

[Solange Gueiros](https://medium.com/@solangegueiros?source=post_header_lockup)Follow

Blockchain enthusiast, linkedin: linkedin.com/in/solangegueiros

Sep 24

**The Geth’s saga: setting up Ethereum private network on windows**

When I tried to setup a Ethereum private network on windows, I had some difficulties, mainly because I did not find information to do on windows. I tried many things to make everything work easily, and I called this process the Geth’s saga.

In an Ethereum private network, the nodes are not connected to the main network on internet. In this context private does not mean protected or secure, but it means reserved or isolated.

In this tutorial I’ll show you step by step how to get started with Ethereum private network by setting up a main node and a second node in another computer / IP, both in same network.

**Pre requirements**

Install geth: <https://github.com/ethereum/go-ethereum/wiki/geth>

I’m using version 1.6.7

**Things to define before start**

Custom Genesis File (it is the next topic)

Custom Data Directory (for me: “C:\ETH\”)

Custom NetworkID (for me: 13)

(Recommended) Disable Node Discovery

**Creating The Genesis Block**

Every blockchain starts with the genesis block.

Here’s an example of a custom genesis.json file:

{

"nonce" : "0x0000000000000055",

"mixHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",

"parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",

"difficulty": "0x20000",

"gasLimit" : "0x800000",

"timestamp" : "0x0",

"extraData" : "",

"coinbase" : "0x0000000000000000000000000000000000000000",

"alloc" : {},

"config" : {

"chainId": 100,

"homesteadBlock": 0,

"eip155Block": 0,

"eip158Block": 0

}

}

Pay attention with the character “, it can be changed when you copy and paste, maybe I´ve got this error: “Fatal: invalid genesis file: invalid character ‘“‘ after object key:value pair”.

I saved my genesis.json at “C:\ETH\configs\genesis.json”

**Parameters at genesis.json**

Extract from: <https://ethereum.stackexchange.com/questions/2376/what-does-each-genesis-json-parameter-mean>

**mixhash** A 256-bit hash which proves, combined with the nonce, that a sufficient amount of computation has been carried out on this block: the Proof-of-Work (PoW). The combination of nonce and mixhash must satisfy a mathematical condition described in the Yellowpaper, 4.3.4. Block Header Validity, (44). It allows to verify that the Block has really been cryptographically mined, thus, from this aspect, is valid.

**nonce** A 64-bit hash, which proves, combined with the mix-hash, that a sufficient amount of computation has been carried out on this block: the Proof-of-Work (PoW). The combination of nonce and mixhash must satisfy a mathematical condition described in the Yellowpaper, 4.3.4. Block Header Validity, (44), and allows to verify that the Block has really been cryptographically mined and thus, from this aspect, is valid. The nonce is the cryptographically secure mining proof-of-work that proves beyond reasonable doubt that a particular amount of computation has been expended in the determination of this token value. (Yellowpager, 11.5. Mining Proof-of-Work).

**parentHash** The Keccak 256-bit hash of the entire parent block header (including its nonce and mixhash). Pointer to the parent block, thus effectively building the chain of blocks. In the case of the Genesis block, and only in this case, it’s 0.

**difficulty** A scalar value corresponding to the difficulty level applied during the nonce discovering of this block. It defines the mining Target, which can be calculated from the previous block’s difficulty level and the timestamp. The higher the difficulty, the statistically more calculations a Miner must perform to discover a valid block. This value is used to control the Block generation time of a Blockchain, keeping the Block generation frequency within a target range. On the test network, we keep this value low to avoid waiting during tests, since the discovery of a valid Block is required to execute a transaction on the Blockchain.

**gasLimit** A scalar value equal to the current chain-wide limit of Gas expenditure per block. High in our case to avoid being limited by this threshold during tests. Note: this does not indicate that we should not pay attention to the Gas consumption of our Contracts.

**timestamp** A scalar value equal to the reasonable output of Unix time() function at this block inception. This mechanism enforces a homeostasis in terms of the time between blocks. A smaller period between the last two blocks results in an increase in the difficulty level and thus additional computation required to find the next valid block. If the period is too large, the difficulty, and expected time to the next block, is reduced. The timestamp also allows verifying the order of block within the chain (Yellowpaper, 4.3.4. (43)).

**extraData** An optional free, but max. 32-byte long space to conserve smart things for ethernity.

**coinbase** The 160-bit address to which all rewards (in Ether) collected from the successful mining of this block have been transferred. They are a sum of the mining reward itself and the Contract transaction execution refunds. Often named “beneficiary” in the specifications, sometimes “etherbase” in the online documentation. This can be anything in the Genesis Block since the value is set by the setting of the Miner when a new Block is created.

My note: It is no problem if you don’t have the address now because you can define the address when you start geth to mine.

**alloc** Allows defining a list of pre-filled wallets. That’s an Ethereum specific functionality to handle the “Ether pre-sale” period. Since we can mine local Ether quickly, we don’t use this option.

**config** Configuration to describe the chain itself. Specifically the chain ID, the consensus engines to use, as well as the block numbers of any relevant hard forks.

Chainid — identifies the current chain and is used for replay protection. This is a unique value for your private chain.

homesteadBlock — your chain won’t be undergoing the switch to Homestead, so leave this as 0.

eip155Block — your chain won’t be hard-forking for these changes, so leave as 0.

eip158Block — your chain won’t be hard-forking for these changes, so leave as 0.

This is all the chain configuration field, defined in [config.go](https://github.com/ethereum/go-ethereum/blob/feeccdf4ec1084b38dac112ff4f86809efd7c0e5/params/config.go#L71):

// ChainConfig is the core config which determines the blockchain settings.

//

// ChainConfig is stored in the database on a per block basis.

// This means that any network, identified by its genesis block,

// can have its own set of configuration options.

type ChainConfig struct {

ChainId \*big.Int 'json:"chainId"' // Chain id identifies the current chain and is used for replay protection

HomesteadBlock \*big.Int 'json:"homesteadBlock,omitempty"' // Homestead switch block (nil = no fork, 0 = already homestead)

DAOForkBlock \*big.Int 'json:"daoForkBlock,omitempty"' // TheDAO hard-fork switch block (nil = no fork)

DAOForkSupport bool 'json:"daoForkSupport,omitempty"' // Whether the nodes supports or opposes the DAO hard-fork

// EIP150 implements the Gas price changes (https://github.com/ethereum/EIPs/issues/150)

EIP150Block \*big.Int 'json:"eip150Block,omitempty"' // EIP150 HF block (nil = no fork)

EIP150Hash common.Hash 'json:"eip150Hash,omitempty"' // EIP150 HF hash (fast sync aid)

EIP155Block \*big.Int 'json:"eip155Block,omitempty"' // EIP155 HF block

EIP158Block \*big.Int 'json:"eip158Block,omitempty"' // EIP158 HF block

// Various consensus engines

Ethash \*EthashConfig 'json:"ethash,omitempty"'

Clique \*CliqueConfig 'json:"clique,omitempty"'

}

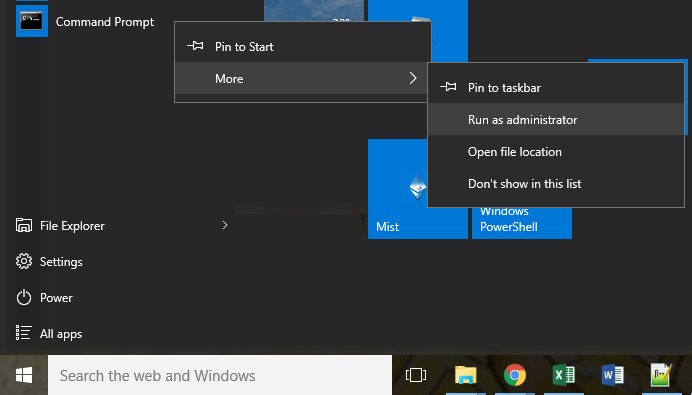
**Init GETH to create the genesis block**

Remember my settings:

genesis.json is in “C:\ETH\configs\genesis.json”

datadir is “C:\ETH\data-private”

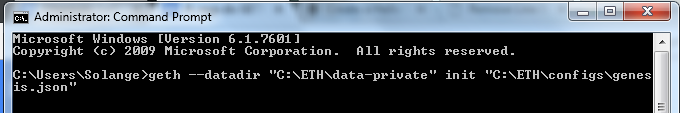
At windows, run **Command Prompt** or **PowerShell** with administrator privileges:



running **Command Prompt** with administrator privileges

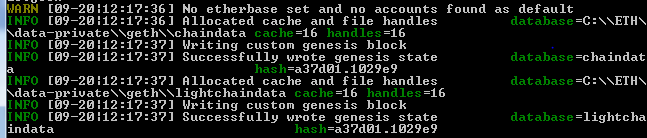
Execute geth with the parameters *—* *datadir* and *init:*

geth --datadir "C:\ETH\data-private" init "C:\ETH\configs\genesis.json"



executing geth with the parameters — datadir and init

You should expect to see output similar to this:



result of geth with the parameters — datadir and init

**Resolving the warning: execute geth to create account**

WARN [09–20|12:17:36] No etherbase set and no accounts found as default

Let’s create the account that will be used for mining.

**Creating account at geth**

Run geth with the following parameters:

geth --networkid 13 --port 60303 --rpc --lightkdf --cache 16 --datadir "C:\ETH\data-private" console

**Parameters**

Here are explanations about the parameters used.

More info: <https://github.com/ethereumproject/go-ethereum/wiki/Command-Line-Options>

**— networkid** Network identifier (integer, 0=Olympic, 1=Homestead, 2=Morden) (default: 1). For a private network you define another number. I defined 13 for me

**— port** This is the “network listening port”, which you will use to connect with other peers manually.

**— rpc** Enable the HTTP-RPC server.

**— rpcaddr** HTTP-RPC server listening interface (default: “localhost”).

**— rpcport** HTTP-RPC server listening port (default: 8545).

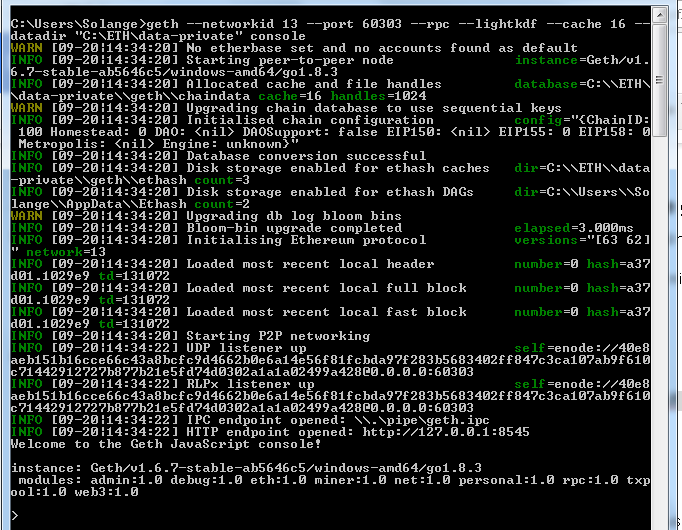
**— lightkdf** Reduce key-derivation RAM & CPU usage at some expense of KDF strength.

**— cache** Megabytes of memory allocated to internal caching (min 16MB / database forced) (default: 128)

**— datadir** Data directory for the databases and keystore. Choose a location that is separate from your public Ethereum chain folder.

**console** Geth Console: interactive JavaScript environment

You’ll see something like this:



Executing geth with console

**Create new Account**

At geth console:

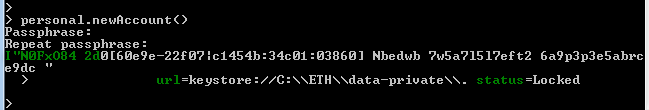
>

>personal.newAccount()

Passphrase: \*\*\*\*\*

Repeat passphrase: \*\*\*\*\*

Define your password and don’t forget it!



Creating an account

**Check your account**

At geth console:

>

>personal

Check the listAccounts:



Executing personal at geth console

In this case, the account is:

0x842d06ee2f7c45b3c1080bdb75757f269335bc9c

**Configure enode**

Enode’s configuration will be used when other computers connect to the private network.

An enode is a way to describe an Ethereum node in the form of a URI, separated from the host by an @ sign.

The hexadecimal node ID is encoded in the username portion of the URL. The username portion is a 512-bit public key that is used to verify communication came from a particular node on the network.

The hostname can only be given as an IP address, DNS domain names are not allowed. The port in the host name section is the TCP listening port. If the TCP and UDP (discovery) ports differ, the UDP port is specified as query parameter “discport”.

At geth console, run this command and verify the node information:

>admin.nodeInfo.enode



Running admin.nodeInfo.enode at geth console

This is the enode:

"enode://ba4a9b4e8fb7a46776ad1f3ace7cb63fab5c50b185516b01515de12b2d7adacc525e69c133b01611e1a7009a18121c5e0b7f9950c3b70f9124a1a784469e96e6@0.0.0.0:60303"

**IP and port for enode**

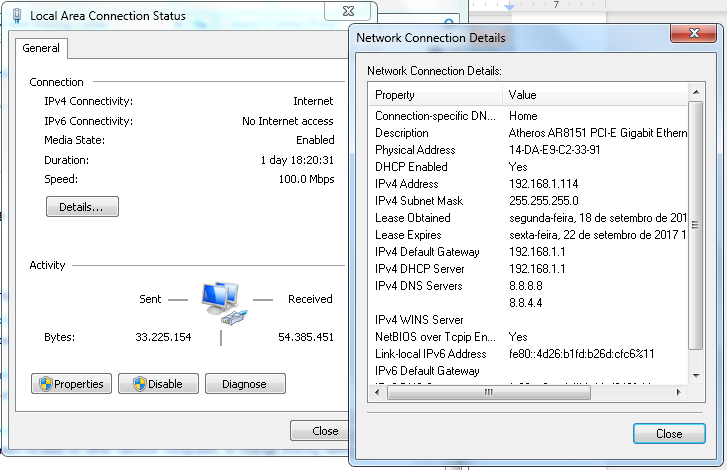
Pay attention at port at enode’s end.

This is the port defined when execute geth:

geth — networkid 13 **— port 60303** — rpc — lightkdf — cache 16 — datadir “C:\ETH\data-private” console

So, I´m using port 60303.

Now check your computer’s IP.



Check computer’s IP

My IP is 192.168.1.114

Copy this information at enode:

IP: change 0.0.0.0 to 192.168.1.114

Port: 60303

This is the enode with changes:

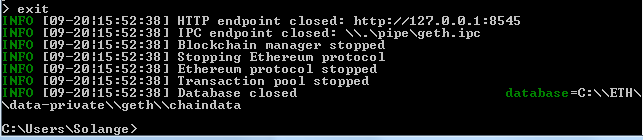
"enode://ba4a9b4e8fb7a46776ad1f3ace7cb63fab5c50b185516b01515de12b2d7adacc525e69c133b01611e1a7009a18121c5e0b7f9950c3b70f9124a1a784469e96e6@**192.168.1.114:60303"**

Keep this information to use at second node.

I finished the configurations at principal node.

Now exit console to run it again with parameters to mine

>exit

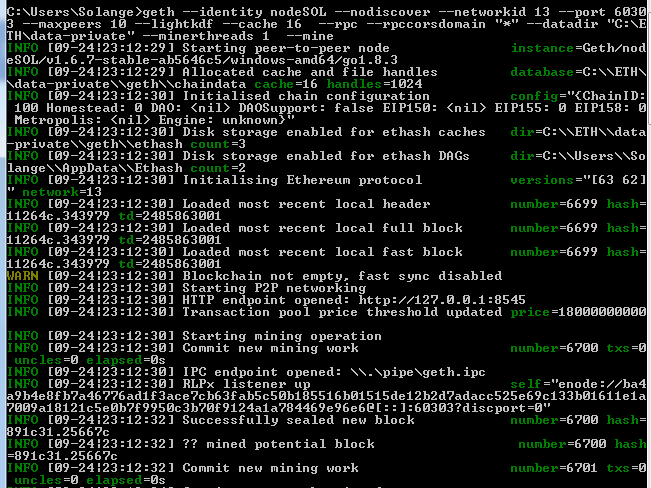


Exiting geth console

**Execute node to mine**

Now we are already to mine. I will execute geth again, but adding some parameters:

geth --identity nodeSOL --nodiscover --networkid 13 --port 60303 --maxpeers 10 --lightkdf --cache 16 --rpc --rpccorsdomain "\*" --datadir "C:\ETH\data-private" --minerthreads 1 --mine



Execute geth to mine

Explaining the parameters:

**— identity** This will set up an identity for your node so it can be identified more easily in a list of peers.

**— nodiscover** Use this to make sure that your node is not discoverable by people who do not manually add you. Otherwise, there is a chance that your node may be inadvertently added to a stranger’s blockchain if they have the same genesis file and network id.

**— rpccorsdomain** Comma separated list of domains from which to accept cross origin requests (browser enforced)

**— rpcapi** API’s offered over the HTTP-RPC interface (default: “eth,net,web3”).

**— minerthreads** Number of CPU threads to use for mining (default: 2)

**— mine** Enable mining. You may not start mining now and wait until the second node is connected. This way you will have fewer blocks to synchronize at when the second node starts.

You can use etherbase to force mining to your account:

**— etherbase** Public address for block mining rewards (default = first account created) (default: “0”).

geth --identity nodeSOL --nodiscover --networkid 13 --port 60303 --maxpeers 10 --lightkdf --cache 16 --rpc --rpccorsdomain "\*" --datadir "C:\ETH\data-private" --etherbase "0x91e516b943c032d843356fa590bb2d56d52eb72e" --minerthreads 1 --mine

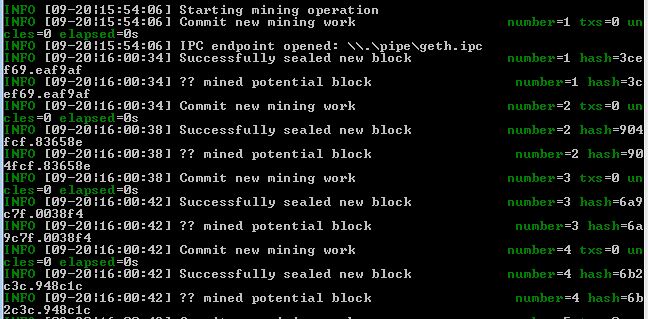
Pay attention to this line when run geth to mine at the first time:



Starting mining operation

It means that it is initializing mining. It may take a few minutes.

And… the first block is mined!



The first block is mined!

**Attach to geth console in another window**

I already have a geth node running, so I can attach another geth instance to interact with it.

I will start geth with parameters:

**— ipcpath** Filename for IPC socket/pipe within the datadir (explicit paths escape it)

**attach** Geth Console: interactive JavaScript environment (connect to node)

It is important to attach at your previous datadir!

At windows, run another instance of **Command Prompt** or **PowerShell** with administrator privileges and run:

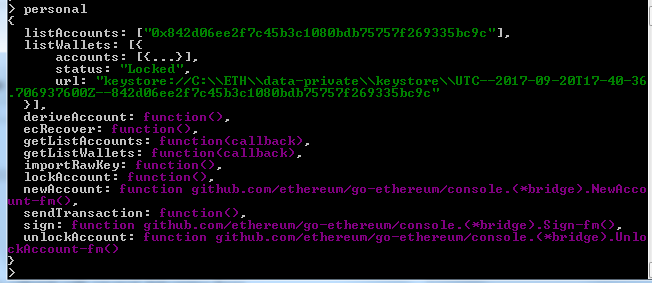
geth --ipcpath geth.ipc --datadir "C:\ETH\data-private" attach

To check if you are connect at the correct node, run this command at geth console:

> personal

Check your account number, my is:

0x842d06ee2f7c45b3c1080bdb75757f269335bc9c



run personal at geth

**Configure another node**

Remember the first node’s configurations:

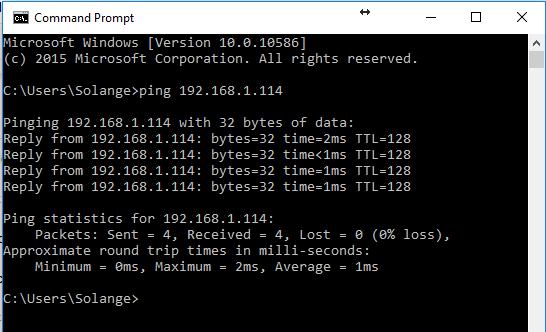
IP: 192.168.1.114

enode://ba4a9b4e8fb7a46776ad1f3ace7cb63fab5c50b185516b01515de12b2d7adacc525e69c133b01611e1a7009a18121c5e0b7f9950c3b70f9124a1a784469e96e6@**192.168.1.114:60303**

First of all, I recommend testing ping access on another computer that you would like to connect to the first node.

At windows, run **Command Prompt** or **PowerShell:**

ping 192.168.1.114



ping at command pronpt

**Creating The Genesis Block**

The genesis block is the same in all nodes at private network.

{

"nonce" : "0x0000000000000055",

"mixHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",

"parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",

"difficulty": "0x20000",

"gasLimit" : "0x800000",

"timestamp" : "0x0",

"extraData" : "",

"coinbase" : "0x0000000000000000000000000000000000000000",

"alloc" : {}

"config" : {

"chainId": 100,

"homesteadBlock": 0,

"eip155Block": 0,

"eip158Block": 0

}

}

II saved my genesis.json at “C:\ETH2\configs\genesis.json” in seconde node

**Init GETH to create the genesis block at node 2**

Remember my settings:

genesis.json is in “C:\ETH2\configs\genesis.json”

datadir is “C:\ETH2\data-private”

At windows, run **Command Prompt** or **PowerShell** with administrator privileges and execute geth with the parameters *—* *datadir* and *init*

geth --datadir "C:\ETH2\data-private" init "C:\ETH2\configs\genesis.json"

**Creating account at geth at node 2**

Run geth with the following parameters:

geth --networkid 13 --port 60303 --rpc --lightkdf --cache 16 --datadir "C:\ETH2\data-private" console

At geth console:

>

>personal.newAccount()

Passphrase: \*\*\*\*\*

Repeat passphrase: \*\*\*\*\*

Define your password and don’t forget it!

Check your account at geth console:

>

>personal



Executing personal at geth console — node 2

My account at node 2 is:

0x4c549625a09cff4831350e5400537674b7e1ac37

Exit geth console.

**Clock error**

Pay attention to this warning at geth console:



Waning: System clock seems off by …

The nodes don’t synchronize when You have this warning.

You have to synchronize clock with internet time.

**Connect at another computer**

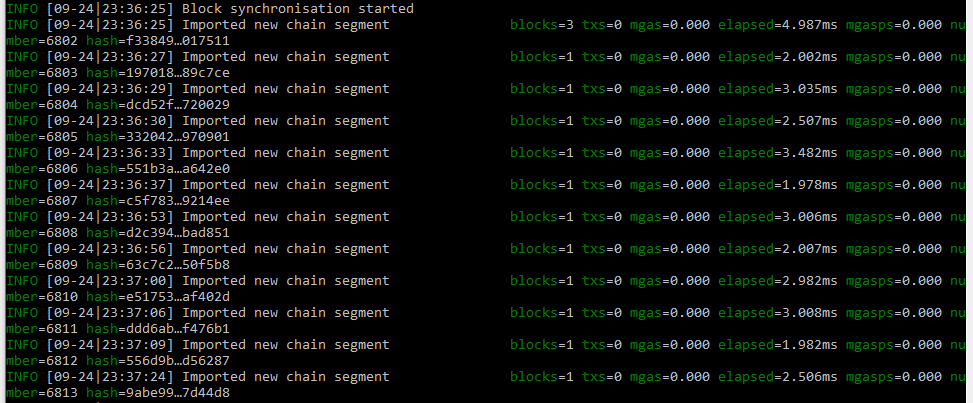
It´s time to use enode info to execute geth connecting the private network.

geth --networkid 13 --port 60303 --rpc --rpcport 8545 --rpccorsdomain "\*" --datadir "C:\ETH2\data-private" --minerthreads 1 --bootnodes "enode://ba4a9b4e8fb7a46776ad1f3ace7cb63fab5c50b185516b01515de12b2d7adacc525e69c133b01611e1a7009a18121c5e0b7f9950c3b70f9124a1a784469e96e6@192.168.1.114:60303"

I started geth with a new parameter:

**— bootnodes** Comma separated enode URLs for P2P discovery bootstrap. Remember that we are “nodiscovery” and another computer can connect at first node only with enode.

In a few minutes you will see the second node synchronize with the first node!



Block synchronisation started

**Attach to geth console in another window**

At windows, run another instance of **Command Prompt** or **PowerShell** with administrator privileges and run:

geth — ipcpath geth.ipc — datadir "C:\ETH2\data-private" attach

**Check your private network**

At geth console:

net.peerCount

You will see how many nodes you have in your network.



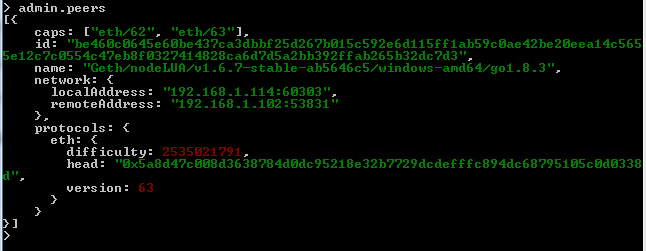
net.peerCount

We have the miner and one more node here.

admin.peers

Show all nodes at the network.

At first node:



admin.peers at first node

And this is executing at second node:



admin.peers at second node

You can see that in the first node we have nodeLUA connected and in the second node we have nodeSOL connected.

**Wrap up**

Now you know how to set up a private network and connect a node in one computer to a second node im another computer, including mine ethers and synchronize nodes.

Thank you taking the time to read this post. This is my first article and first tutorial. I hope the Geth’s saga has been helpful and I’d appreciate any of your feedback. Share it if you like it :)