Step 1: Environment Setup

First, you need to install Geth which is a command-line interface (CLI) tool that communicates with the Ethereum network and acts as the link between your computer and the rest of the Ethereum nodes.

Go tothe[**Go Ethereum**](https://geth.ethereum.org/downloads/)site. Download and install the binary for your operating system.

Step 2: Configure the Genesis file

To run a private network, you need to provide **geth** with some basic information required to create the initial block. Every blockchain starts with a **Genesis Block,**the very first block in the chain. To create our private blockchain, we will create a genesis block with a custom genesis file. Then, ask Geth to use that genesis file to create our own genesis block.

The blockchain is a distributed digital register in which all transactions are recorded in sequential order in the form of blocks. There are a limitless number of blocks, but there is always one separate block that gave rise to the whole chain i.e.the genesis block.

Create new folder called AlphaChain for this project.

In this AlphaChain folder, to create a private blockchain, a genesis block is needed. To do this, create a genesis file, which is a JSON file with the following commands-

{

    "config": {

      "chainId": 1121,

      "homesteadBlock": 0,

      "eip150Block": 0,

      "eip155Block": 0,

      "eip158Block": 0,

      "byzantiumBlock": 0,

      "constantinopleBlock": 0,

      "petersburgBlock": 0,

      "ethash": {}

    },

    "difficulty": "0x400",

    "gasLimit": "0x8000000",

    "alloc": {}

  }

Step 3: Creating the Private AlphaChain

The Genesis block is the root block, i.e. the first block in the AlphaChain.

So genesis file and nodes exist in the same directory.

Once you done like this, you are ready to create your nodes.

To create your first node, open a new terminal window and navigate to your AlpahChain, and type in the following command:

geth init --datadir node1 genesis.json

This command will create a directory called **node1**in your project.

You type command like as above command to create 2 nodes.

geth init --datadir node2 genesis.json

geth init --datadir node3 genesis.json

Now that your Node 1, Node2, Node3 is initialized, let’s start Nodes using Geth. Within your each other terminal window, navigate to your project folder (where you saved your **genesis.json** file) and type in the following:

geth --datadir node1 -–ipcpath geth01 --networkid 2022 --nodiscover --allow-insecure-unlock --http

Above is the command line to start node 1. The default listening port is **30303**, and the default rpc port is **8545**. The **ipcpath** name must be a unique pipe name.

Type command like as above commands in each other console.

geth --datadir node2 -–ipcpath geth02 --networkid 2022 --port 30304 --authrpc.port 8552 --nodiscover --allow-insecure-unlock

geth --datadir node3 -–ipcpath geth03 --networkid 2022 --port 30305 --authrpc.port 8553 --nodiscover --allow-insecure-unlock

With three nodes started up, they are running as a server process, waiting for events and commands to be executed. You should not interrupt the server process by entering commands in its console window. Instead, you should send commands to it from another client process.

This is achieved by starting another geth process as a client and connecting it to the geth server process.

Once attached, you can then send interactive commands to the server process. You can do this by first attaching your geth client to a geth server process.

This command line attaches to geth node 1 via namepipe:

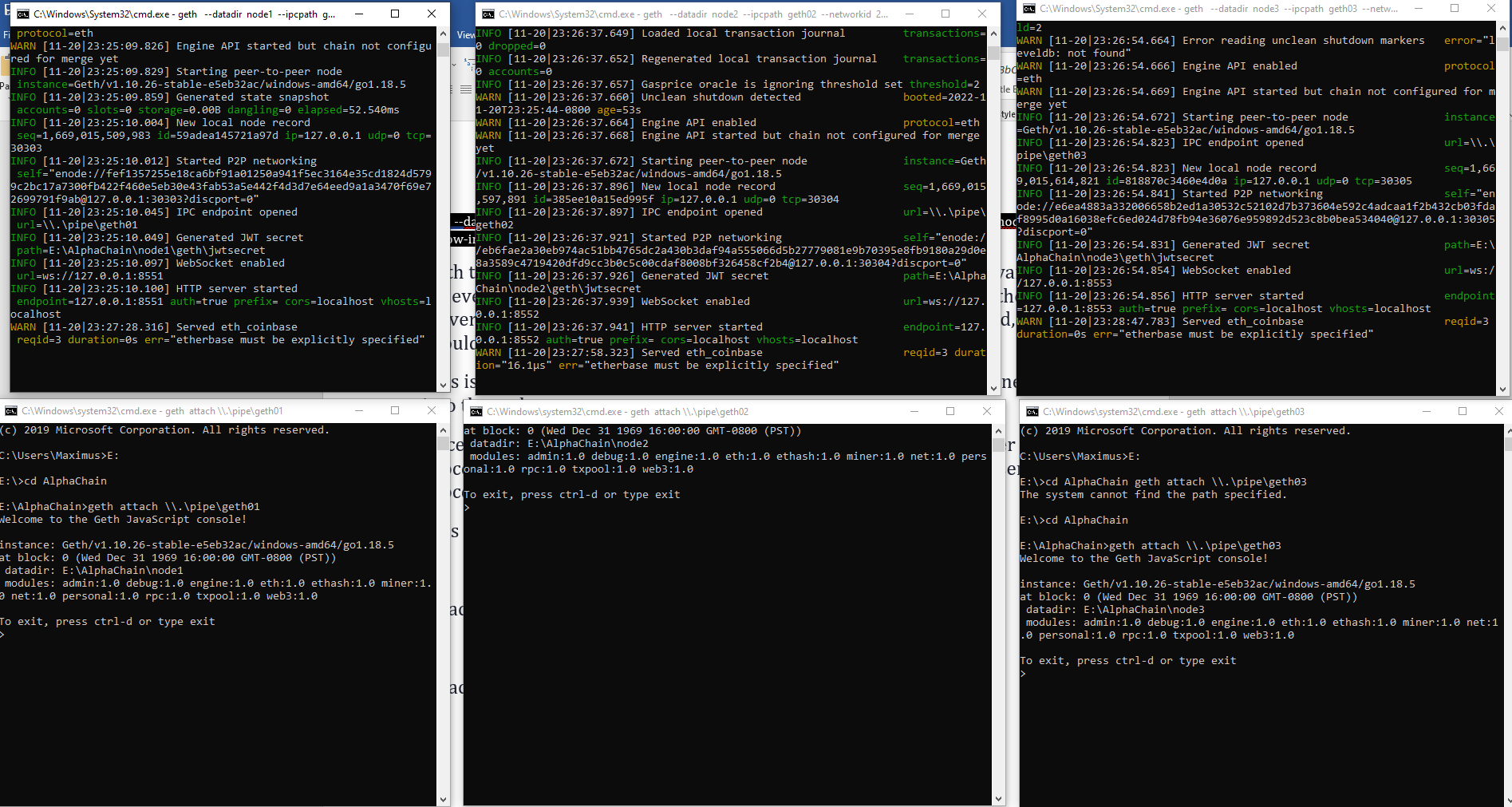
geth attach \\.\pipe\**geth01**

Attaching to geth node 2 via namepipe:

geth attach \\.\pipe\**geth02**

Attaching to geth node 3 via namepipe:

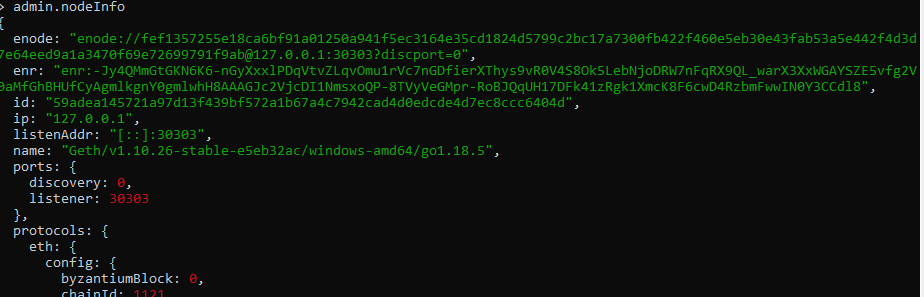
geth attach \\.\pipe\**geth03**



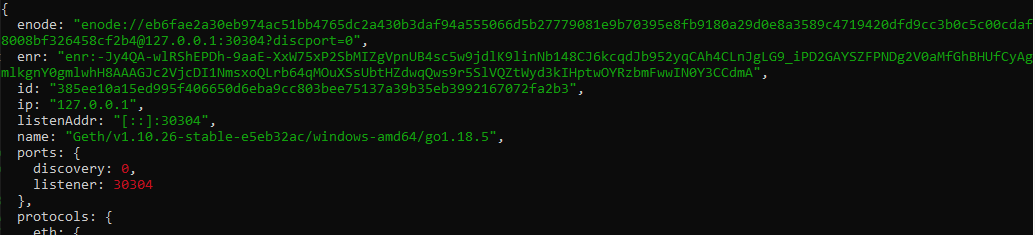
**Adding Peer Nodes Manually**

The two running nodes do not know each other even though we have specified that they are in the same network, with id 1121. For now, we are not bootstrapping nodes on startup, i.e. not configuring known static nodes on startup. We will add a peer node manually.

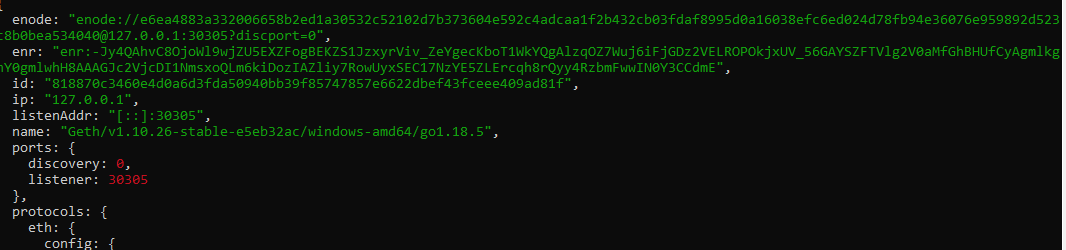
To get the node information of node 1, enter the function admin.nodeInfo from the node 1 client console:



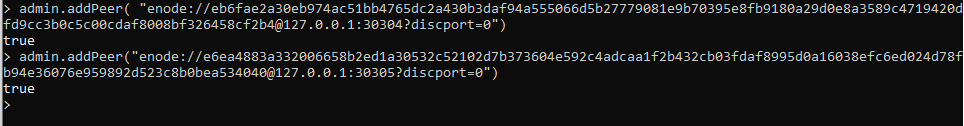
To get the node information of node 2, enter the function admin.nodeInfo from the node 2 client console:



To get the node information of node 3, enter the function admin.nodeInfo from the node 3 client console:



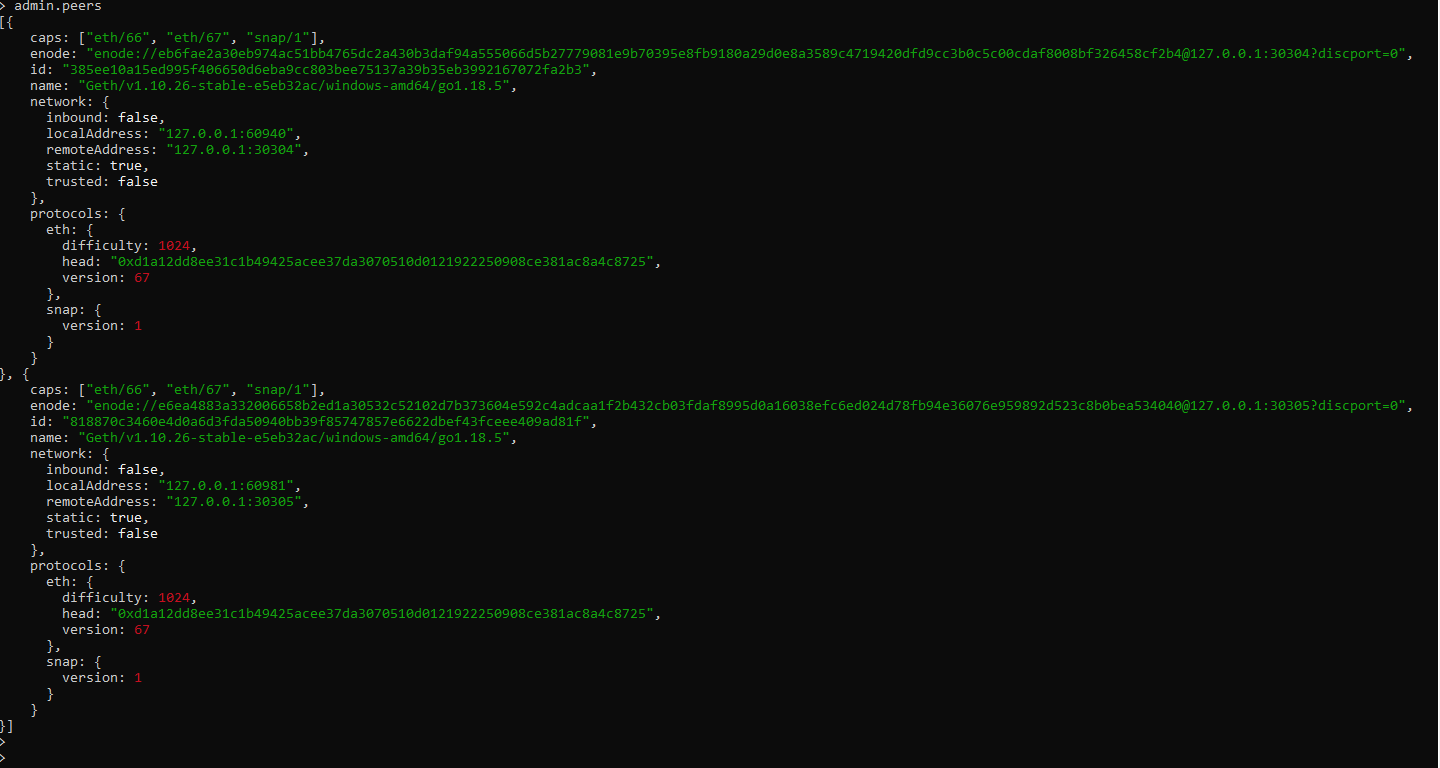
In node 1, you can add node 2 , node3 by using the admin.addPeer function by specifying the entire enode data of node 2,node3 as below. This function will return "true":



In node2, you can add node 3, by using the admin.addPeer function by specifying the entire enode data of node3 as below.

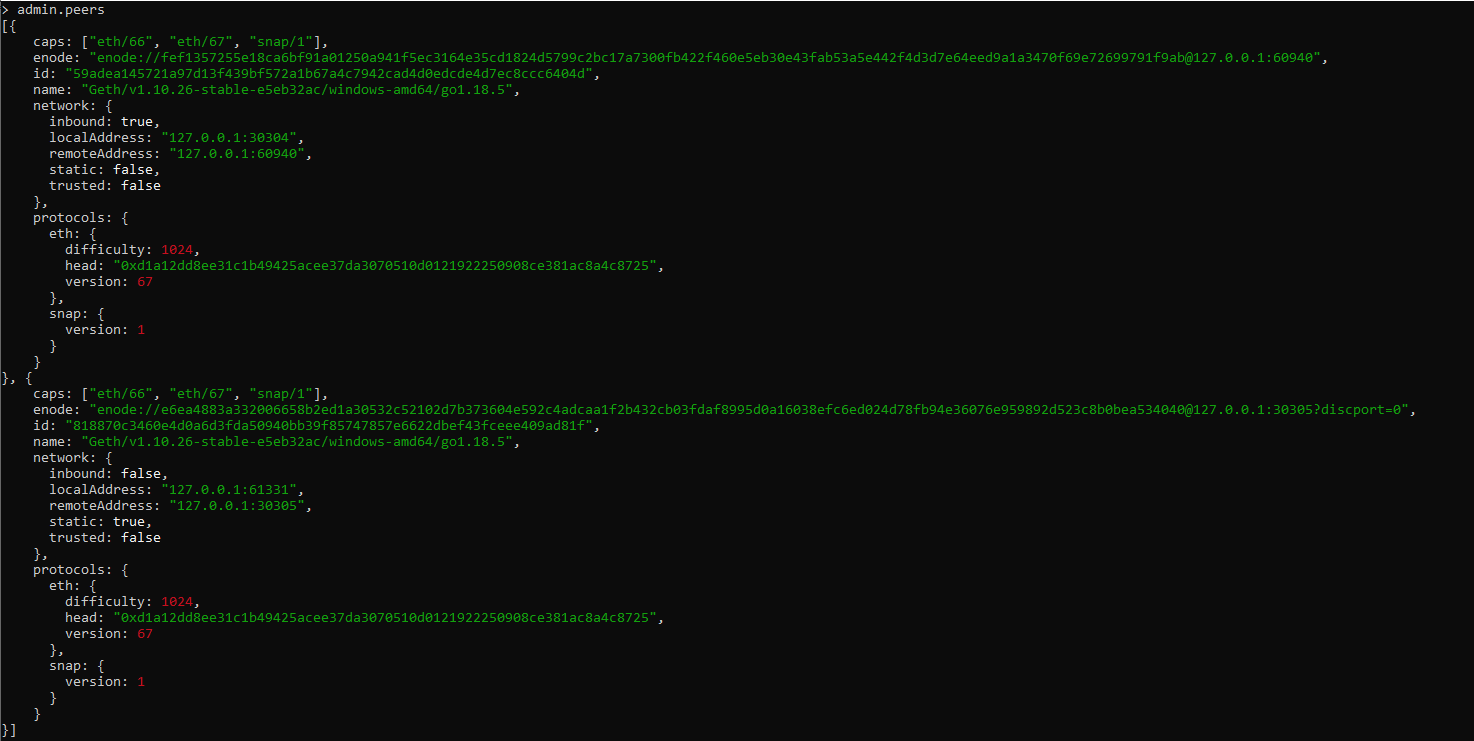
Once added, both nodes will recognize each other. You can verify this using the admin.peers function.

From node 1:



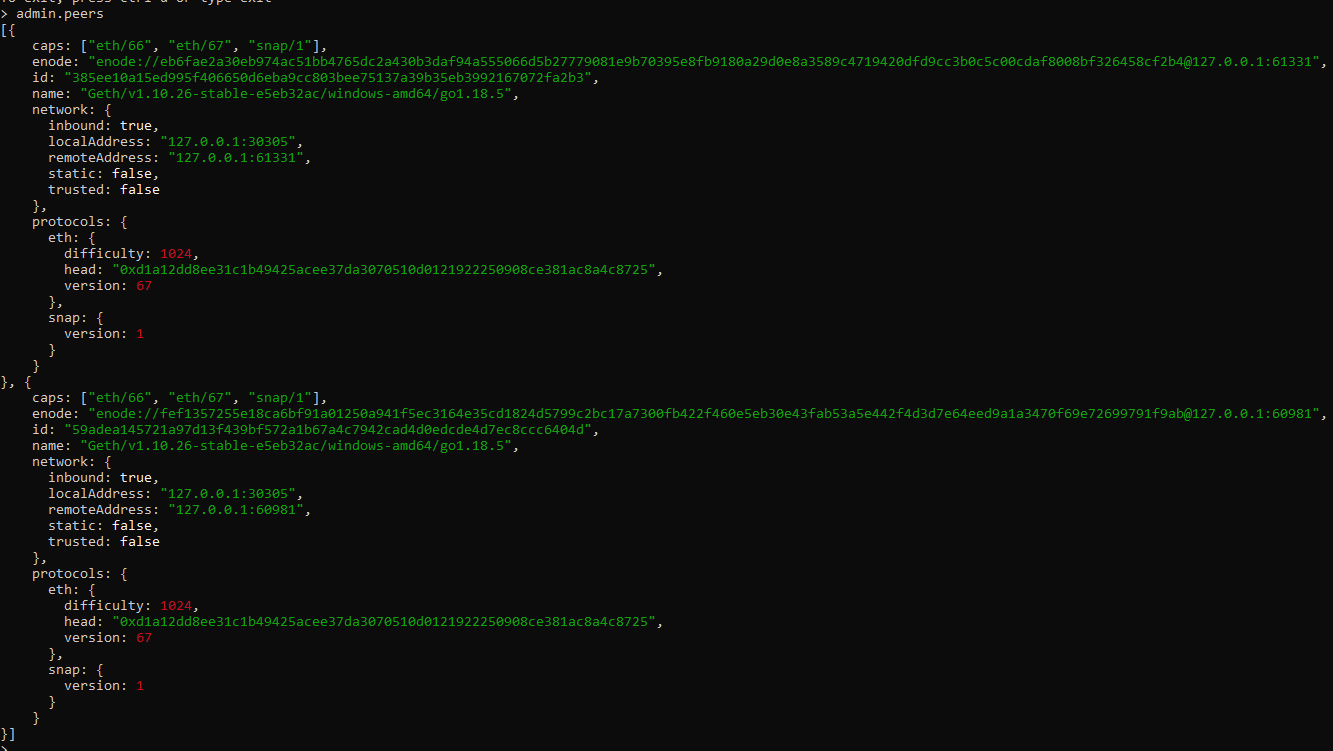


From node 2:





From node 3:





#### Add a New Account

We have to add our first account to the blockchain of node 1 so that the account can be used for executing transactions. This is the External account, owned by a person.

From node 1, enter the function personal.newAccount and provide your password. This function will return the address of the newly created account.



The string of numbers and letters is the address of the new account in node 1.

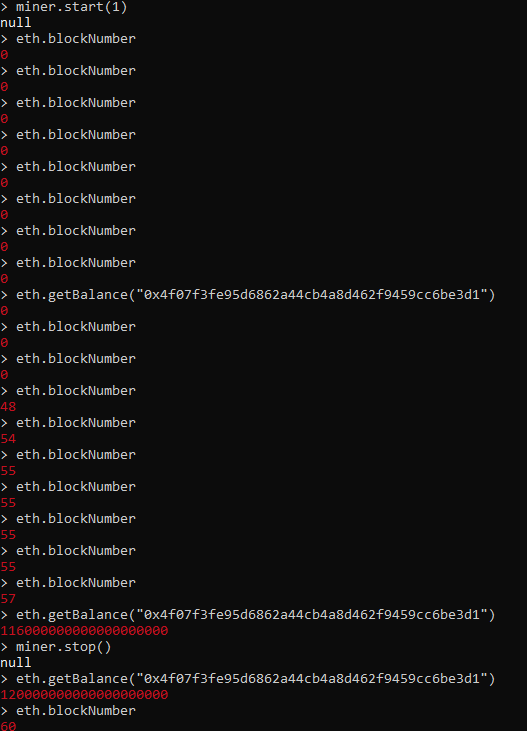
**Important: There is no way to restore your account if you forget your password. Never forget your password. Keep it in a safe place, and do not store it on your machine.**

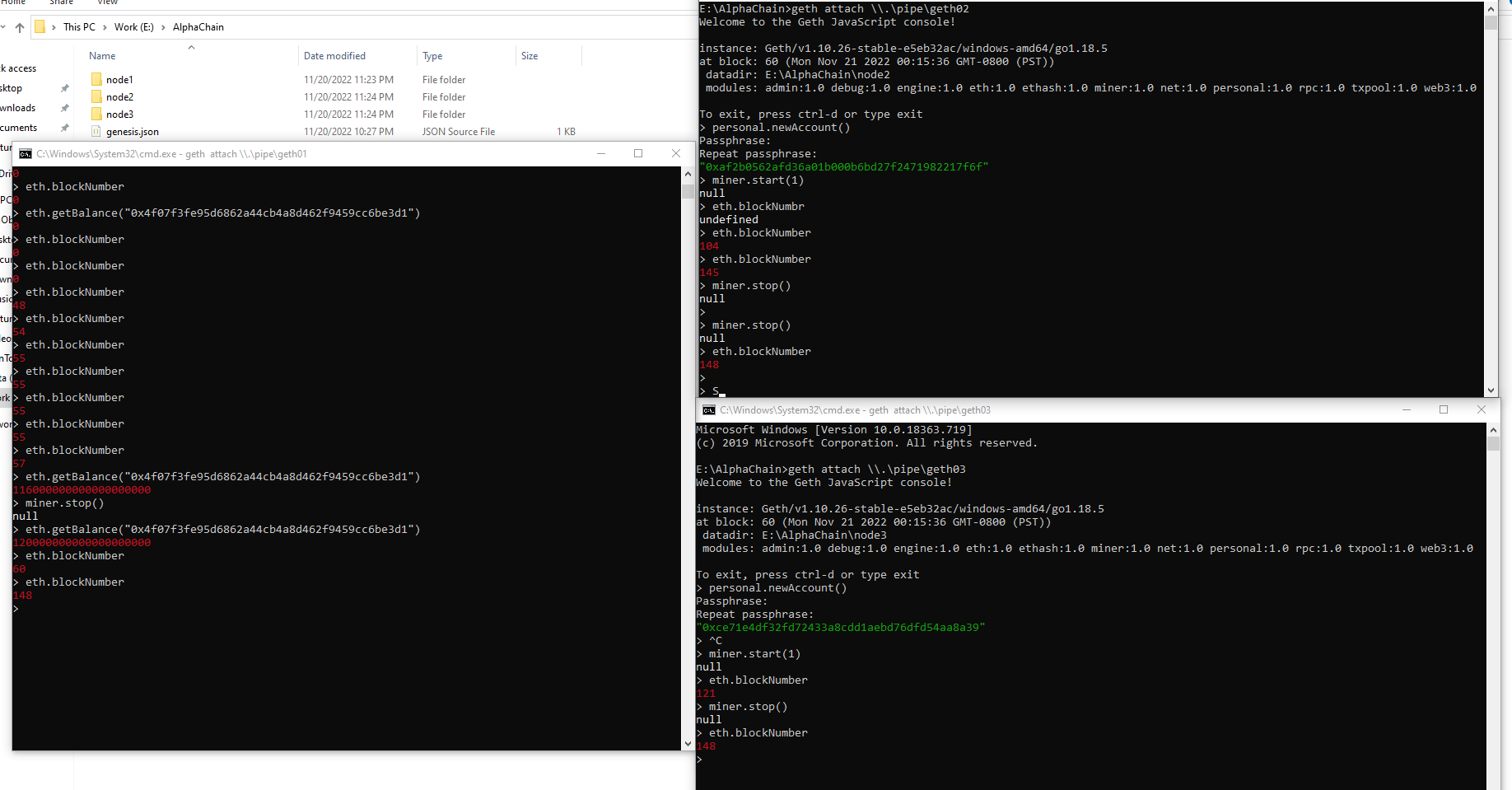
#### 

#### Eventual Consistency

Below is a demonstration of eventual consistency between the three nodes.

Start mining on node 1 by using the function miner.start(1), where 1 refers to the number of threads. Note that the miner.start(n) function will always return "null." Unless you have many CPU cores, keep the thread number low to avoid high CPU usage. Note that mining without any pending transaction can still earn your default account incentive (ETH). It creates empty blocks, thus strengthening the integrity of the blockchain tree.



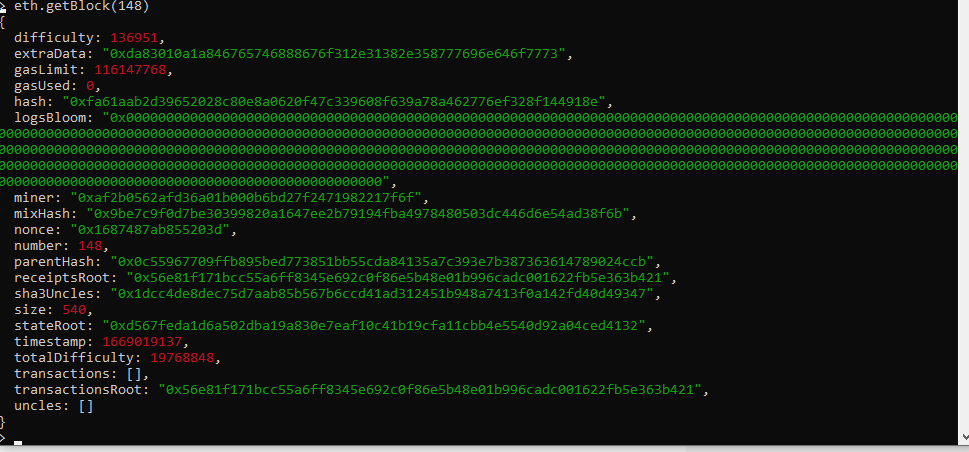


All nodes will have the same number of blocks in their blockchain.

From either node, as shown below, you can query the number of blocks by using the function eth.blockNumber.

Both nodes will get the same block per block number,

e.g. hash shown below, being mined by node 2:



Alternatively, you can pass in the last block number to the eth.getBlock function and get the miner address directly:

mined by node 2 account

Step 4: Connecting Metamask

