# Exercises: Play with a Simple Blockchain with Simple Mining

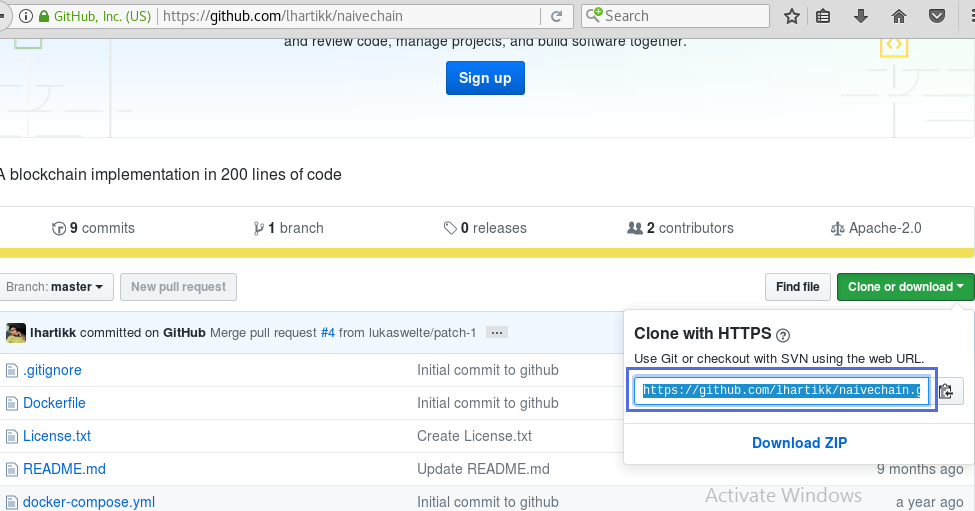
The goal of this exercise is to understand the principles of blockchain technology and where happens the moment of **mining**. The basic concept of blockchain is quite simple: a distributed database that maintains a continuously growing list of ordered records. You will run the simple JavaScript-based **model of blockchain** called “Naivechain”, **add nodes** and **mine blocks**.

These exercise is based on this project: <https://github.com/lhartikk/naivechain>. Thanks to the original authors.

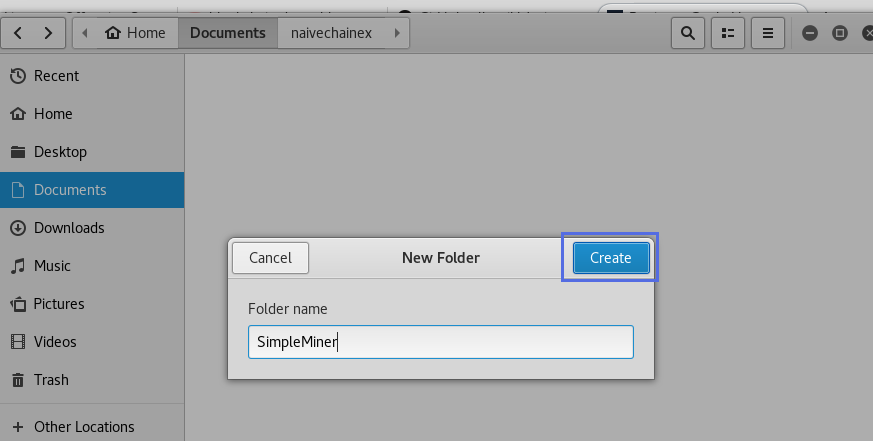
For this exercise you will need **Linux** or Linux-like command-line environment. We use “Debian” based distribution, but each other will work fine in most cases. Also, you will need "**node.js**", "**npm**" and "**cURL**" installed.

## Clone the “Naivechain” from GitHub

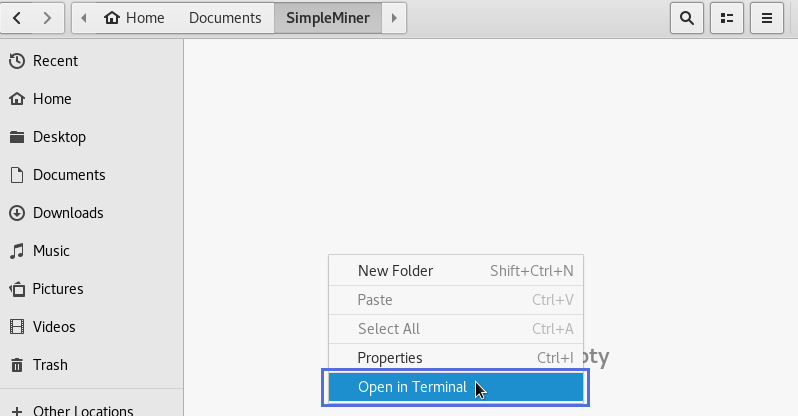
1. Go to the <https://github.com/lhartikk/naivechain>. Click the green button "**Clone or download**" and **copy the repository address**.



1. **Create directory** for the project. Name it “**SimpleMiner**”.

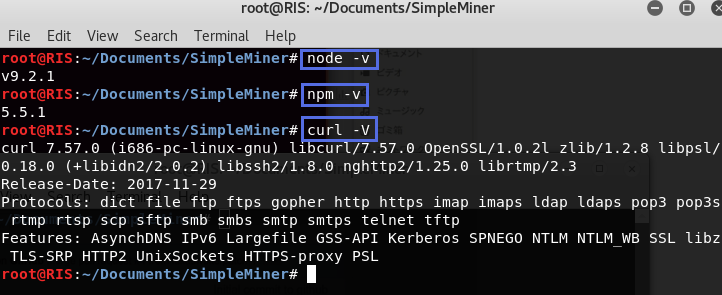


1. Open the directory and right click to **open** it in the **terminal**.



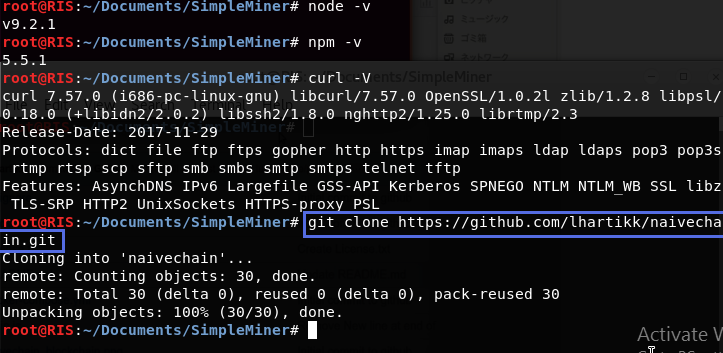
1. First **check the versions** of "**node.js**", "**npm**" and "**cURL**". In terminal type: "**node -v**", "**npm -v**" and "**curl -V**" (for cURL type capital "V" for version)**.** If you haven`t them installed in your computer please install them.

|  |
| --- |
| node -v  npm -v  curl -V |



1. **Clone the Git Hub** repository by type in cons ole: "**git clone https://github.com/lhartikk/naivechain.git**" and press enter.

|  |
| --- |
| git clone https://github.com/lhartikk/naivechain.git |



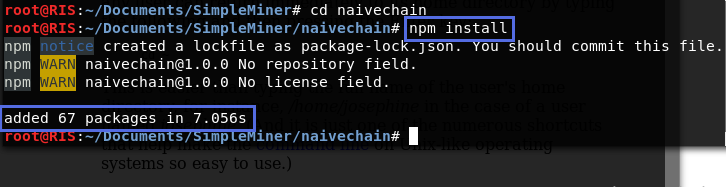
1. Then go to “**naivechain**” directory by typing "**cd naivechain**" in terminal.

|  |
| --- |
| cd naivechain |



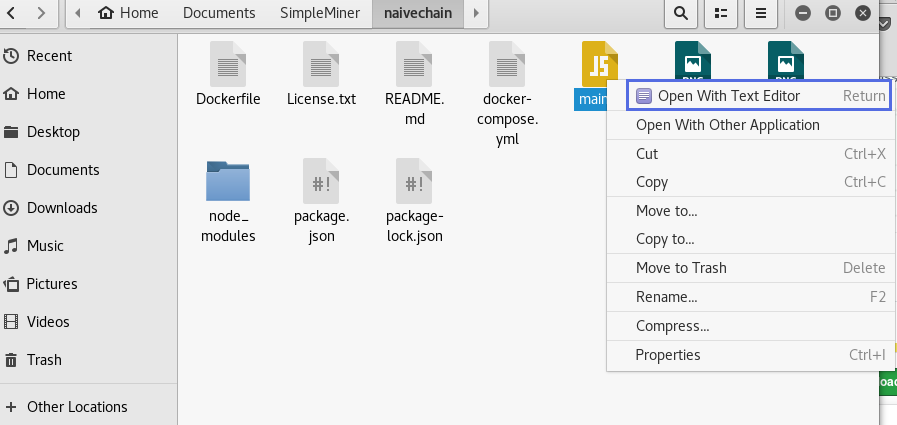
1. Now type "**npm install**" to get the **npm packages**. You must see something like this: "added 67 packages in 7.056s".

|  |
| --- |
| npm install |

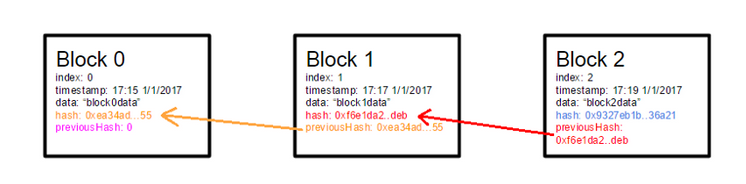


## Open the File "main.js" and Understand the Code Inside

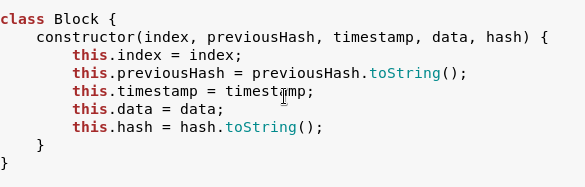
1. **Open** the file “**main.js**” in your preferred text editor or IDE. For example, you can open it with “**Text Editor**”.



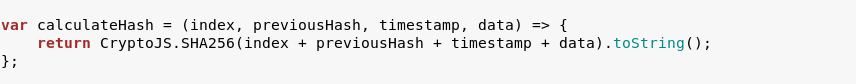
1. Let`s pay attention on the **block structure**. To keep things as simple as possible our example contains only the most necessary elements: **index, timestamp, data, hash and previous hash**. Following the blockchain conception the **hash** of the previous block must be **found** in the block to **preserve** the chain integrity.



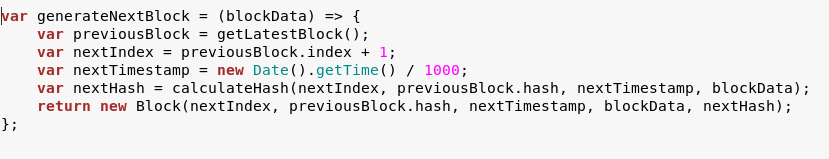
1. In the code we have **class Block** with constructor and **five fields**.



1. This is the sacral **moment of mining** when the miner calculates hash and found the valid one. The block needs to be **hashed** to keep the integrity of the data. A **SHA-256** is taken over the content of the block. It should be noted that in our example we want to learn where the mining happens and illustrate how the blockchain works. In our case, this hash has nothing to do with real process of “mining”, since there is no Proof Of Work problem to solve, but it illustrate the **process of block generating**. It real blockchains a Proof Of Work is a piece of data which is **difficult** (costly, time-consuming) to produce but **easy for others** to verify and which **satisfies** certain requirements. Producing a Proof Of Work can be a random process with low probability so that a lot of trial and error is required on average before a valid proof of work is generated. In order for a block to be accepted by network participants, miners must complete a Proof Of Work which covers all of the data in the block. Due to the very low probability of successful generation, this makes it **unpredictable** which worker computer in the network will be able to **generate** the next block. In our example this competition is not presented and **any** generated block is valid.



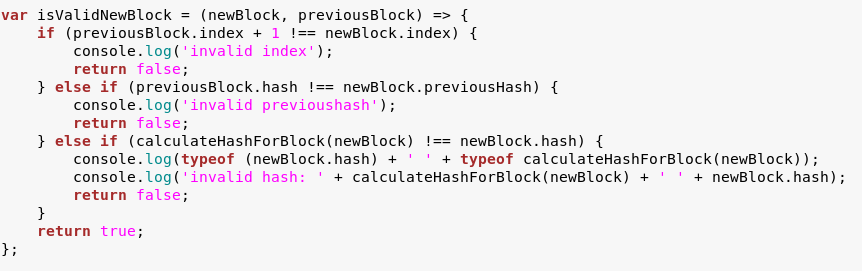
1. When we want to **generate** a block, first we must know the **hash of the previous block** because this is the link between chains of the blockchain. Next, we must **create** the rest of the **required content** (index, hash, data and timestamp). The **Block data** is something that is provided by the end-user how you will see further.



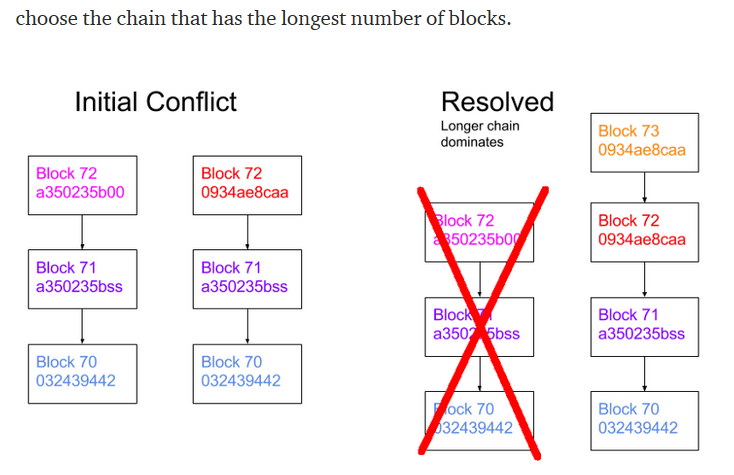
1. An in-memory JavaScript **array** is used to **store the blockchain**. The **first block** of the blockchain is always a so-called “**genesis-block**”, which is hard coded. We take it with function which returns a new Block with usual attributes. The Block index is 0, the “**previousHash**” don’t exist in reality and we give him a string value “0”. The current Block **hash** is hardcoded and **data field** contains string “**my genesis block**”.



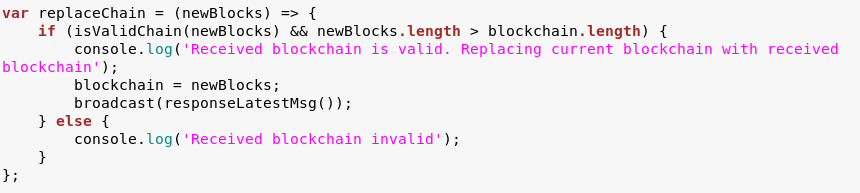
1. At any given time, we must be able to **validate** if a block or a chain of blocks are valid in terms of **integrity**. This is true especially when we receive **new blocks** from other **nodes** and must decide whether to **accept** them or not.



1. There should always be only **one** explicit **set of blocks** in the chain at a given time. In case of **conflicts** (e.g. two nodes both generate block number 72) we choose the chain that has the **longest number of blocks**.



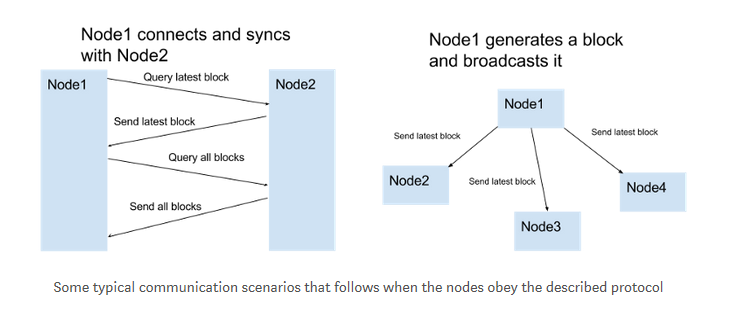
1. In our model of blockchain we compare the length of **new blocks sequence** with **length of existing blockchain.** The **longest chain** is accepted like **valid** blockchain.



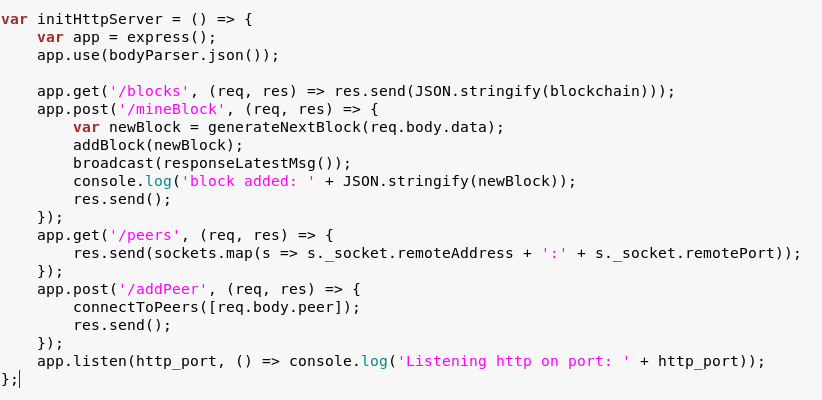
1. In purpose of modeling the blockchain and mining blocks the program has system of **communication** between nodes. An essential part of a node is to **share** and sync the blockchain with **other nodes**. The following **rules** are used to **keep the network in sync**.

* When a node **generates a new block**, it broadcasts it to the **network**
* When a node **connects to a new peer** it queries for the latest block
* When a node encounters a block that has an **index larger than the current known block**, it either adds the block to its current chain or queries for the full blockchain.

No automatic peer discovery is used. The location (URLs) of peers must be **manually** added.



1. The user must be able to **control the node** in some way. This is done by setting up a **HTTP server**.



As seen, the user is able to **interact** with the node in the following ways:

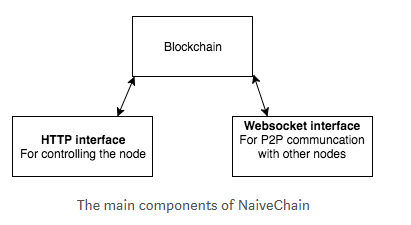
* **List all blocks**
* **Create a new block** with a content given by the user
* **List** or **add** **peers**

The most straightforward way to control the node is e.g. with Curl:

For example, to **get all blocks** from the node enter:

"curl PC\_ADDRESS/blocks" for PC with address localhost:3001 it will be "curl <http://localhost:3001/blocks>"

1. It should be noted that the node actually exposes **two web servers**: One for the **user to control the node** (**HTTP server**) and one for the **peer-to-peer communication** between the nodes. (Websocket HTTP server).



## Setup Connected Nodes and Mine a Block

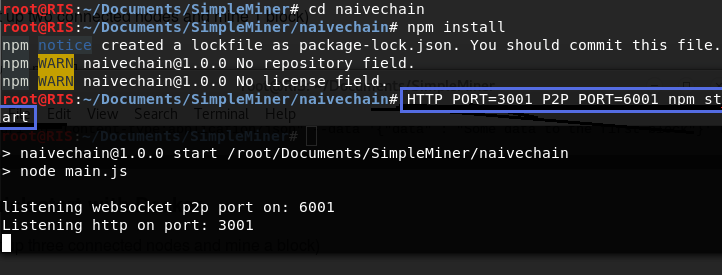
1. First let`s establish the **first node**. Get back to the Linux terminal and type "**HTTP\_PORT=3001 P2P\_PORT=6001 npm start**" then press "Enter". The node will listen for **signals from another nodes** by websocket interface on port 6001 and will listen for commands via HTTP interface on port 3001. We can see the node`s info on address: **lockalhost:3001**.

**Linux:**

|  |
| --- |
| HTTP\_PORT=3001 P2P\_PORT=6001 npm start |

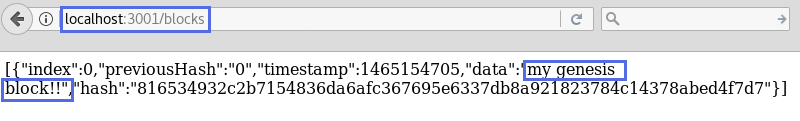
**Windows:**

|  |
| --- |
| set HTTP\_PORT=3001 && set P2P\_PORT=6001 && npm start |



1. Open your preferred web browser. Go to node`s address and attach command “**blocks**" in attempt to receive the **list of all blocks**. On browser write “**localhost:3001/blocks**". Here is the **first block** in blockchain that was **hardcoded**.

|  |
| --- |
| localhost:3001/blocks |



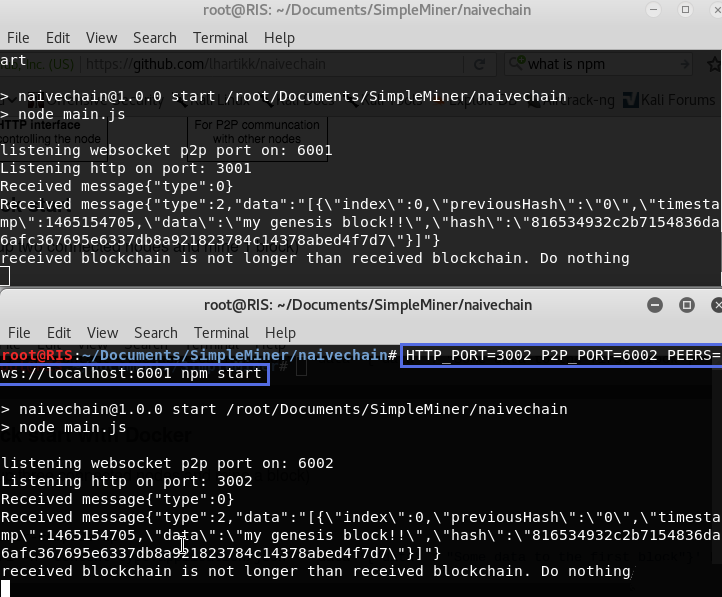
1. Now open **second terminal** window and set the **second peer** in chain. Type command: "**HTTP\_PORT=3002 P2P\_PORT=6002 PEERS=ws://localhost:6001 npm start**". The **second peer** will listen for **signals from another nodes** on **port 6002** and will listen for commands via HTTP interface on **port 3002**. This node will receive information by first peer via P2P communication **by port 6001.** We can see the node`s info on address: **lockalhost:3002**. Here on the picture are the both nodes terminals.

**Linux:**

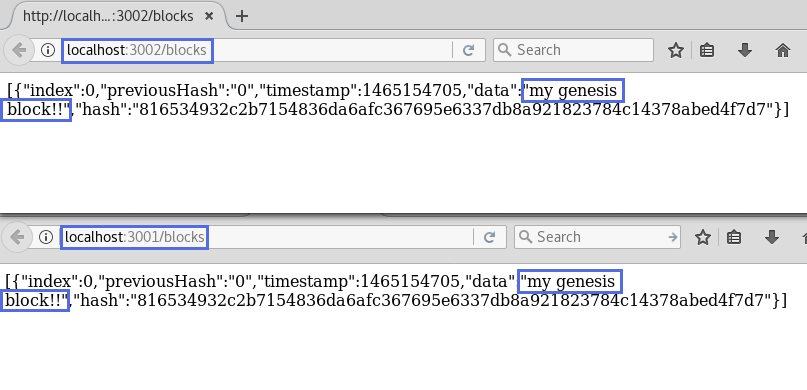
|  |
| --- |
| HTTP\_PORT=3002 P2P\_PORT=6002 PEERS=ws://localhost:6001 npm start |

**Windows:**

|  |
| --- |
| set HTTP\_PORT=3002 && set P2P\_PORT=6002 && set PEERS=http://localhost:6001 && npm start |



1. Let`s see **both nodes** in browser and get the **lists of all blocks.** As we see they have the **identic blockchain**.

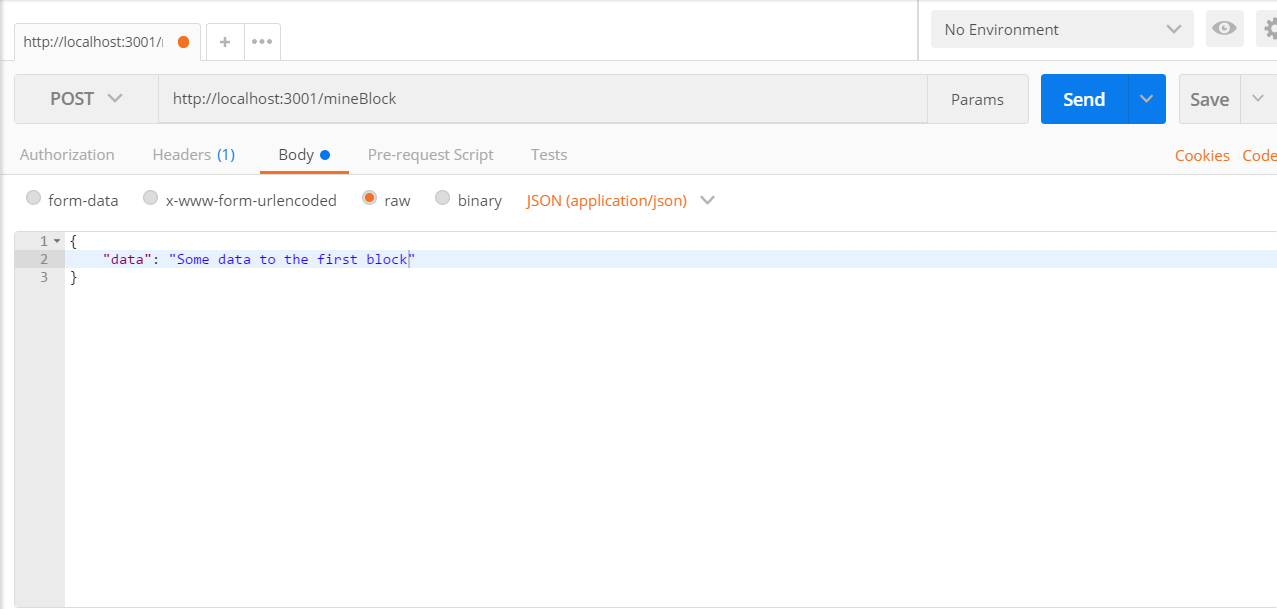


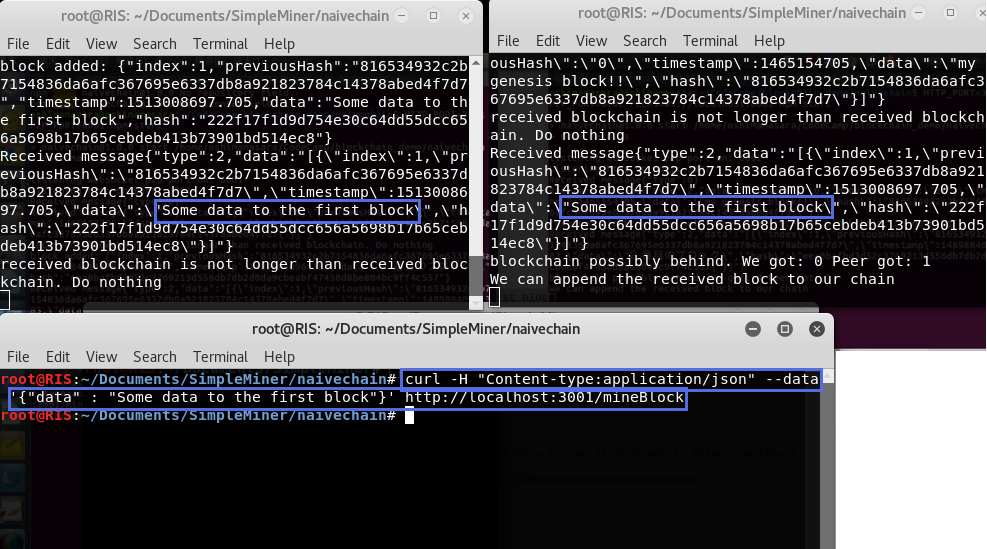
1. Now comes the most interestic part. Open the **third terminal** and write: "**curl -H "Content-type:application/json" --data '{"data" : "Some data to the first block"}' http://localhost:3001/mineBlock**". With this we command first node to **mine new block**. His **index** will be the index of previous block + 1. He contains data **"Some data to the first block**”.

**Linux:**

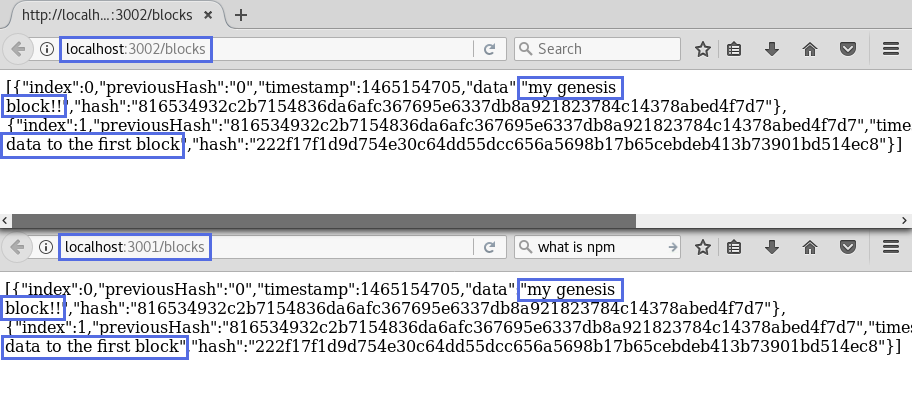
|  |
| --- |
| curl -H "Content-type:application/json" --data '{"data" : "Some data to the first block"}' http://localhost:3001/mineBlock |

**Windows:**





1. Let`s see how blockchain was changed. Open **both nodes addresses** in browser and get the **lists of all blocks.** The **new block** is here after the genesis block.

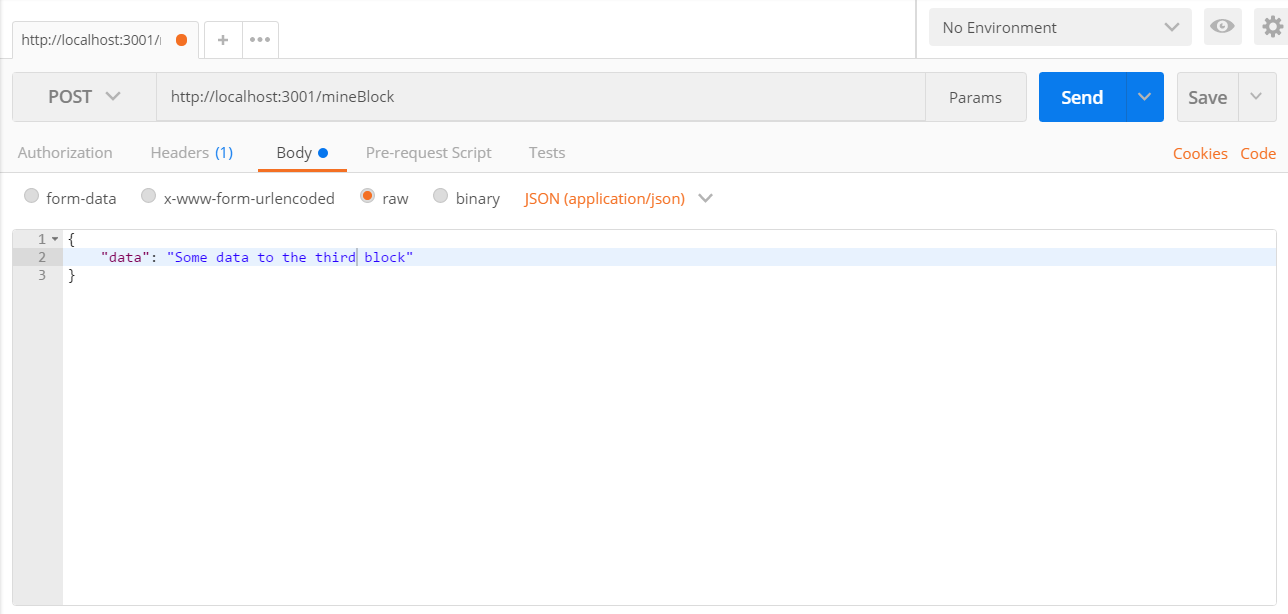


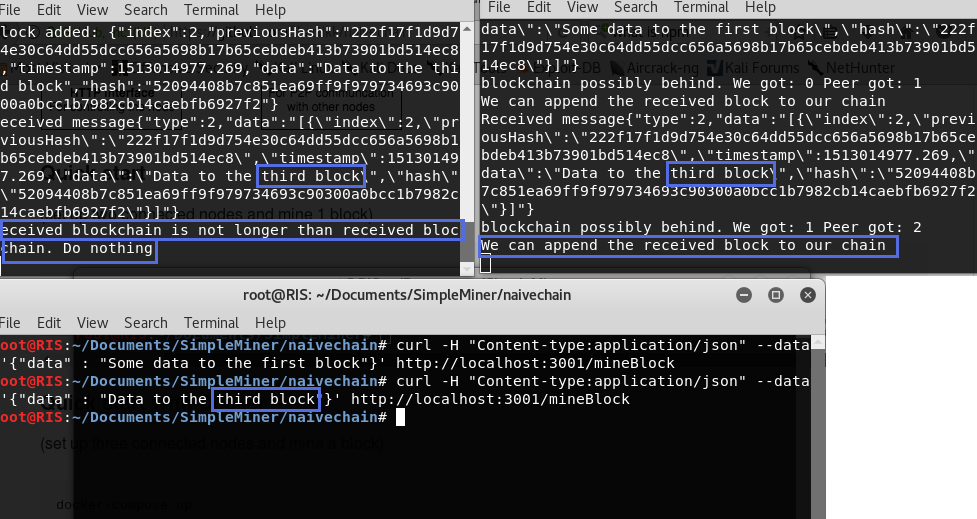
1. Now let`s mine **third block**. Type in terminal: "**curl -H "Content-type:application/json" --data '{"data" : "Data to the third block"}' http://localhost:3001/mineBlock**". Pay attention that the **first node** is the emitter of the block so he reports that new blocks sequence is no longer then existing blockchain and “**Received blockchain is not longer than received blockchain. Do nothing**”. But the other node report that “**We can append the received block to our chain**”.

**Linux:**

|  |
| --- |
| curl -H "Content-type:application/json" --data '{"data" : "Data to the third block"}' http://localhost:3001/mineBlock |

**Windows:**





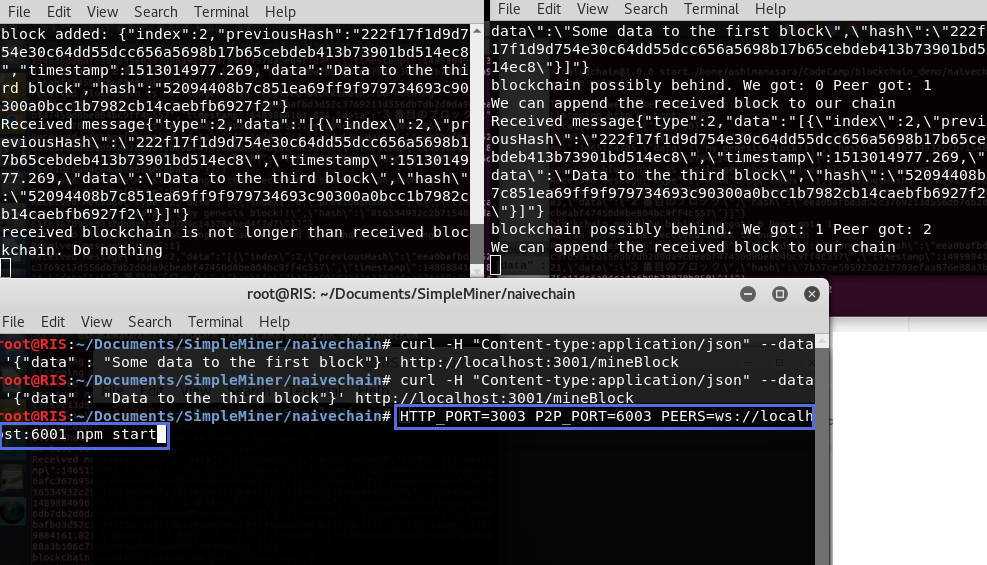
1. The next task is to create a **third node**. In terminal type the command: “**HTTP\_PORT=3003 P2P\_PORT=6003 PEERS=ws://localhost:6001 npm start**”. This is the screen before executing the command.

**Linux:**

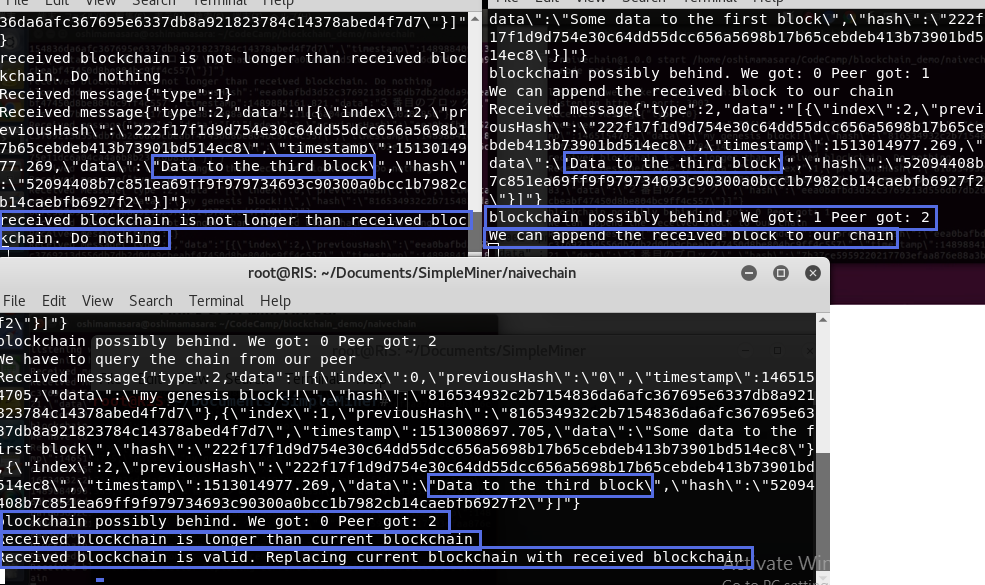
|  |
| --- |
| HTTP\_PORT=3003 P2P\_PORT=6003 PEERS=ws://localhost:6001 npm start |

**Windows:**

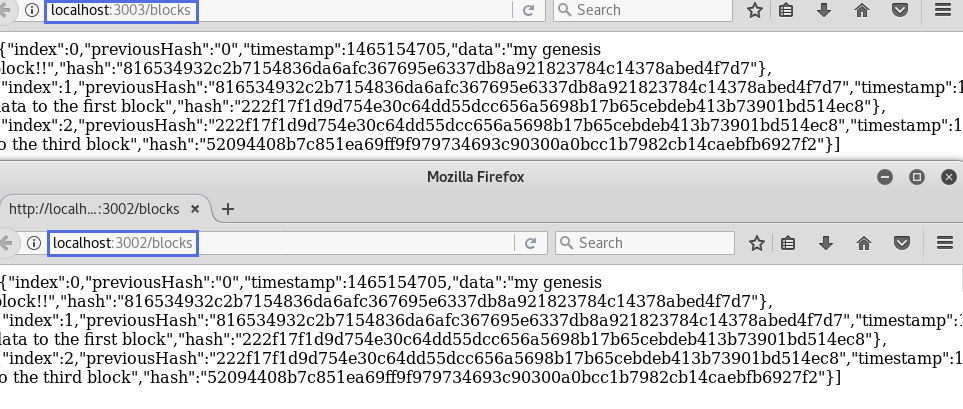
|  |
| --- |
| set HTTP\_PORT=3003 && set P2P\_PORT=6003 && set PEERS=http://localhost:6001 && npm start |



1. And this is the screen after enter key was pressed. The **third** node was created and participate in blockchain.



1. Open the **third node address** in the browser and compare with one of the old nodes. Third node has the **same blocks**.



## Follow the Described Steps

Your task is follow the tutorial described above and perform the experiments.

1. Create 3 nodes.
2. Mine two blocks. Enter your name in blocks “data” field.

# What to Submit?

Create a **zip file** (e.g. your-name-simple\_miner-exercise.zip) holding the screenshots with your experiments. Make screenshots of terminals and data in browser.

Submit your **zip** file as **homework** at the course Web site.