaction hash:

0x16bedc1339a8fe59e270b0c6d5175851010bb93d0cf6c4974f1705b9ead7ee6e

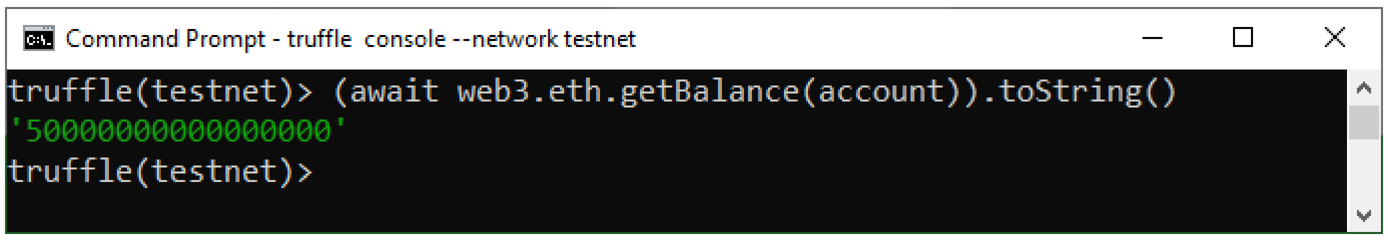
Now I have 0.05 tRBTC!

**Recheck balance**

To check balance again, run this command in the Truffle console:

```
(await web3.eth.getBalance(account)).toString()
```

Go to top

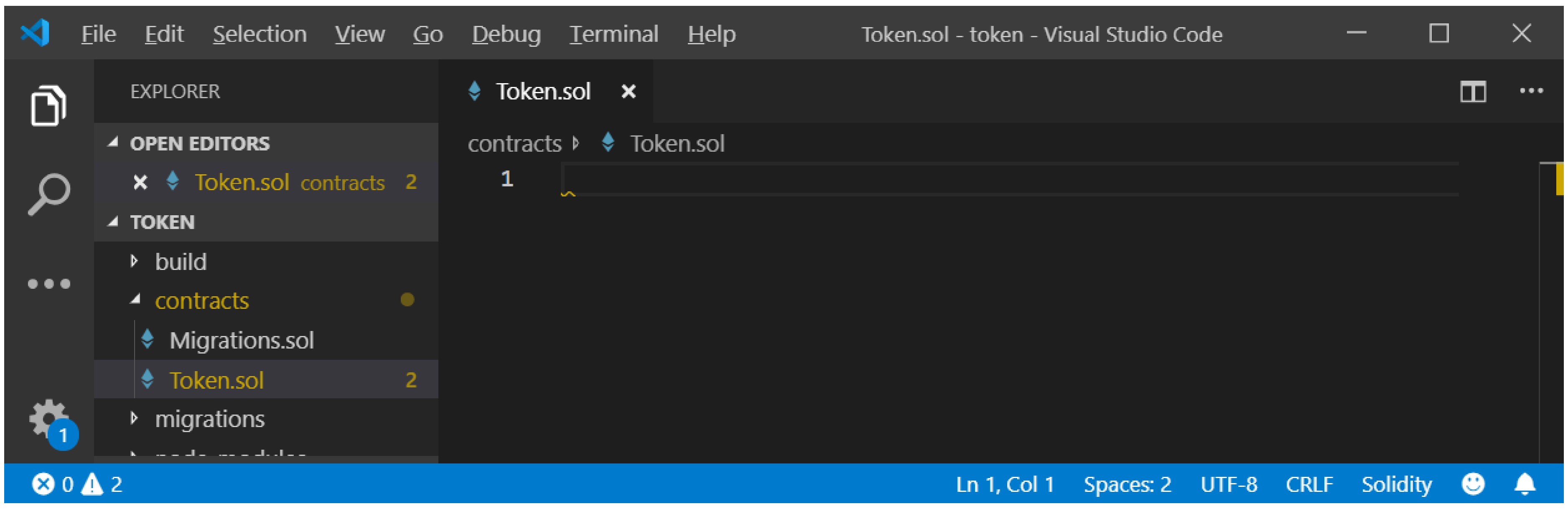


```
truffle(testnet)> (await web3.eth.getBalance(account)).toString()
'50000000000000000'
truffle(testnet)>
```

Now I have 50000000000000000, which means that I have 0.05 tRBTC with 18 decimal places of precision.

## Create the smart contract

In the `contracts` folder, create a new file named `Token.sol`.



**Token.sol with only 7 lines!**

This smart contract is a mintable ERC20 token. This means that, in addition to the standard ERC20 specification, it has a function for issuing new tokens.

> *To learn more about it, go to [EIP 20: ERC-20 Token Standard]*

Copy and paste the following code:

```
pragma solidity 0.5.2;
import '@openzeppelin/contracts/token/ERC20/ERC20Mintable.sol';
contract Token is ERC20Mintable {
       string public name = "My RSK token";
       string public symbol = "MRT";
       uint8 public decimals = 2;
}
```

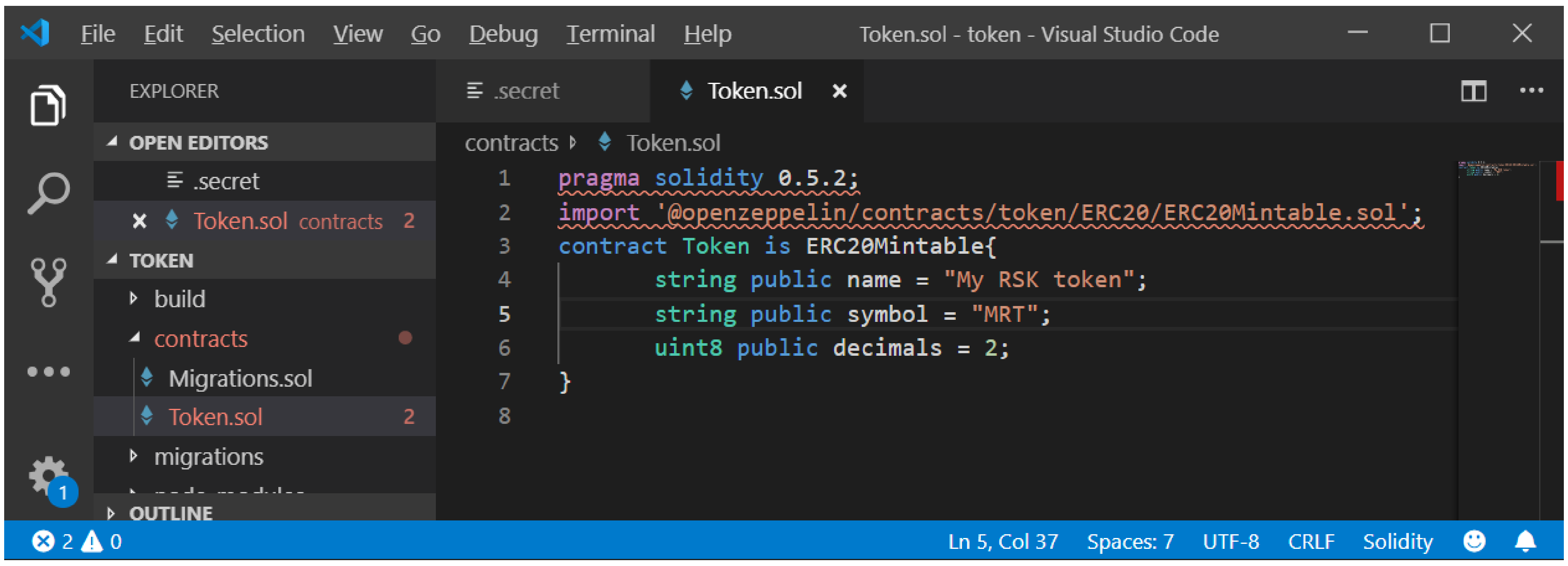
Let’s review the above code.

To create our ERC20 Token, we will import `ERC20Mintable` from Open Zeppelin. This library itself imports several other libraries such as `SafeMath.sol`, the standards for this kind of token, and the capability to mint tokens.

Inside the token, we define some basic information about the token: `name`, `symbol`, and number of `decimals` for the precision.

To inherit the library’s attributes and functions, we simply define our contract as a `ERC20Mintable` using the `is` keyword in this way.

Go to top

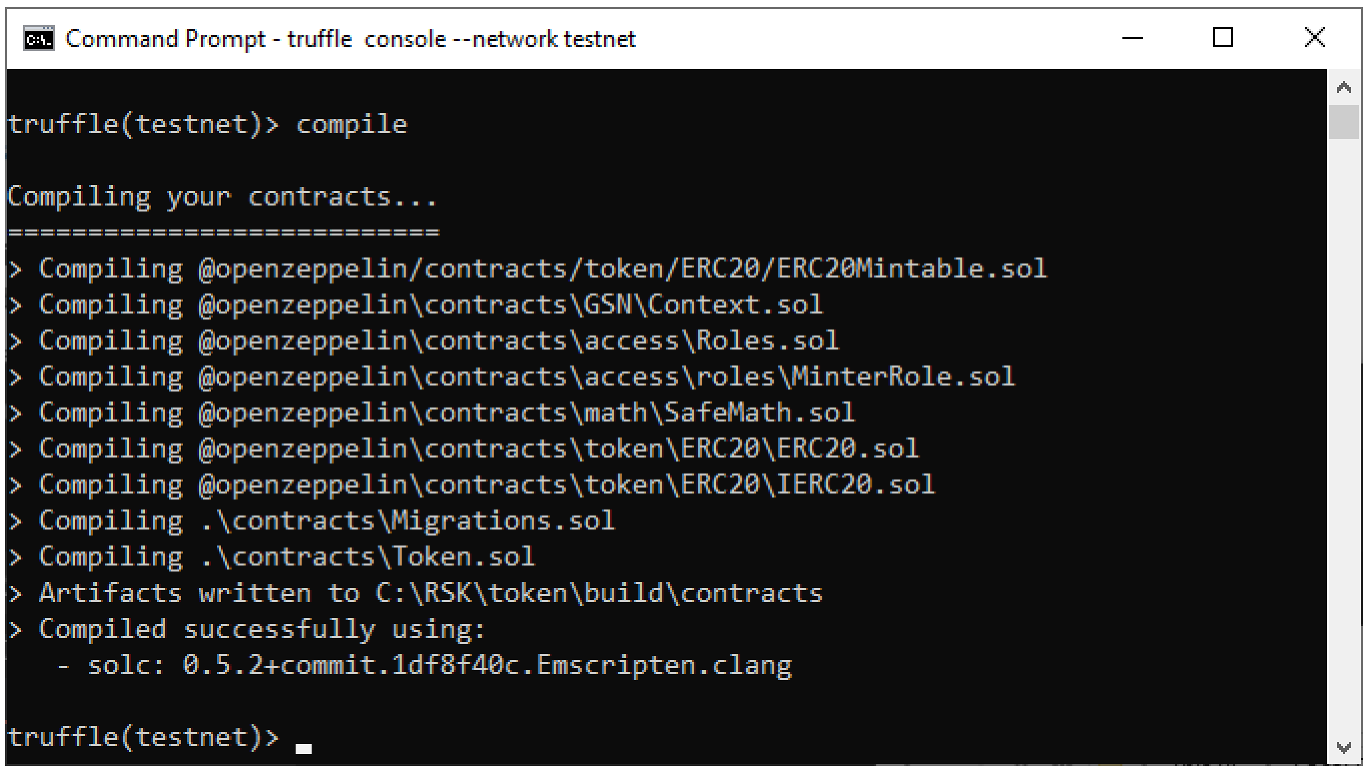


```
pragma solidity 0.5.2;
import '@openzeppelin/contracts/token/ERC20/ERC20Mintable.sol';
contract Token is ERC20Mintable{
        string public name = "My RSK token";
        string public symbol = "MRT";
        uint8 public decimals = 2;
}
```

## Compile a smart contract

In the Truffle console, run this command:

```
compile
```



Command Prompt - truffle console --network testnet

```
truffle(testnet)> compile

Compiling your contracts...
===========================
> Compiling @openzeppelin/contracts/token/ERC20/ERC20Mintable.sol
> Compiling @openzeppelin\contracts\GSN\Context.sol
> Compiling @openzeppelin\contracts\access\Roles.sol
> Compiling @openzeppelin\contracts\access\roles\MinterRole.sol
> Compiling @openzeppelin\contracts\math\SafeMath.sol
> Compiling @openzeppelin\contracts\token\ERC20\ERC20.sol
> Compiling @openzeppelin\contracts\token\ERC20\IERC20.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\Token.sol
> Artifacts written to C:\RSK\token\build\contracts
> Compiled successfully using:
   - solc: 0.5.2+commit.1df8f40c.Emscripten.clang

truffle(testnet)>
```

## Deploy a smart contract

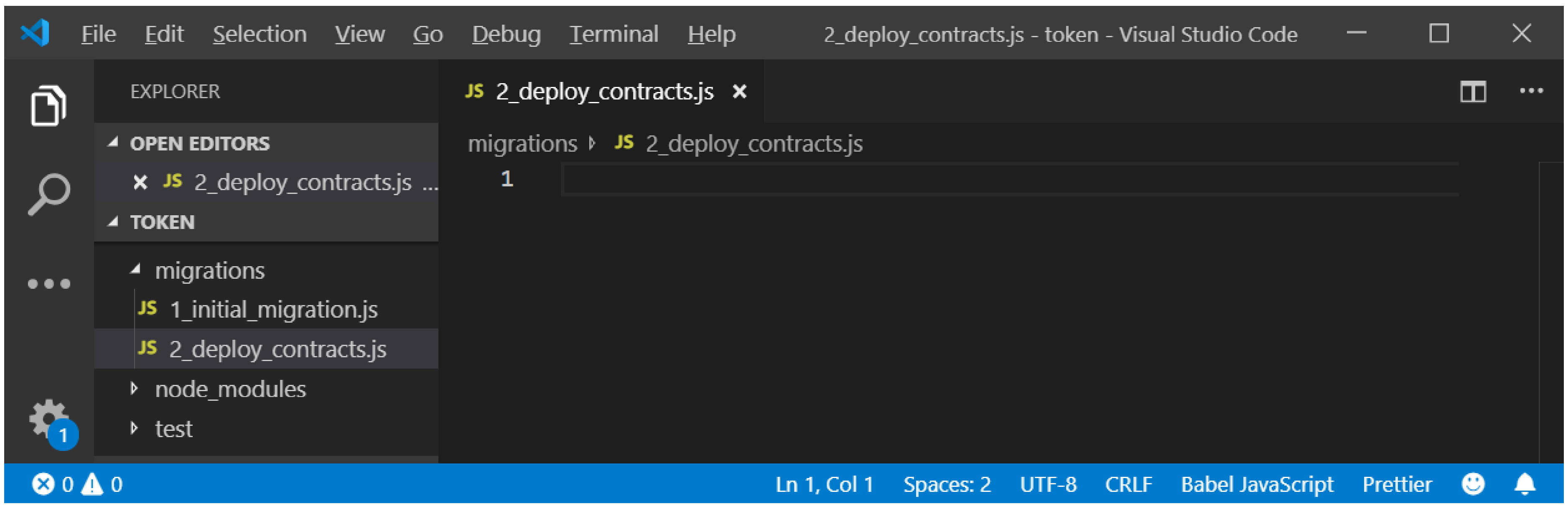
First of all, we need to create a a new migrations file where Truffle will find it, containing instructions to deploy the smart contract.

### Create token deployment file

The `migrations` folder has JavaScript files that help you deploy contracts to the network. These files are responsible for staging your deployment tasks, and they're written under the assumption that your deployment needs will change over time. A history of previously run migrations is recorded on-chain through a special Migrations contract. (source: **truffle: running-migrations**)

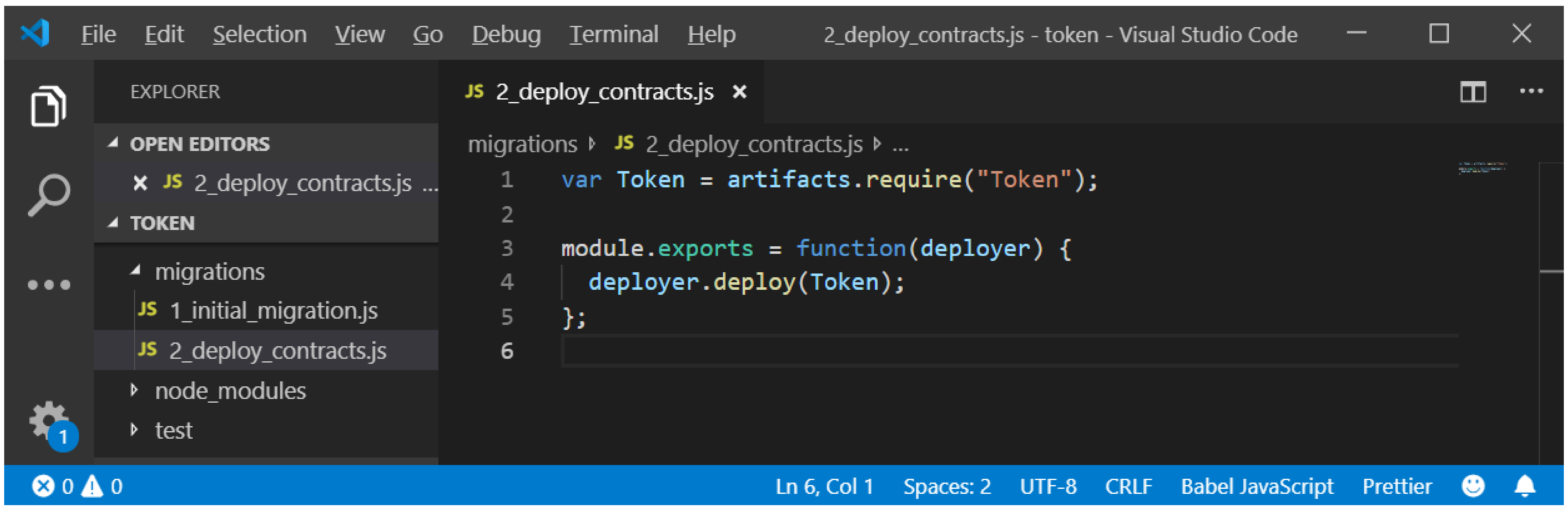
In the migrations folder, create the file `2_deploy_contracts.js`

Go to top



Copy and paste this code.

```
var Token = artifacts.require("Token");

module.exports = function(deployer) {
  deployer.deploy(Token);
};
```



```
var Token = artifacts.require("Token");

module.exports = function(deployer) {
  deployer.deploy(Token);
};
```

### Migrate

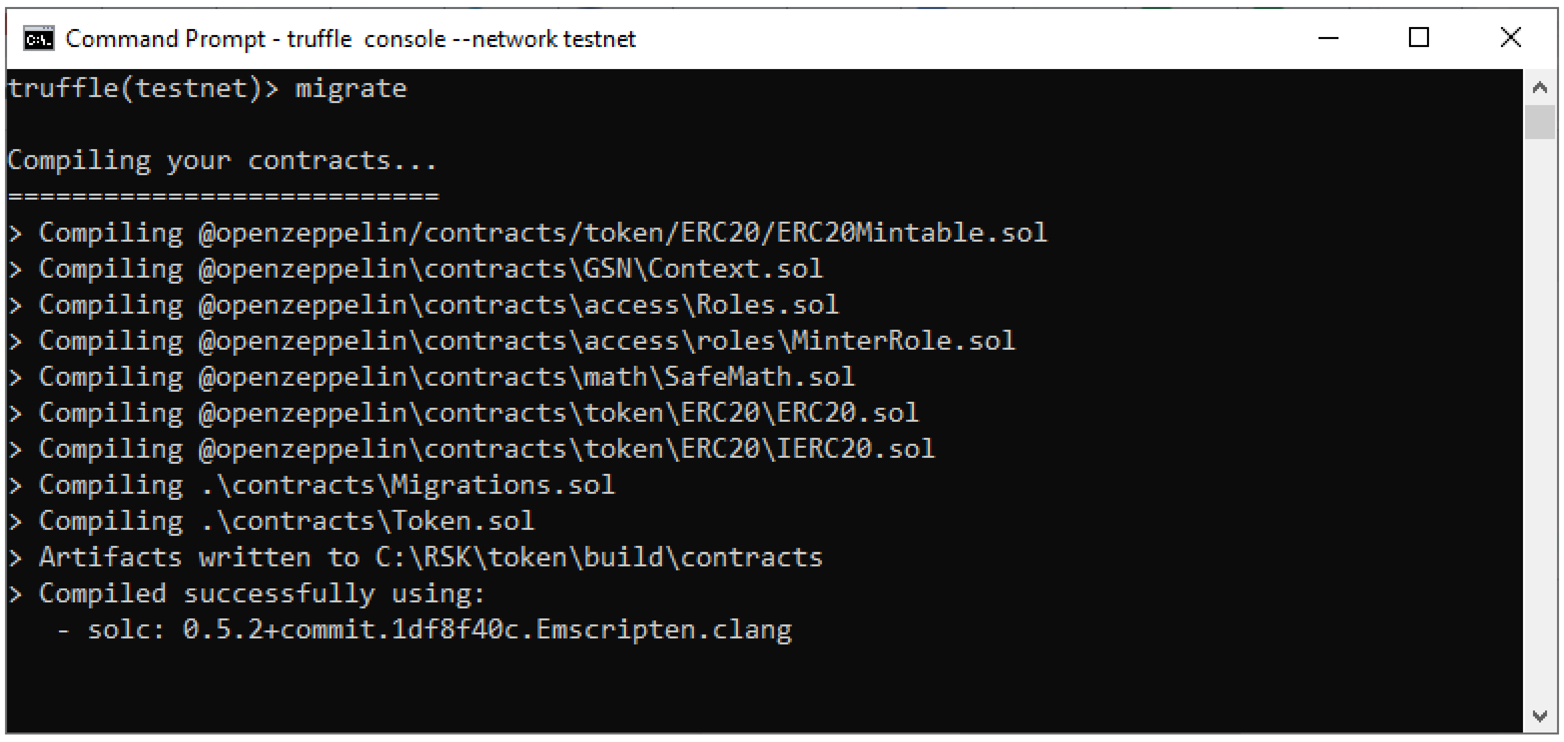
In the Truffle console, run this command:

```
migrate
```

Wait a few minutes while the transactions for the smart contract deployments are sent to the blockchain...

The migrate command will compile the smart contract again if necessary.

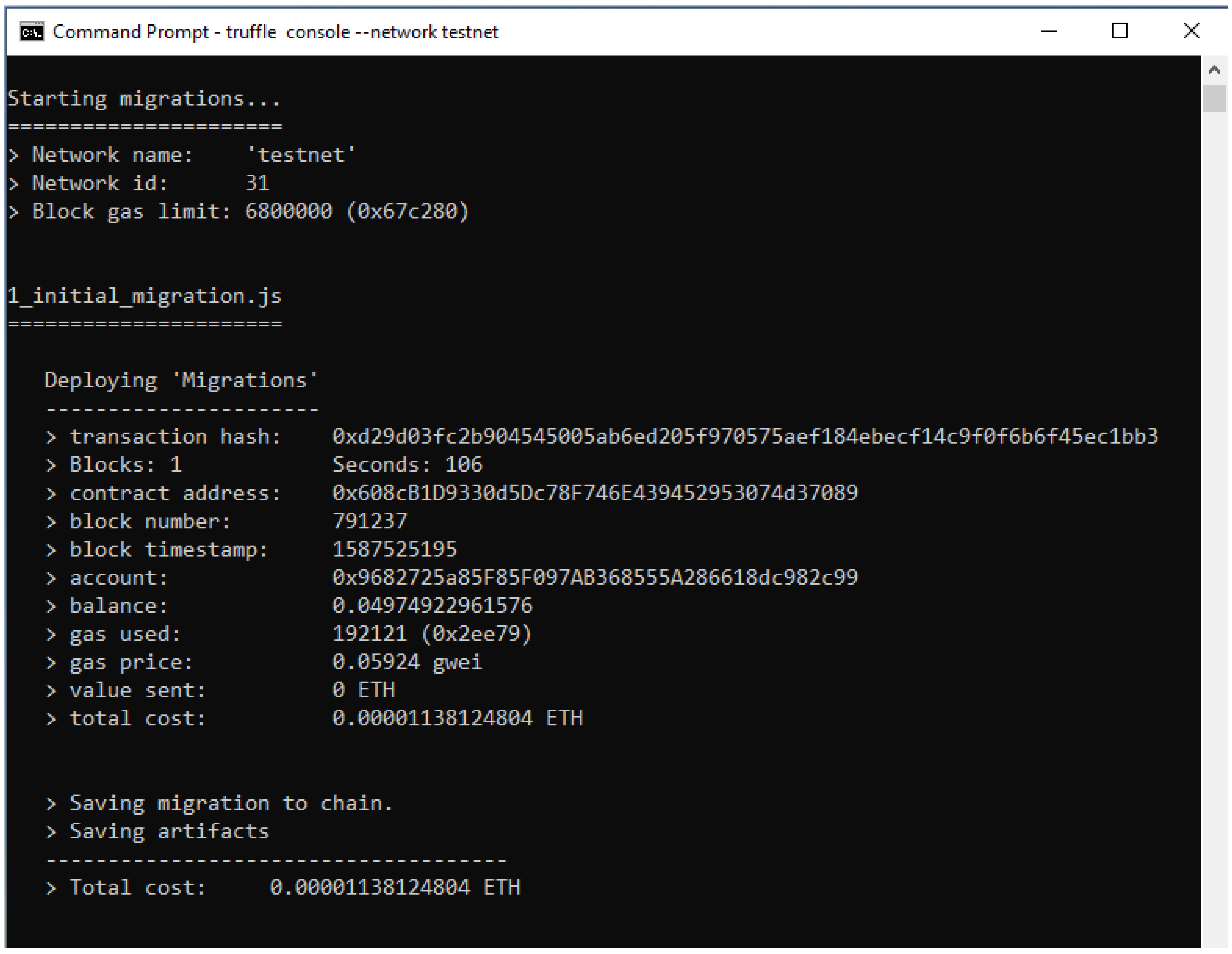
Go to top



Command Prompt - truffle console --network testnet

```
truffle(testnet)> migrate

Compiling your contracts...
===========================
> Compiling @openzeppelin/contracts/token/ERC20/ERC20Mintable.sol
> Compiling @openzeppelin\contracts\GSN\Context.sol
> Compiling @openzeppelin\contracts\access\Roles.sol
> Compiling @openzeppelin\contracts\access\roles\MinterRole.sol
> Compiling @openzeppelin\contracts\math\SafeMath.sol
> Compiling @openzeppelin\contracts\token\ERC20\ERC20.sol
> Compiling @openzeppelin\contracts\token\ERC20\IERC20.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\Token.sol
> Artifacts written to C:\RSK\token\build\contracts
> Compiled successfully using:
   - solc: 0.5.2+commit.1df8f40c.Emscripten.clang
```

First, it deploys the smart contract `Migrations.sol`, file generated by Truffle:



Command Prompt - truffle console --network testnet

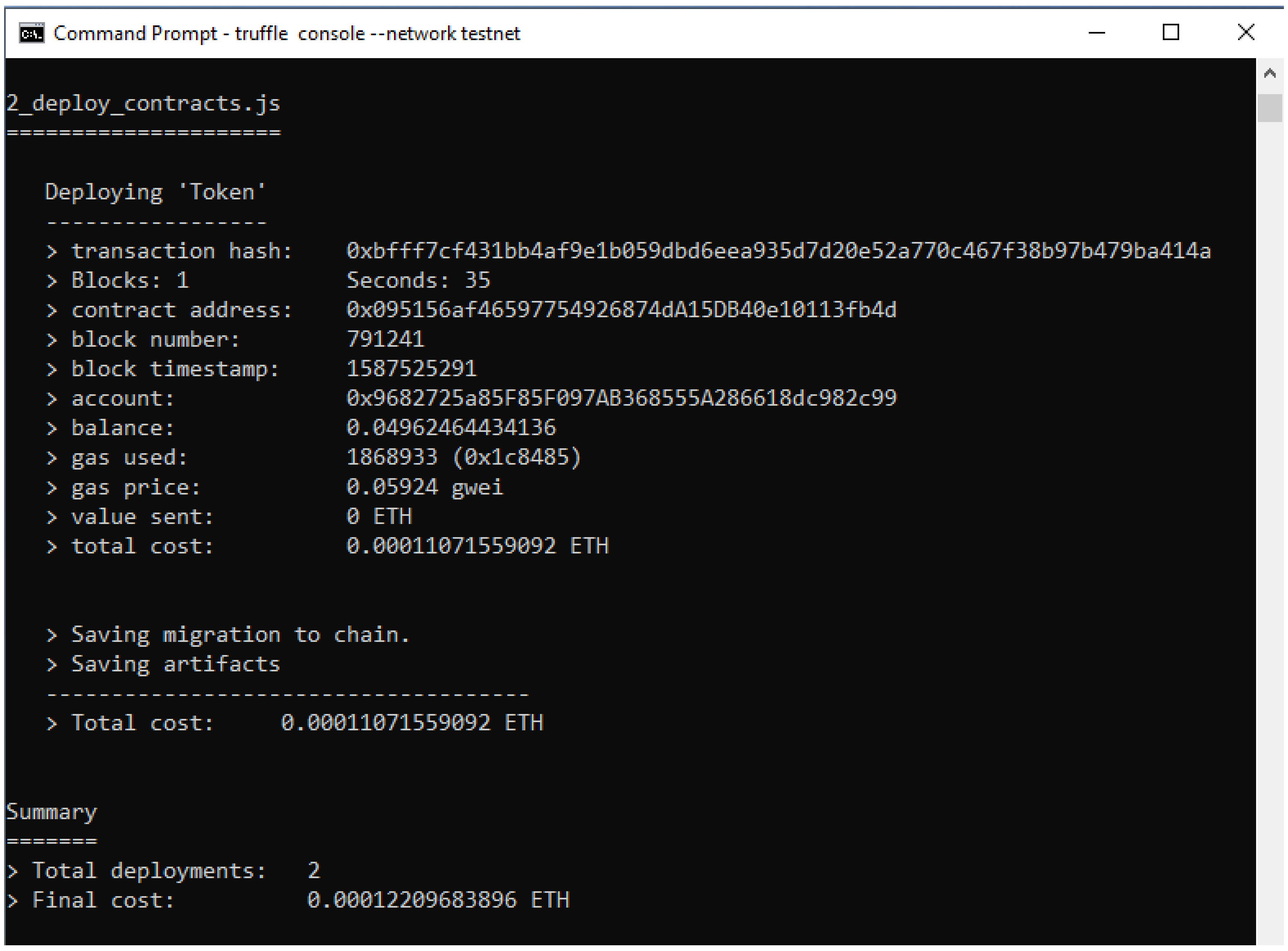
```
Starting migrations...
======================
> Network name:    'testnet'
> Network id:      31
> Block gas limit: 6800000 (0x67c280)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0xd29d03fc2b904545005ab6ed205f970575aef184ebecf14c9f0f6b6f45ec1bb3
   > Blocks: 1            Seconds: 106
   > contract address:    0x608cB1D9330d5Dc78F746E439452953074d37089
   > block number:        791237
   > block timestamp:     1587525195
   > account:             0x9682725a85F85F097AB368555A286618dc982c99
   > balance:             0.04974922961576
   > gas used:            192121 (0x2ee79)
   > gas price:           0.05924 gwei
   > value sent:          0 ETH
   > total cost:          0.00001138124804 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.00001138124804 ETH
```

This is the transaction:

[0xd29d03fc2b904545005ab6ed205f970575aef184ebecf14c9f0f6b6f45ec1bb3]

And then it deploys our smart contract `Token.sol`:

Go to top



```
2_deploy_contracts.js
=====================

   Deploying 'Token'
   -----------------
   > transaction hash:    0xbfff7cf431bb4af9e1b059dbd6eea935d7d20e52a770c467f38b97b479ba414a
   > Blocks: 1            Seconds: 35
   > contract address:    0x095156af46597754926874dA15DB40e10113fb4d
   > block number:        791241
   > block timestamp:     1587525291
   > account:             0x9682725a85F85F097AB368555A286618dc982c99
   > balance:             0.04962464434136
   > gas used:            1868933 (0x1c8485)
   > gas price:           0.05924 gwei
   > value sent:          0 ETH
   > total cost:          0.00011071559092 ETH


   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:     0.00011071559092 ETH


Summary
=======
> Total deployments:   2
> Final cost:          0.00012209683896 ETH
```

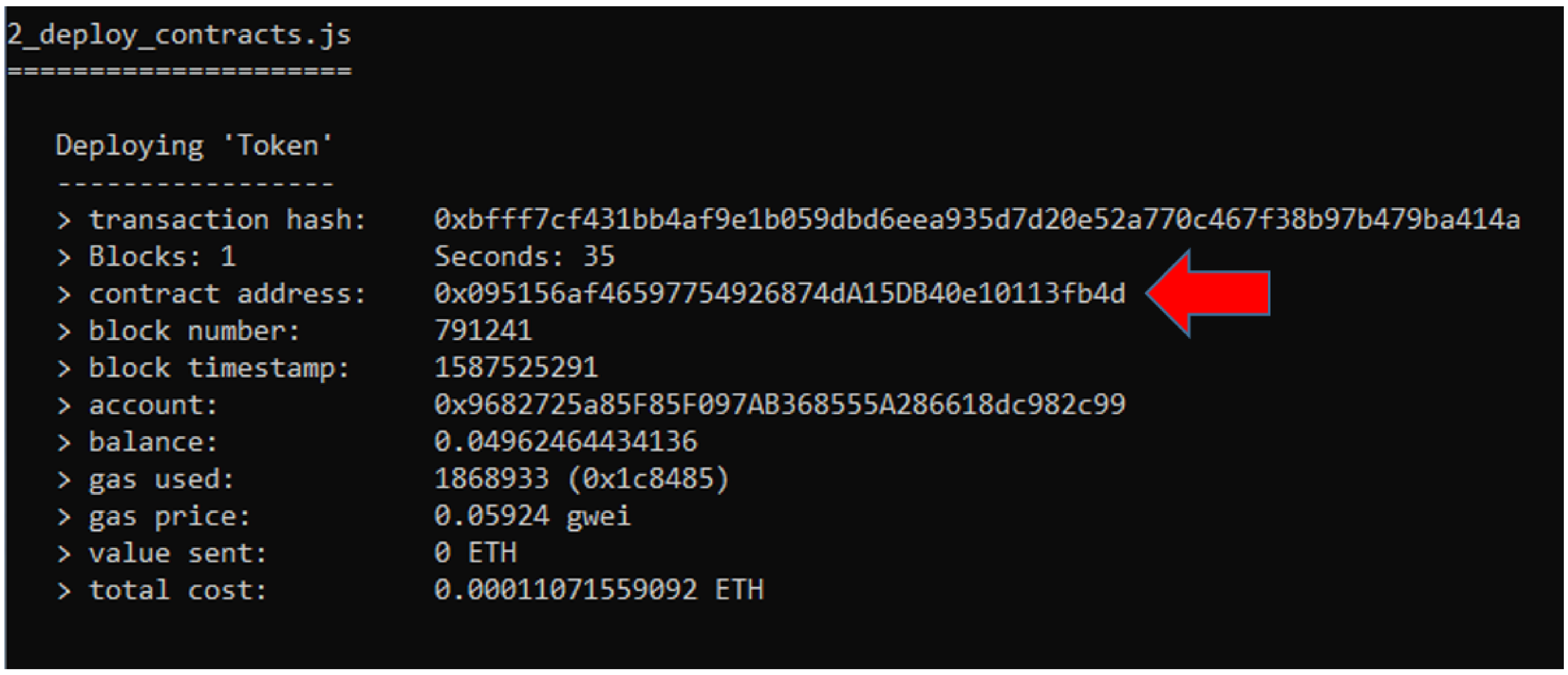
This is the transaction:

0xbfff7cf431bb4af9e1b059dbd6eea935d7d20e52a770c467f38b97b479ba414a

Congratulations!

`My RSK Token` is now published on the RSK Testnet.

Save the contract address of token, it will be used shortly:



```
2_deploy_contracts.js
=====================

   Deploying 'Token'
   -----------------
   > transaction hash:    0xbfff7cf431bb4af9e1b059dbd6eea935d7d20e52a770c467f38b97b479ba414a
   > Blocks: 1            Seconds: 35
   > contract address:    0x095156af46597754926874dA15DB40e10113fb4d
   > block number:        791241
   > block timestamp:     1587525291
   > account:             0x9682725a85F85F097AB368555A286618dc982c99
   > balance:             0.04962464434136
   > gas used:            1868933 (0x1c8485)
   > gas price:           0.05924 gwei
   > value sent:          0 ETH
   > total cost:          0.00011071559092 ETH
```

In the tutorial example:

```
tokenAddress = "0x095156af46597754926874dA15DB40e10113fb4d"
```

Go to top

## Interact with the token using Truffle console

We need to interact with the newly created token in Truffle console.

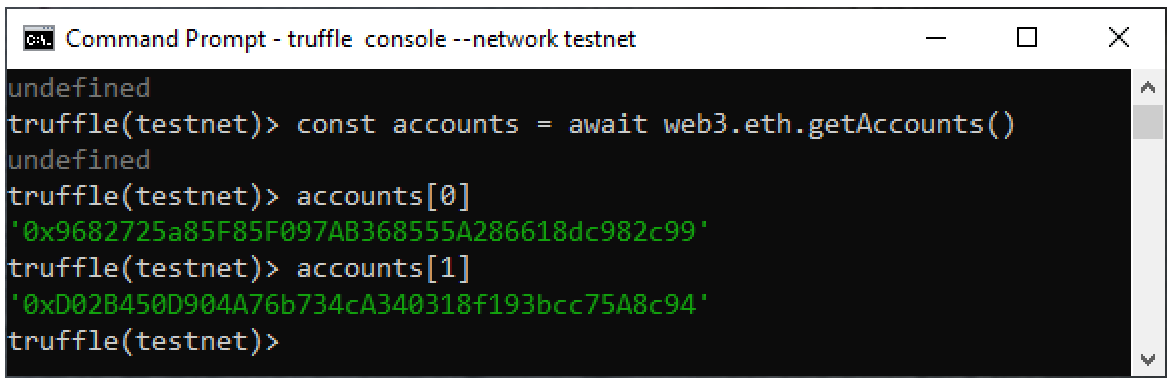
### Get your accounts

In the Truffle console, enter:

```
const accounts = await web3.eth.getAccounts()
```

To view each account:

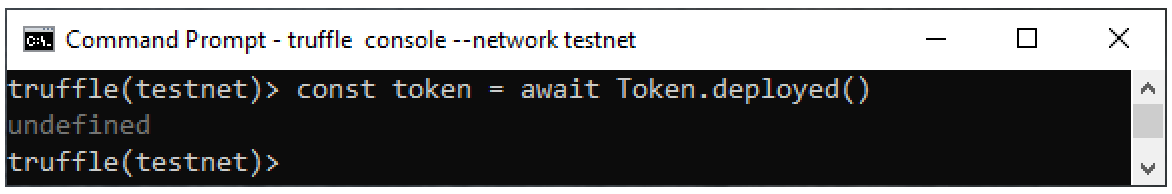
```
accounts[0]
accounts[1]
```



```
undefined
truffle(testnet)> const accounts = await web3.eth.getAccounts()
undefined
truffle(testnet)> accounts[0]
'0x9682725a85F85F097AB368555A286618dc982c99'
truffle(testnet)> accounts[1]
'0xD02B450D904A76b734cA340318f193bcc75A8c94'
truffle(testnet)>
```

### Connect with your token

```
const token = await Token.deployed()
```



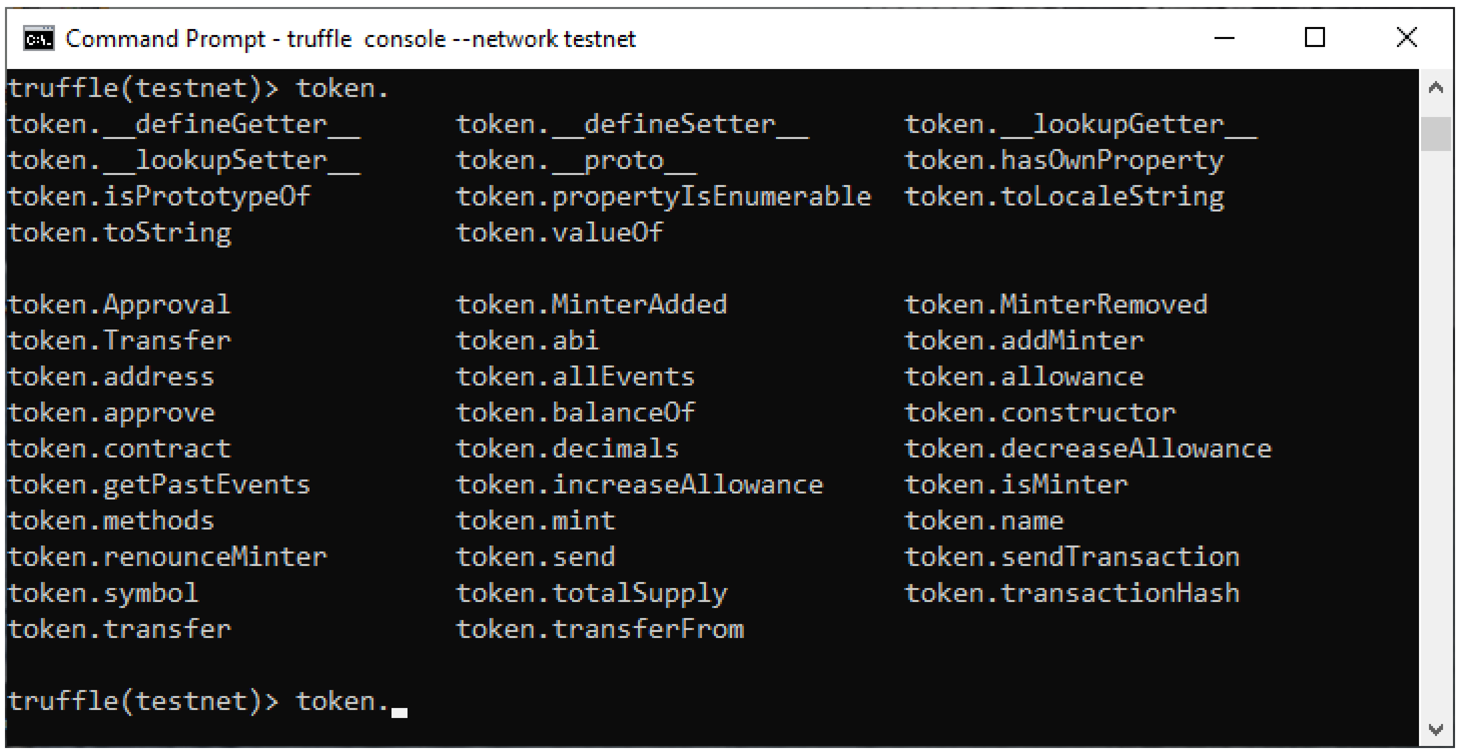
```
truffle(testnet)> const token = await Token.deployed()
undefined
truffle(testnet)>
```

Confirm if our instance is OK.

Enter the instance's name: `token`, then `.`, without space hit the TAB button twice to trigger auto-complete as seen below. This will display the published address of the smart contract, and the transaction hash for its deployment, among other things, including all public variables and methods available.

```
token. [TAB] [TAB]
```

Go to top

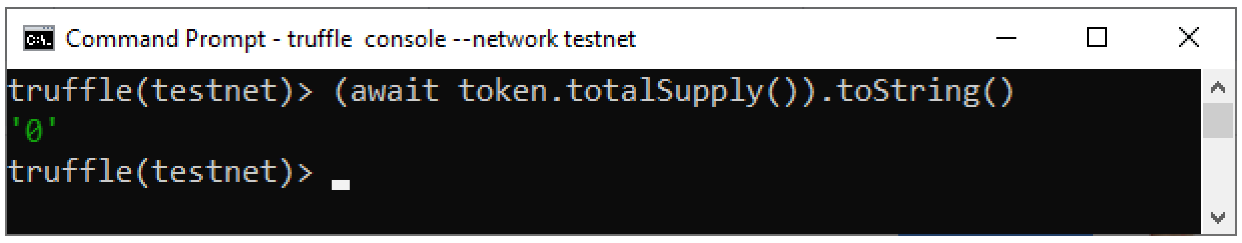


```
Command Prompt - truffle console --network testnet
truffle(testnet)> token.
token.__defineGetter__      token.__defineSetter__      token.__lookupGetter__
token.__lookupSetter__      token.__proto__             token.hasOwnProperty
token.isPrototypeOf         token.propertyIsEnumerable  token.toLocaleString
token.toString              token.valueOf

token.Approval              token.MinterAdded           token.MinterRemoved
token.Transfer              token.abi                   token.addMinter
token.address               token.allEvents             token.allowance
token.approve               token.balanceOf             token.constructor
token.contract              token.decimals              token.decreaseAllowance
token.getPastEvents         token.increaseAllowance     token.isMinter
token.methods               token.mint                  token.name
token.renounceMinter        token.send                  token.sendTransaction
token.symbol                token.totalSupply           token.transactionHash
token.transfer              token.transferFrom

truffle(testnet)> token._
```

### Check the total supply

To check if we have tokens already minted, call the `totalSupply` function:

```
(await token.totalSupply()).toString()
```



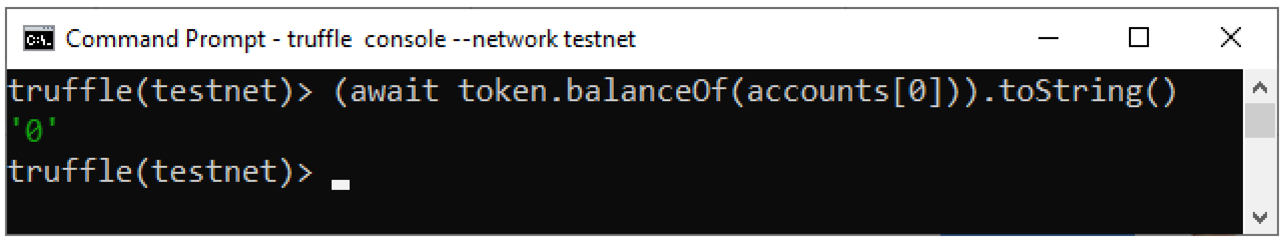
```
Command Prompt - truffle console --network testnet
truffle(testnet)> (await token.totalSupply()).toString()
'0'
truffle(testnet)> _
```

The returned value is 0, which is expected, since we did not perform any initial mint when we deployed the token.

### Check the token balance

To check the balance of an account, call the `balanceOf` function. For example, to check the balance of account 0:

```
(await token.balanceOf(accounts[0])).toString()
```



```
Command Prompt - truffle console --network testnet
truffle(testnet)> (await token.balanceOf(accounts[0])).toString()
'0'
truffle(testnet)> _
```

The returned value is also 0, which is expected, since we did not make any initial mint when we deployed the token, and by definition no accounts can have any tokens yet.

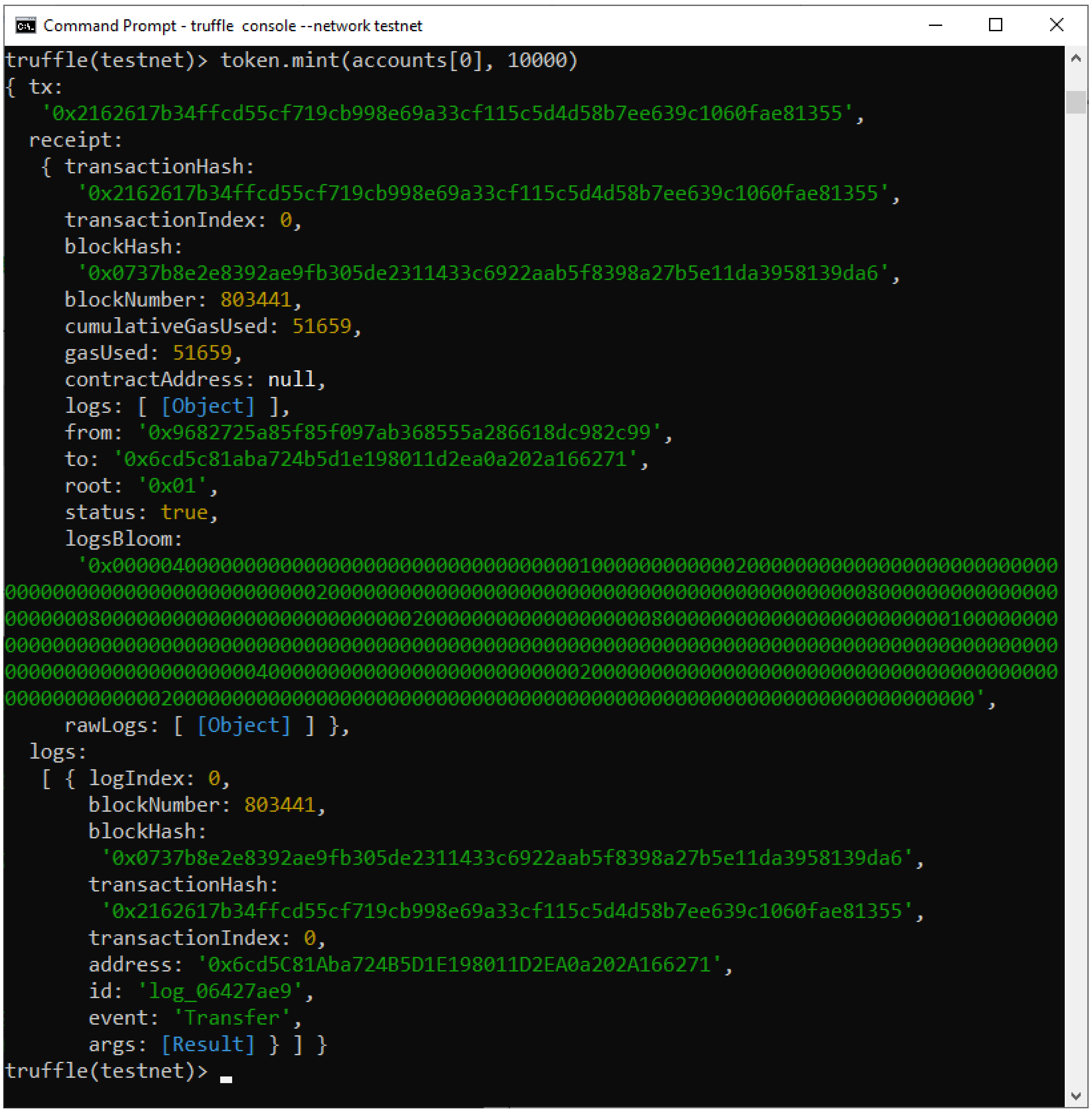
### Mint tokens

Run this command:

```
token.mint(accounts[0], 10000)
```

This command sent a transaction to mint 100 tokens for account 0.

Go to top



```
truffle(testnet)> token.mint(accounts[0], 10000)
{ tx:
   '0x2162617b34ffcd55cf719cb998e69a33cf115c5d4d58b7ee639c1060fae81355',
  receipt:
   { transactionHash:
      '0x2162617b34ffcd55cf719cb998e69a33cf115c5d4d58b7ee639c1060fae81355',
     transactionIndex: 0,
     blockHash:
      '0x0737b8e2e8392ae9fb305de2311433c6922aab5f8398a27b5e11da3958139da6',
     blockNumber: 803441,
     cumulativeGasUsed: 51659,
     gasUsed: 51659,
     contractAddress: null,
     logs: [ [Object] ],
     from: '0x9682725a85f85f097ab368555a286618dc982c99',
     to: '0x6cd5c81aba724b5d1e198011d2ea0a202a166271',
     root: '0x01',
     status: true,
     logsBloom:
      '0x0000400000000000000000000000000000000010000000000002000000000000000000000000000
0000000000000000000000000020000000000000000000000000000000000000000000008000000000000000
0000000800000000000000000000000000200000000000000000008000000000000000000000000100000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000004000000000000000000000000002000000000000000000000000000000000000000
0000000000000200000000000000000000000000000000000000000000000000000000000000000',
     rawLogs: [ [Object] ] },
  logs:
   [ { logIndex: 0,
       blockNumber: 803441,
       blockHash:
        '0x0737b8e2e8392ae9fb305de2311433c6922aab5f8398a27b5e11da3958139da6',
       transactionHash:
        '0x2162617b34ffcd55cf719cb998e69a33cf115c5d4d58b7ee639c1060fae81355',
       transactionIndex: 0,
       address: '0x6cd5C81Aba724B5D1E198011D2EA0a202A166271',
       id: 'log_06427ae9',
       event: 'Transfer',
       args: [Result] } ] }
truffle(testnet)>
```

To verify the transaction in the explorer, visit:

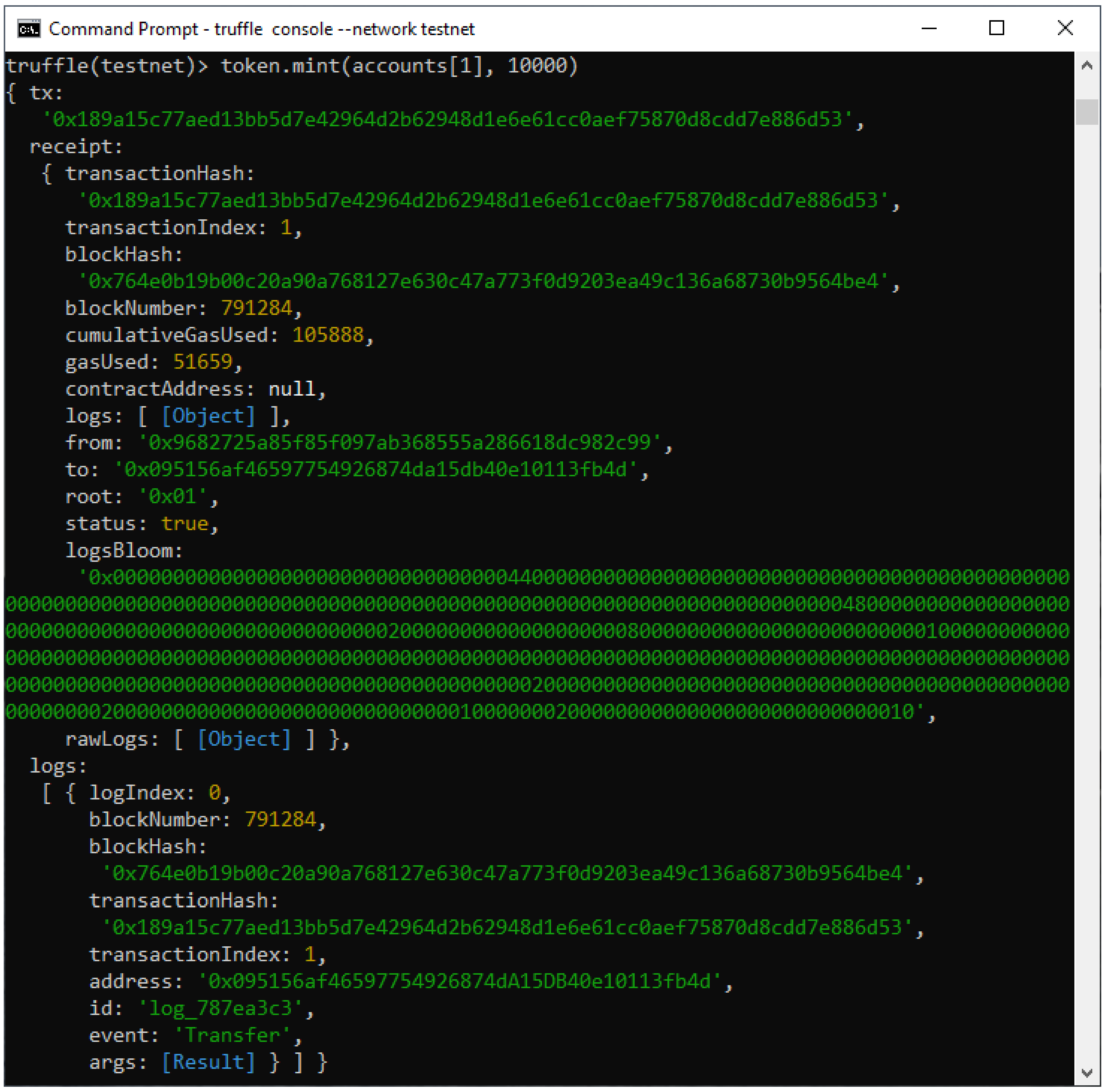
`0x2162617b34ffcd55cf719cb998e69a33cf115c5d4d58b7ee639c1060fae81355`

You can mint tokens for other accounts, for example, account 1:

```
token.mint(accounts[1], 10000)
```

For each account, the result will be like the following:

Go to top

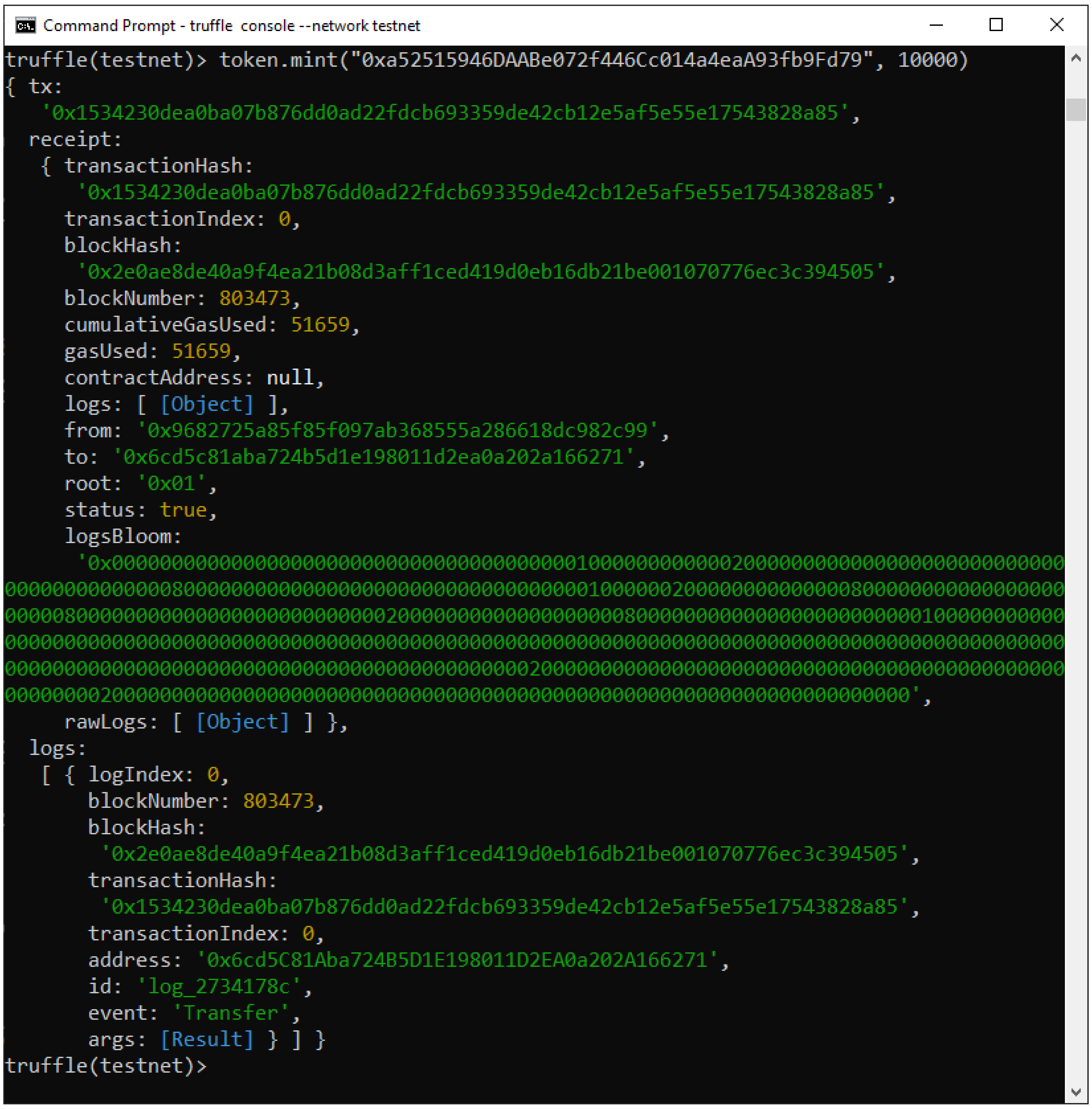


```
truffle(testnet)> token.mint(accounts[1], 10000)
{ tx:
   '0x189a15c77aed13bb5d7e42964d2b62948d1e6e61cc0aef75870d8cdd7e886d53',
  receipt:
   { transactionHash:
      '0x189a15c77aed13bb5d7e42964d2b62948d1e6e61cc0aef75870d8cdd7e886d53',
     transactionIndex: 1,
     blockHash:
      '0x764e0b19b00c20a90a768127e630c47a773f0d9203ea49c136a68730b9564be4',
     blockNumber: 791284,
     cumulativeGasUsed: 105888,
     gasUsed: 51659,
     contractAddress: null,
     logs: [ [Object] ],
     from: '0x9682725a85f85f097ab368555a286618dc982c99',
     to: '0x095156af46597754926874da15db40e10113fb4d',
     root: '0x01',
     status: true,
     logsBloom:
      '0x00000000000000000000000000000000044000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000048000000000000000000
00000000000000000000000000000000200000000000000000008000000000000000000000000100000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000200000000000000000000000000000000000000000000
00000000200000000000000000000000000000010000002000000000000000000000000000010',
     rawLogs: [ [Object] ] },
  logs:
   [ { logIndex: 0,
       blockNumber: 791284,
       blockHash:
        '0x764e0b19b00c20a90a768127e630c47a773f0d9203ea49c136a68730b9564be4',
       transactionHash:
        '0x189a15c77aed13bb5d7e42964d2b62948d1e6e61cc0aef75870d8cdd7e886d53',
       transactionIndex: 1,
       address: '0x095156af46597754926874dA15DB40e10113fb4d',
       id: 'log_787ea3c3',
       event: 'Transfer',
       args: [Result] } ] }
```

I can also mint to a specific address, `0xa52515946DAABe072f446Cc014a4eaA93fb9Fd79`:

```
token.mint("0xa52515946DAABe072f446Cc014a4eaA93fb9Fd79", 10000)
```

Go to top



```
truffle(testnet)> token.mint("0xa52515946DAABe072f446Cc014a4eaA93fb9Fd79", 10000)
{ tx:
   '0x1534230dea0ba07b876dd0ad22fdcb693359de42cb12e5af5e55e17543828a85',
  receipt:
   { transactionHash:
      '0x1534230dea0ba07b876dd0ad22fdcb693359de42cb12e5af5e55e17543828a85',
     transactionIndex: 0,
     blockHash:
      '0x2e0ae8de40a9f4ea21b08d3aff1ced419d0eb16db21be001070776ec3c394505',
     blockNumber: 803473,
     cumulativeGasUsed: 51659,
     gasUsed: 51659,
     contractAddress: null,
     logs: [ [Object] ],
     from: '0x9682725a85f85f097ab368555a286618dc982c99',
     to: '0x6cd5c81aba724b5d1e198011d2ea0a202a166271',
     root: '0x01',
     status: true,
     logsBloom:
      '0x000000000000000000000000000000000000000100000000000020000000000000000000000000000
00000000000000800000000000000000000000000000000001000000200000000000000800000000000000000
00000800000000000000000000000000200000000000000000008000000000000000000000001000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000200000000000000000000000000000000000000000000
00000000200000000000000000000000000000000000000000000000000000000000000000000',
     rawLogs: [ [Object] ] },
  logs:
   [ { logIndex: 0,
       blockNumber: 803473,
       blockHash:
        '0x2e0ae8de40a9f4ea21b08d3aff1ced419d0eb16db21be001070776ec3c394505',
       transactionHash:
        '0x1534230dea0ba07b876dd0ad22fdcb693359de42cb12e5af5e55e17543828a85',
       transactionIndex: 0,
       address: '0x6cd5C81Aba724B5D1E198011D2EA0a202A166271',
       id: 'log_2734178c',
       event: 'Transfer',
       args: [Result] } ] }
truffle(testnet)>
```

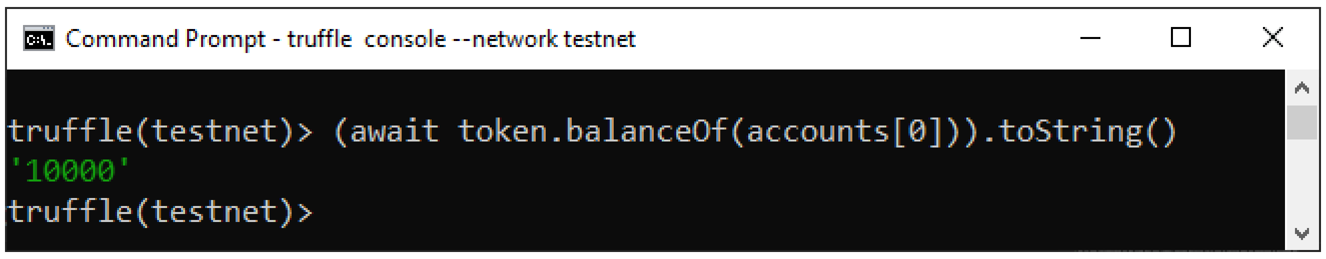
The transaction:

[0x1534230dea0ba07b876dd0ad22fdcb693359de42cb12e5af5e55e17543828a85]

**Reconfirm the token balance**

Check the balance of account 0 again:

```
(await token.balanceOf(accounts[0])).toString()
```



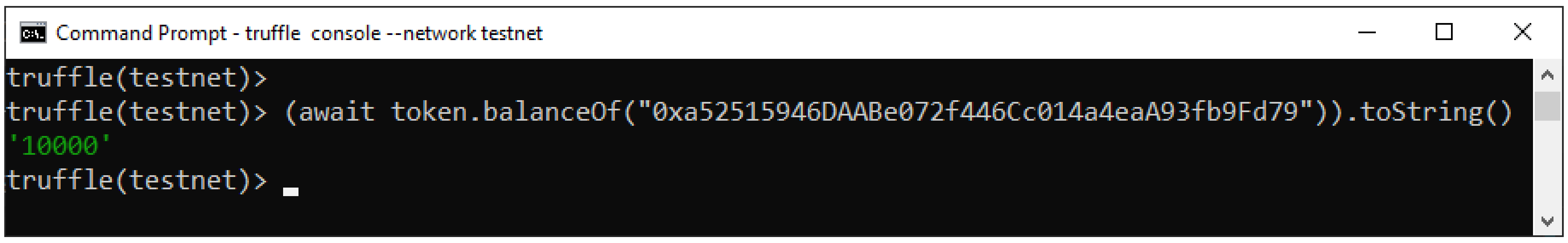
```
truffle(testnet)> (await token.balanceOf(accounts[0])).toString()
'10000'
truffle(testnet)>
```

The returned value is 10000, which is 100 with 2 decimal places of precision. This is exactly what we expected, as we issued 100 tokens

Also, you can get the balance of a specific address, for example, `0xa52515946DAABe072f446Cc014a4eaA93fb9Fd79`:

Go to top

```
(await token.balanceOf("0xa52515946DAABe072f446Cc014a4eaA93fb9Fd79")).toString()
```

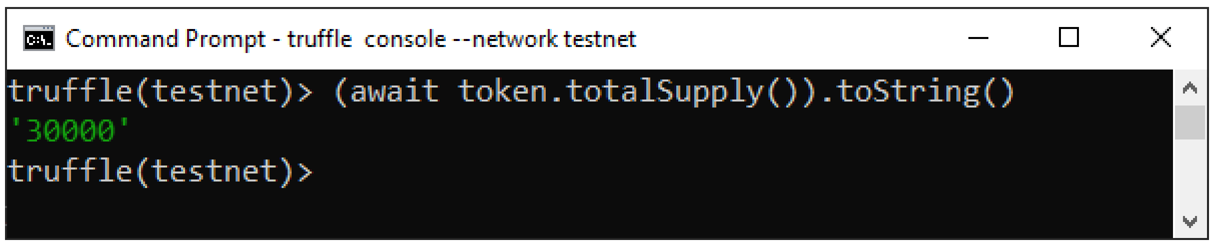


```
truffle(testnet)>
truffle(testnet)> (await token.balanceOf("0xa52515946DAABe072f446Cc014a4eaA93fb9Fd79")).toString()
'10000'
truffle(testnet)>
```

#### Check the total supply (again)

Check the total supply again:

```
(await token.totalSupply()).toString()
```



```
truffle(testnet)> (await token.totalSupply()).toString()
'30000'
truffle(testnet)>
```

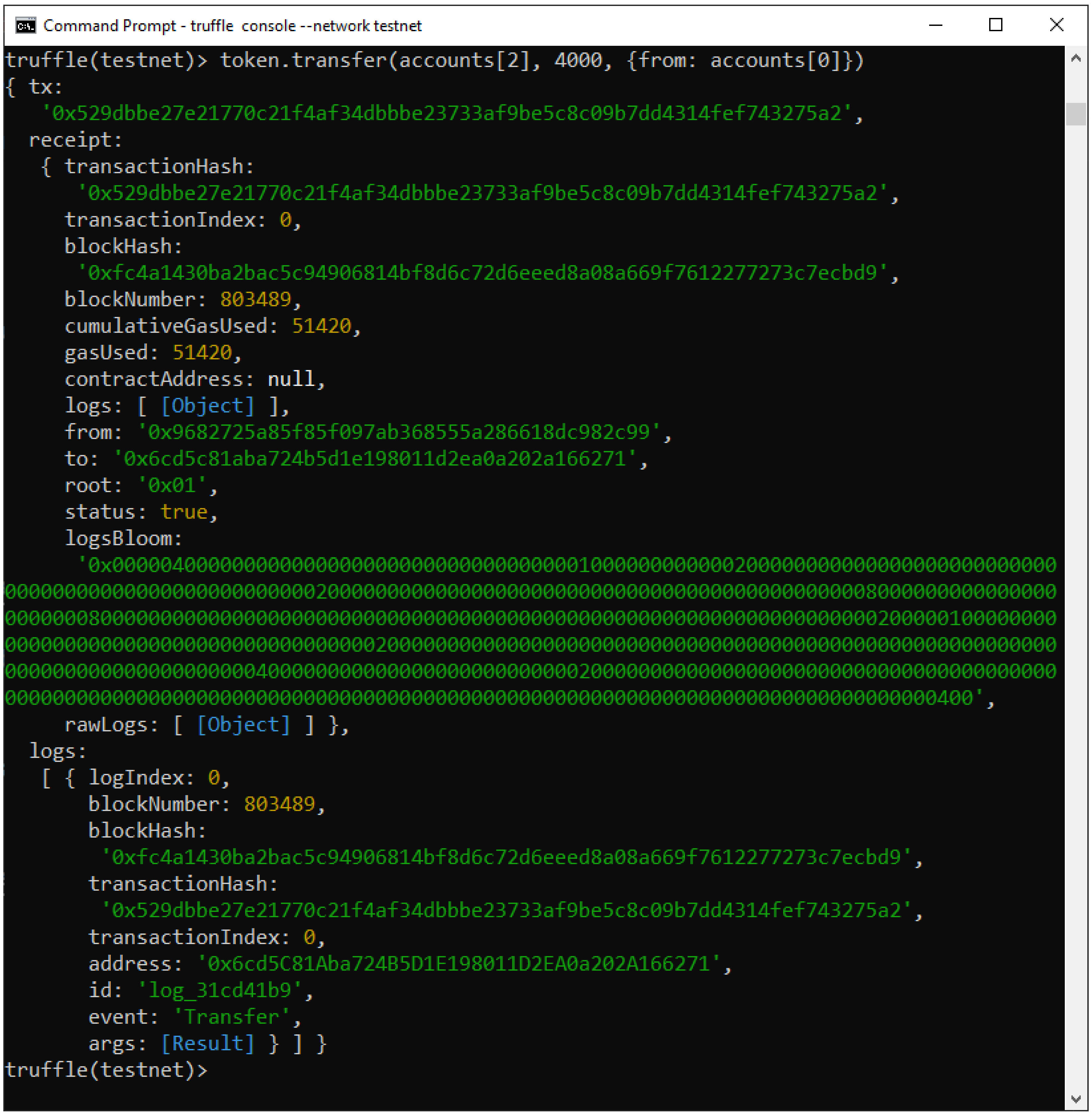
The returned value is 30000, which is 300 with 2 decimal places of precision. After minting 100 tokens for 3 accounts, this is perfect!

#### Transfer tokens

To transfer 40 tokens from account 0 to account 2. This can be done by calling the `transfer` function.

```
token.transfer(accounts[2], 4000, {from: accounts[0]})
```

Go to top



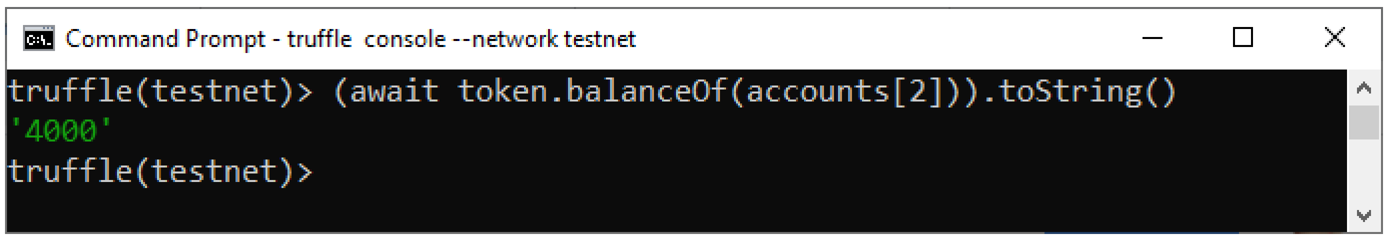
```
truffle(testnet)> token.transfer(accounts[2], 4000, {from: accounts[0]})
{ tx:
   '0x529dbbe27e21770c21f4af34dbbbe23733af9be5c8c09b7dd4314fef743275a2',
  receipt:
   { transactionHash:
      '0x529dbbe27e21770c21f4af34dbbbe23733af9be5c8c09b7dd4314fef743275a2',
     transactionIndex: 0,
     blockHash:
      '0xfc4a1430ba2bac5c94906814bf8d6c72d6eeed8a08a669f7612277273c7ecbd9',
     blockNumber: 803489,
     cumulativeGasUsed: 51420,
     gasUsed: 51420,
     contractAddress: null,
     logs: [ [Object] ],
     from: '0x9682725a85f85f097ab368555a286618dc982c99',
     to: '0x6cd5c81aba724b5d1e198011d2ea0a202a166271',
     root: '0x01',
     status: true,
     logsBloom:
      '0x00000400000000000000000000000000000000001000000000000200000000000000000000000000
0000000000000000000000000020000000000000000000000000000000000000000000008000000000000000
0000000800000000000000000000000000000000000000000000000000000000000000000200000100000000
0000000000000000000000000000000200000000000000000000000000000000000000000000000000000000
0000000000000000000004000000000000000000000000002000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000400',
     rawLogs: [ [Object] ] },
  logs:
   [ { logIndex: 0,
       blockNumber: 803489,
       blockHash:
        '0xfc4a1430ba2bac5c94906814bf8d6c72d6eeed8a08a669f7612277273c7ecbd9',
       transactionHash:
        '0x529dbbe27e21770c21f4af34dbbbe23733af9be5c8c09b7dd4314fef743275a2',
       transactionIndex: 0,
       address: '0x6cd5C81Aba724B5D1E198011D2EA0a202A166271',
       id: 'log_31cd41b9',
       event: 'Transfer',
       args: [Result] } ] }
truffle(testnet)>
```

Transaction:

0x529dbbe27e21770c21f4af34dbbbe23733af9be5c8c09b7dd4314fef743275a2

Account 2 had no tokens before the transfer, and now it should have 40. Let's check the balance of account 2:

```
(await token.balanceOf(accounts[2])).toString()
```



```
truffle(testnet)> (await token.balanceOf(accounts[2])).toString()
'4000'
truffle(testnet)>
```

Great! The balance of account 2 is correct.

## Final considerations

Did you think that it would be so easy to use the Truffle framework connected to the RSK network, and that it would be possible to create a token with less than 10 lines of code?

I showed you how to connect Truffle to the RSK network and deploy your own token with only 7 lines of code! Also I hope that you saw how simple it is to use the Open Zeppelin libraries, and that they work on the RSK network.

Go to top

Our goal is to join forces and give options to people who believe in smart contracts based on Ethereum, and also believe in the power of Bitcoin, through RSK.

I hope this tutorial has been helpful and I'd appreciate your feedback. Share it if you like it :)