CSED 490U Blockchain & Cryptocurrency

Assignment 5

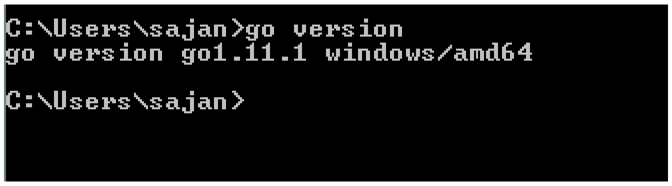
Submitted by- Sajan Maharjan

POVIS id- thesajan@postech.ac.kr

Registration Number- 20182095

<Step. 1> Install Go language

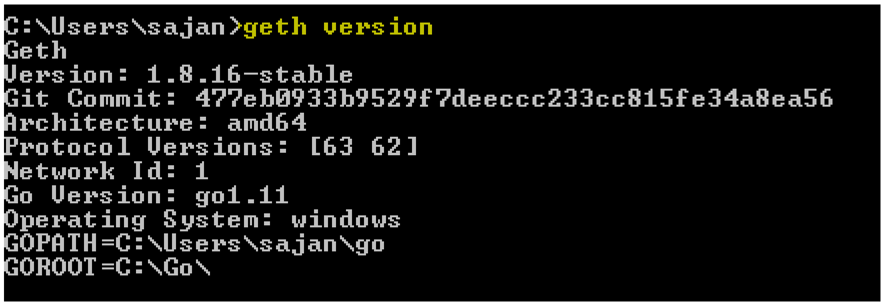
Installation of the Go programming language was completed. The Path variable for “go” command was automatically added after installation. Version 1.11.1 was installed



```
C:\Users\sajan>go version
go version go1.11.1 windows/amd64

C:\Users\sajan>
```

<Step. 2> Install go-ethereum (geth)

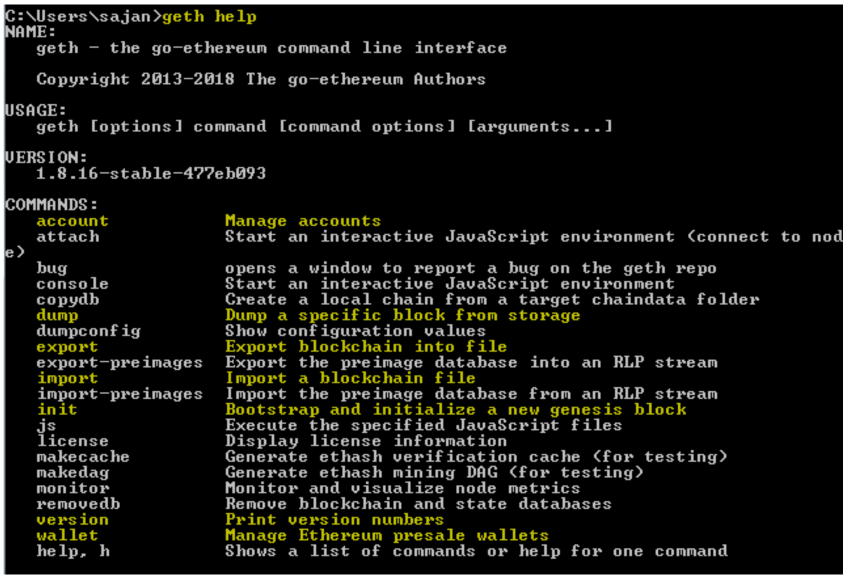
Installation of go-ethereum was completed using windows installer file. The Path variable for “geth” command was automatically added after installation. Version 1.8.16 was installed



```
C:\Users\sajan>geth version
Geth
Version: 1.8.16-stable
Git Commit: 477eb0933b9529f7deeccc233cc815fe34a8ea56
Architecture: amd64
Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.11
Operating System: windows
GOPATH=C:\Users\sajan\go
GOROOT=C:\Go\
```

<Step. 3> Check out the command line and management APIs.

The list of available command line options for geth can be viewed by simply typing “geth” or “geth help” in the command line prompt.



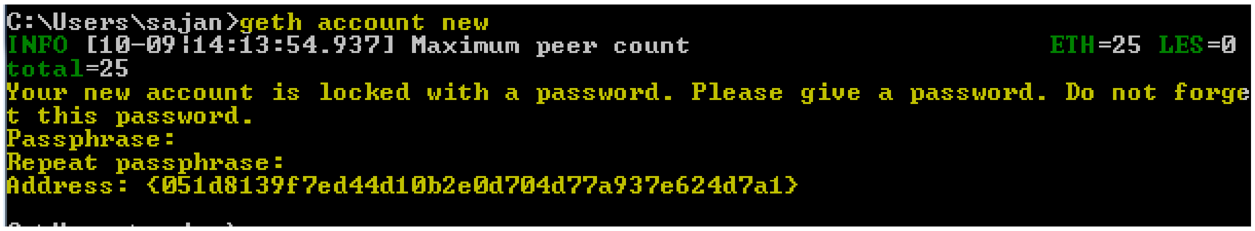
```
C:\Users\sajan>geth help
NAME:
   geth - the go-ethereum command line interface

   Copyright 2013-2018 The go-ethereum Authors

USAGE:
   geth [options] command [command options] [arguments...]

VERSION:
   1.8.16-stable-477eb093

COMMANDS:
   account            Manage accounts
   attach             Start an interactive JavaScript environment (connect to nod
e)
   bug                opens a window to report a bug on the geth repo
   console            Start an interactive JavaScript environment
   copydb             Create a local chain from a target chaindata folder
   dump               Dump a specific block from storage
   dumpconfig         Show configuration values
   export             Export blockchain into file
   export-preimages   Export the preimage database into an RLP stream
   import             Import a blockchain file
   import-preimages   Import the preimage database from an RLP stream
   init               Bootstrap and initialize a new genesis block
   js                 Execute the specified JavaScript files
   license            Display license information
   makecache          Generate ethash verification cache (for testing)
   makedag            Generate ethash mining DAG (for testing)
   monitor            Monitor and visualize node metrics
   removedb           Remove blockchain and state databases
   version            Print version numbers
   wallet             Manage Ethereum presale wallets
   help, h            Shows a list of commands or help for one command
```

The above snapshot shows “geth” command line options for managing accounts, dumping a specific block, importing and exporting private keys, initializing genesis blocks etc. Besides these command line options there are also other options that enable us to connect to the ethereum network, ethashing option, transaction pool options among others.

Let us look at how to create a new account in geth-

>> geth account new

command line can be used to create a new account, which will then prompt the user to enter corresponding password to create a new account address. eg.

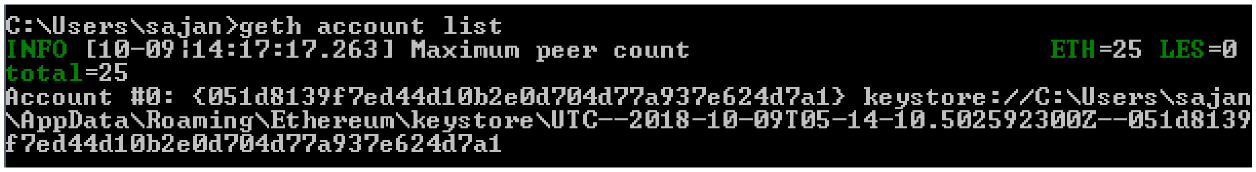


```
C:\Users\sajan>geth account new
INFO [10-09|14:13:54.937] Maximum peer count                       ETH=25 LES=0
total=25
Your new account is locked with a password. Please give a password. Do not forge
t this password.
Passphrase:
Repeat passphrase:
Address: {051d8139f7ed44d10b2e0d704d77a937e624d7a1}
```

There is no way to recover lost passwords in geth, so it is important to secure an account with a password you will remember.

You can list the recently created account using the command-

>> geth account list



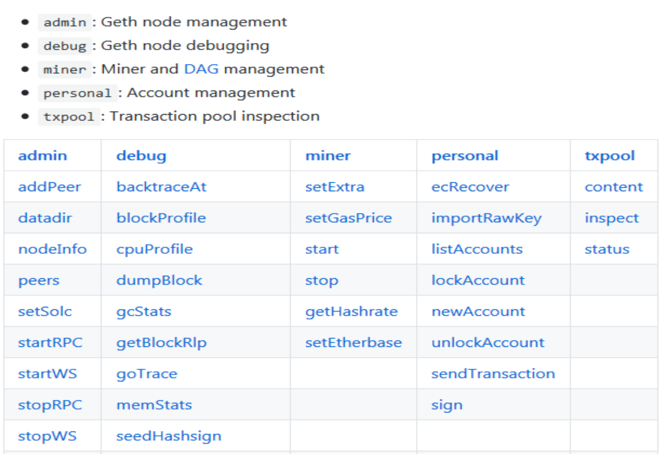
```
C:\Users\sajan>geth account list
INFO [10-09|14:17:17.263] Maximum peer count                       ETH=25 LES=0
total=25
Account #0: {051d8139f7ed44d10b2e0d704d77a937e624d7a1} keystore://C:\Users\sajan
\AppData\Roaming\Ethereum\keystore\UTC--2018-10-09T05-14-10.502592300Z--051d8139
f7ed44d10b2e0d704d77a937e624d7a1
```

To run geth on the testnet, use the option –testnet.

To use the interactive javascript runtime environment within geth, use the command-

>> geth console --testnet --syncmode=“fast”

When you run this command, it will import block headers from the testnet and once its done, you can run various management APIs such as- admin, debug, miner, txpool. The following are the management APIs available:



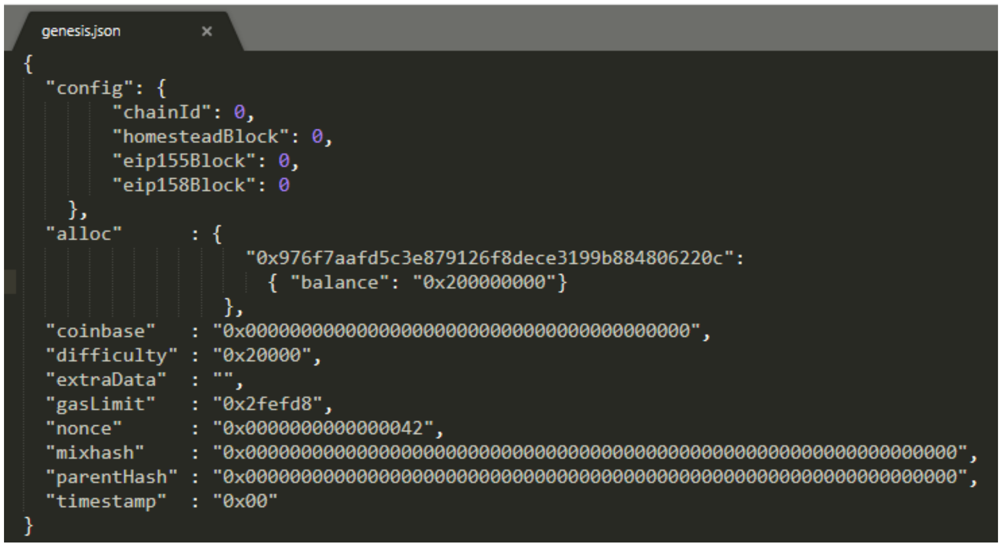
- `admin` : Geth node management
- `debug` : Geth node debugging
- `miner` : Miner and DAG management
- `personal` : Account management
- `txpool` : Transaction pool inspection

admin	debug	miner	personal	txpool
addPeer	backtraceAt	setExtra	ecRecover	content
datadir	blockProfile	setGasPrice	importRawKey	inspect
nodeInfo	cpuProfile	start	listAccounts	status
peers	dumpBlock	stop	lockAccount	
setSolc	gcStats	getHashrate	newAccount	
startRPC	getBlockRlp	setEtherbase	unlockAccount	
startWS	goTrace		sendTransaction	
stopRPC	memStats		sign	
stopWS	seedHashsign			

<Step. 4> Building Private Network

4-1) Generate a genesis file

In order to build your own private network in ethereum, where you can test your dApps you have developed, you first need to create a genesis file, defined as a JSON file. The contents of the JSON file is as follows-



genesis.json

```
{
  "config": {
        "chainId": 0,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
  "alloc"      : {
                    "0x976f7aafd5c3e879126f8dece3199b884806220c":
                      { "balance": "0x200000000"}
                },
  "coinbase"   : "0x0000000000000000000000000000000000000000",
  "difficulty" : "0x20000",
  "extraData"  : "",
  "gasLimit"   : "0x2fefd8",
  "nonce"      : "0x0000000000000042",
  "mixhash"    : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp"  : "0x00"
}
```

The alloc field will contain the account address, we had created earlier. Nested within this field is another field called balance, which will assign corresponding value of the balance to that account. Usually, it is required to save this .*JSON* file in a directory from which the geth command is running. (usually the user's home folder, else the path to .JSON file is required while initializing the private chain).

4-2) Create a data directory to use in the private network

By default in windows, the blockchain data received from a network (when connected to) is stored in the directory- ***%appdata%/Ethereum/***

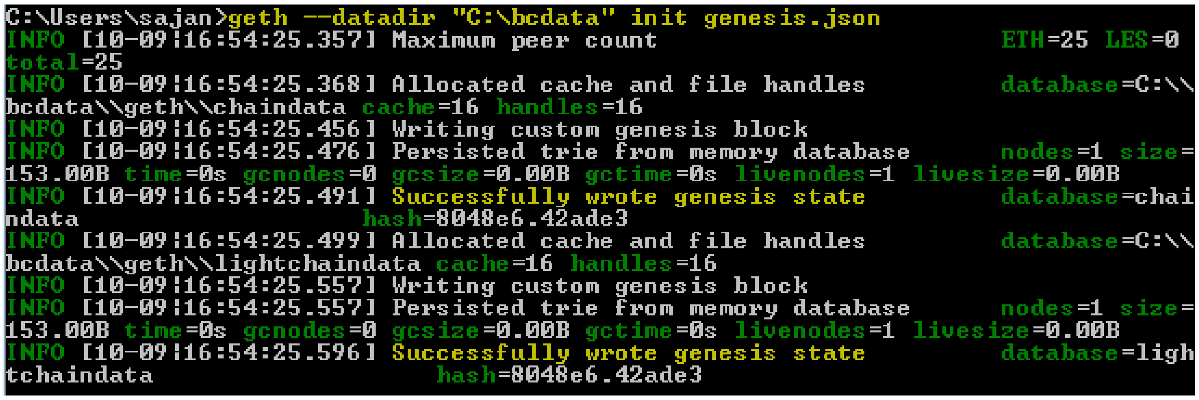
We can specify a custom directory to store blockchain data using –datadir option.

I created a new directory named 'bcdata' in the path '**C:\bcdata**'

4-3) Make genesis block by using a genesis file

To create/initialize a genesis block in the private network, we use the following command-

>> geth --datadir "Path\to\store\blockchain\data" init Path/genesisfile.json



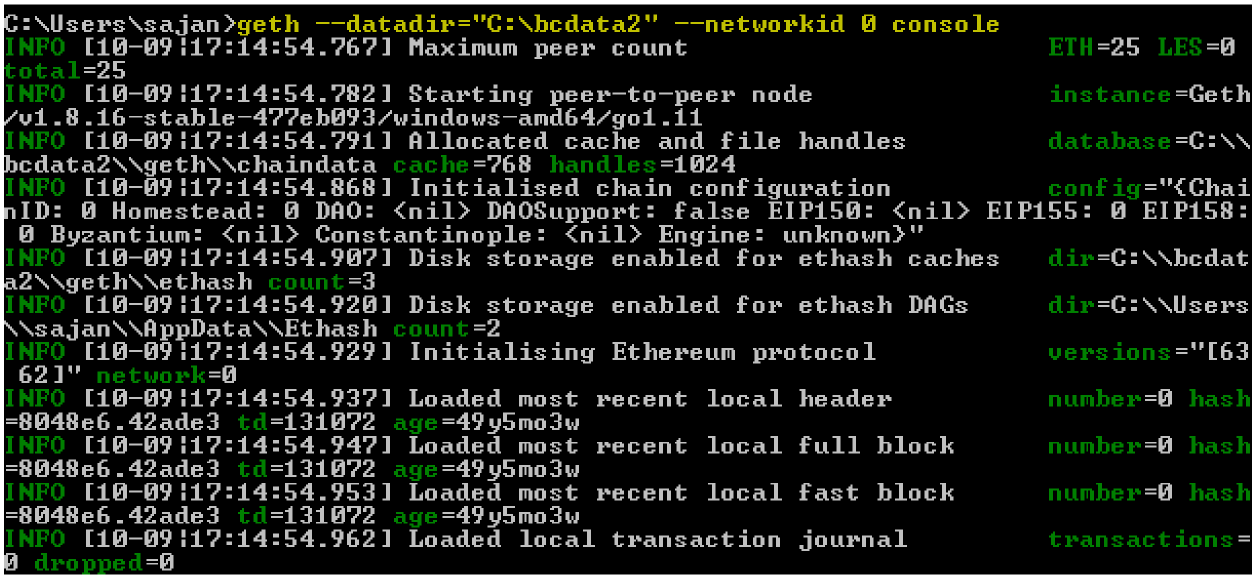
```
C:\Users\sajan>geth --datadir "C:\bcdata" init genesis.json
INFO [10-09|16:54:25.357] Maximum peer count                       ETH=25 LES=0
total=25
INFO [10-09|16:54:25.368] Allocated cache and file handles         database=C:\\
bcdata\\geth\\chaindata cache=16 handles=16
INFO [10-09|16:54:25.456] Writing custom genesis block
INFO [10-09|16:54:25.476] Persisted trie from memory database      nodes=1 size=
153.00B time=0s gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-09|16:54:25.491] Successfully wrote genesis state         database=chai
ndata                    hash=8048e6.42ade3
INFO [10-09|16:54:25.499] Allocated cache and file handles         database=C:\\
bcdata\\geth\\lightchaindata cache=16 handles=16
INFO [10-09|16:54:25.557] Writing custom genesis block
INFO [10-09|16:54:25.557] Persisted trie from memory database      nodes=1 size=
153.00B time=0s gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-09|16:54:25.596] Successfully wrote genesis state         database=ligh
tchaindata                        hash=8048e6.42ade3
```

4-4) Run geth engine with console

To start the geth console, connecting to the private network we just created, we can use the command-

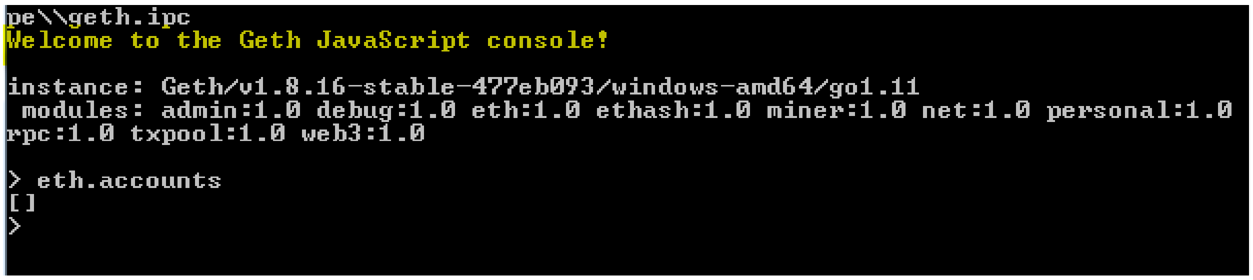
>> geth --datadir PATH_TO_STORE_BLOCK_DATA --networkid VALUE_OF_CHAINID console

In the genesis.json file which we created, we had assigned the value 0 to the field chainID, so we run the above command as-



```
C:\Users\sajan>geth --datadir="C:\bcdata2" --networkid 0 console
INFO [10-09|17:14:54.767] Maximum peer count                       ETH=25 LES=0
total=25
INFO [10-09|17:14:54.782] Starting peer-to-peer node               instance=Geth
/v1.8.16-stable-477eb093/windows-amd64/go1.11
INFO [10-09|17:14:54.791] Allocated cache and file handles         database=C:\\
bcdata2\\geth\\chaindata cache=768 handles=1024
INFO [10-09|17:14:54.868] Initialised chain configuration          config="{Chai
nID: 0 Homestead: 0 DAO: <nil> DAOSupport: false EIP150: <nil> EIP155: 0 EIP158:
 0 Byzantium: <nil> Constantinople: <nil> Engine: unknown}"
INFO [10-09|17:14:54.907] Disk storage enabled for ethash caches   dir=C:\\bcdat
a2\\geth\\ethash count=3
INFO [10-09|17:14:54.920] Disk storage enabled for ethash DAGs     dir=C:\\Users
\\sajan\\AppData\\Ethash count=2
INFO [10-09|17:14:54.929] Initialising Ethereum protocol           versions="[63
 62]" network=0
INFO [10-09|17:14:54.937] Loaded most recent local header          number=0 hash
=8048e6.42ade3 td=131072 age=49y5mo3w
INFO [10-09|17:14:54.947] Loaded most recent local full block      number=0 hash
=8048e6.42ade3 td=131072 age=49y5mo3w
INFO [10-09|17:14:54.953] Loaded most recent local fast block      number=0 hash
=8048e6.42ade3 td=131072 age=49y5mo3w
INFO [10-09|17:14:54.962] Loaded local transaction journal         transactions=
0 dropped=0
```

This will also provide access to the console with the message as shown below-



```
pe\\geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.16-stable-477eb093/windows-amd64/go1.11
 modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0
rpc:1.0 txpool:1.0 web3:1.0

> eth.accounts
[]
>
```

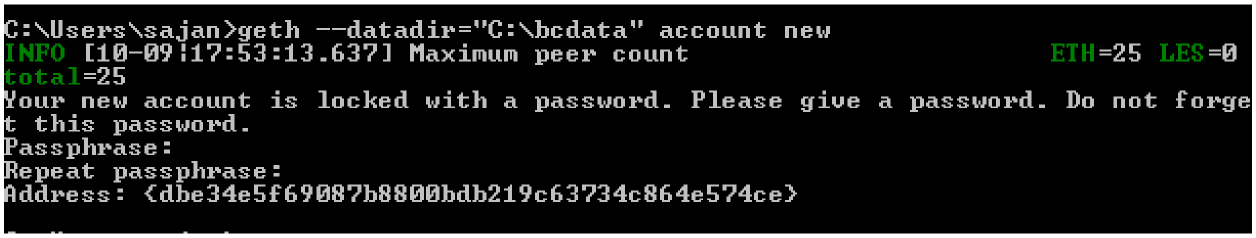
<Step. 5> Generate addresses / Mine Ether / Check the balance

5-1) Create two addresses which can be used for sender and receiver in transmission testing.

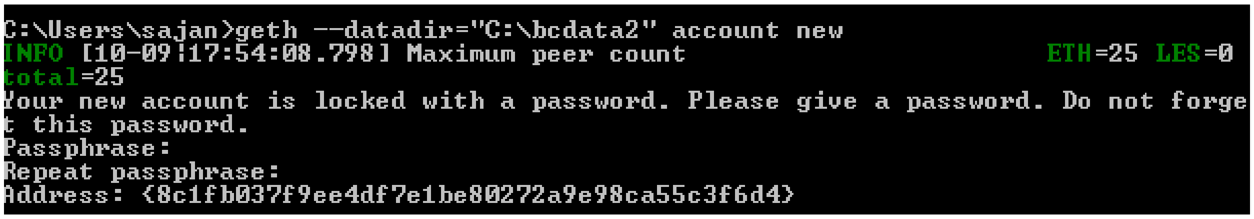
Let us create two different addresses in two different directories- ‘C:\bcdata’ and ‘C:\bcdata2’ respectively. New addresses can be created using the command-

```
>> geth account new
```

which is followed by passphrase for that account



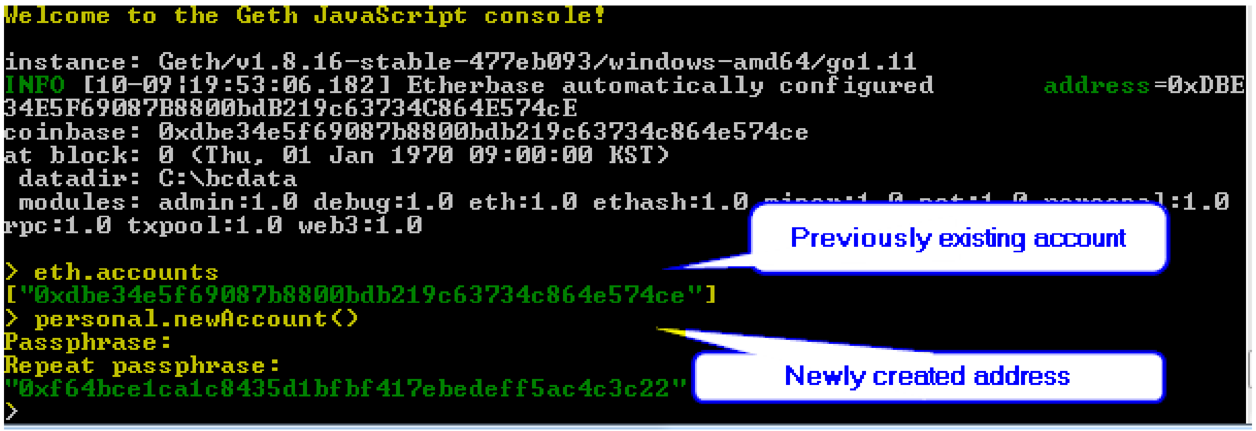
```
C:\Users\sajan>geth --datadir="C:\bcdata" account new
INFO [10-09|17:53:13.637] Maximum peer count                       ETH=25 LES=0
total=25
Your new account is locked with a password. Please give a password. Do not forge
t this password.
Passphrase:
Repeat passphrase:
Address: {dbe34e5f69087b88800bdb219c63734c864e574ce}
```



```
C:\Users\sajan>geth --datadir="C:\bcdata2" account new
INFO [10-09|17:54:08.798] Maximum peer count                       ETH=25 LES=0
total=25
Your new account is locked with a password. Please give a password. Do not forge
t this password.
Passphrase:
Repeat passphrase:
Address: {8c1fb037f9ee4df7e1be80272a9e98ca55c3f6d4}
```

We can see that two new addresses have been created and each of these addresses are being stored in a different directory. We can also create an account from within the geth console. Let us create another account for each of these directories using the geth console-

```
>> geth --datadir="C:\bcdata" --networkid 0 console

>>personal.newAccount()
```



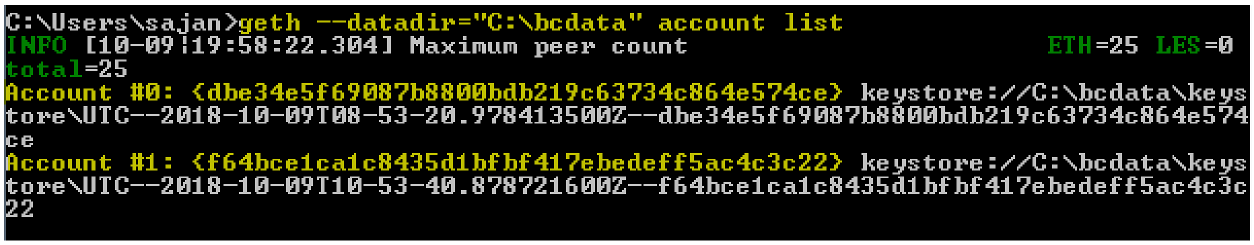
```
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.16-stable-477eb093/windows-amd64/go1.11
INFO [10-09|19:53:06.182] Etherbase automatically configured       address=0xDBE
34E5F69087B8800bdB219c63734C864E574cE
coinbase: 0xdbe34e5f69087b8800bdb219c63734c864e574ce
at block: 0 (Thu, 01 Jan 1970 09:00:00 KST)
 datadir: C:\bcdata
 modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0
rpc:1.0 txpool:1.0 web3:1.0

> eth.accounts
["0xdbe34e5f69087b8800bdb219c63734c864e574ce"]
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0xf64bce1ca1c8435d1bfbf417ebedeff5ac4c3c22"
>
```

5-2) Look up your account list.

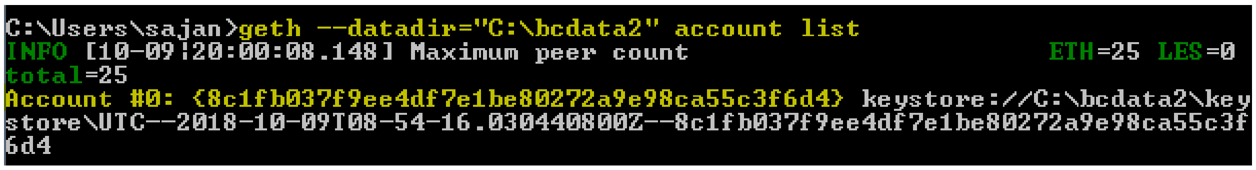
We can query the list of accounts using the command line-

>> geth --datadir=”PATH_TO_DIR” account list

In the case that the option --datadir is not provided, the list of account is retrieved from the default location where the Ethereum data is stored.



```
C:\Users\sajan>geth --datadir="C:\bcdata" account list
INFO [10-09|19:58:22.304] Maximum peer count                       ETH=25 LES=0
total=25
Account #0: {dbe34e5f69087b8800bdb219c63734c864e574ce} keystore://C:\bcdata\keys
tore\UTC--2018-10-09T08-53-20.978413500Z--dbe34e5f69087b8800bdb219c63734c864e574
ce
Account #1: {f64bce1ca1c8435d1bfbf417ebedeff5ac4c3c22} keystore://C:\bcdata\keys
tore\UTC--2018-10-09T10-53-40.878721600Z--f64bce1ca1c8435d1bfbf417ebedeff5ac4c3c
22
```

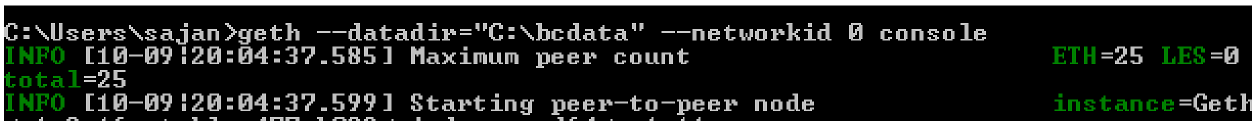


```
C:\Users\sajan>geth --datadir="C:\bcdata2" account list
INFO [10-09|20:00:08.148] Maximum peer count                       ETH=25 LES=0
total=25
Account #0: {8c1fb037f9ee4df7e1be80272a9e98ca55c3f6d4} keystore://C:\bcdata2\key
store\UTC--2018-10-09T08-54-16.030440800Z--8c1fb037f9ee4df7e1be80272a9e98ca55c3f
6d4
```

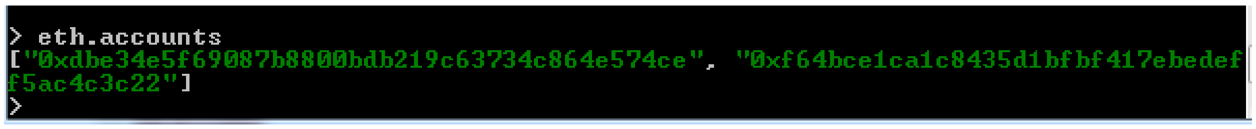
As we can see from the screenshot above, each of the account created in a different folder acts as a different node in the network and will list only those. It is also possible to view the list of account addresses using the geth console-

>> geth --datadir=“C:\bcdata” --networkid 0 console

>> eth.accounts



```
C:\Users\sajan>geth --datadir="C:\bcdata" --networkid 0 console
INFO [10-09|20:04:37.585] Maximum peer count                       ETH=25 LES=0
total=25
INFO [10-09|20:04:37.599] Starting peer-to-peer node               instance=Geth
```



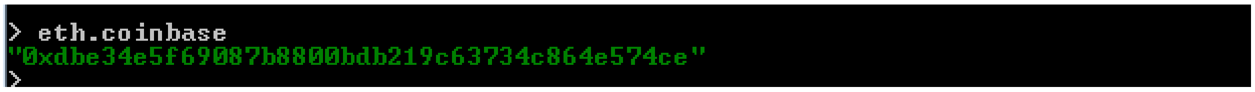
```
> eth.accounts
["0xdbe34e5f69087b8800bdb219c63734c864e574ce", "0xf64bce1ca1c8435d1bfbf417ebedef
f5ac4c3c22"]
>
```

5-3) Set the etherbase address. Before you transfer Ether, you must mine Ether to verify the transaction. So, you have to specify the etherbase to be rewarded after mining.

Etherbase address is the address to which mining profits are received by the miner. By default, the first address created by the node/miner will be set to the etherbase address. We can check the pre-defined etherbase address for a node via the geth console as-

>> geth --datadir="C:\bcdata" --networkid 0 console

>> eth.coinbase

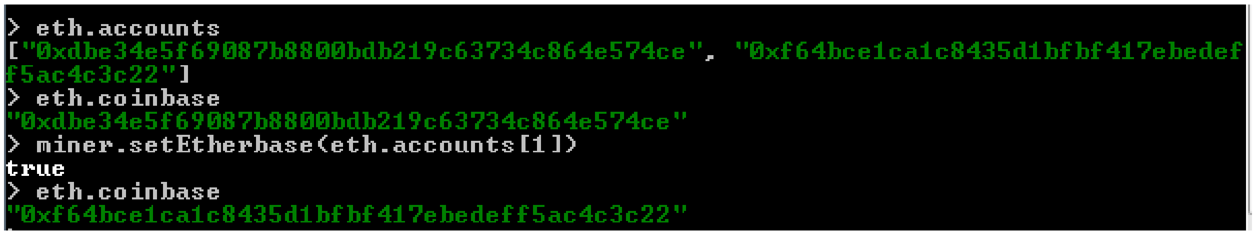


```
> eth.coinbase
"0xdbe34e5f69087b8800bdb219c63734c864e574ce"
>
```

As the above address was the first address created for the node corresponding to the folder "C:\bcdata", by default this address has been assigned the etherbase address. If we wish to change the etherbase address, to another address in the node, we can use the command

>> miner.setEtherbase(NEW_ACCOUNT_ADDRESS)

If the above command is successful, it will return true.

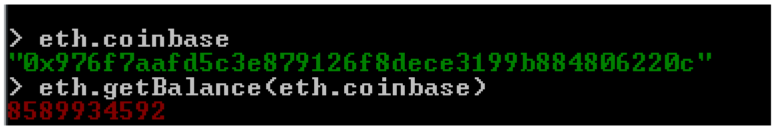


```
> eth.accounts
["0xdbe34e5f69087b8800bdb219c63734c864e574ce", "0xf64bce1ca1c8435d1bfbf417ebedef
f5ac4c3c22"]
> eth.coinbase
"0xdbe34e5f69087b8800bdb219c63734c864e574ce"
> miner.setEtherbase(eth.accounts[1])
true
> eth.coinbase
"0xf64bce1ca1c8435d1bfbf417ebedeff5ac4c3c22"
```

5-4) Start mining.

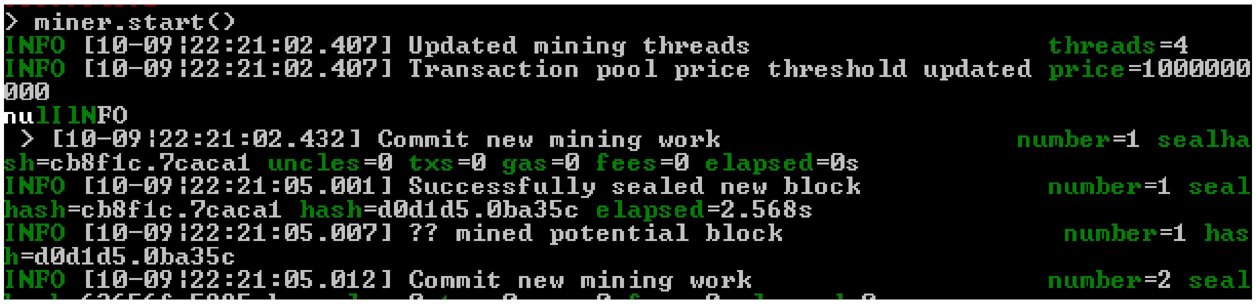
When we first initialized the genesis file, the alloc field contained an address and we assigned some balance into it. Check the current balance for the coinbase address before starting mining-

>> eth.getBalance(eth.coinbase)



```
> eth.coinbase
"0x976f7aafd5c3e879126f8dece3199b884806220c"
> eth.getBalance(eth.coinbase)
8589934592
```

>> miner.start()



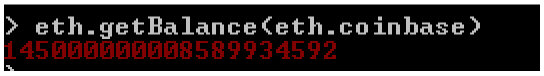
```
> miner.start()
INFO [10-09|22:21:02.407] Updated mining threads                   threads=4
INFO [10-09|22:21:02.407] Transaction pool price threshold updated price=1000000
000
nullINFO
 > [10-09|22:21:02.432] Commit new mining work                   number=1 sealha
sh=cb8f1c.7caca1 uncles=0 txs=0 gas=0 fees=0 elapsed=0s
INFO [10-09|22:21:05.001] Successfully sealed new block            number=1 seal
hash=cb8f1c.7caca1 hash=d0d1d5.0ba35c elapsed=2.568s
INFO [10-09|22:21:05.007] ?? mined potential block                  number=1 has
h=d0d1d5.0ba35c
INFO [10-09|22:21:05.012] Commit new mining work                   number=2 seal
```

This command will the continue with the creation of DAG.

5-5) After mining, check the balance of all accounts.

The balance contained by an address can be checked using the command-

>> eth.getBalance(ACCOUNT_ADDRESS)



```
> eth.getBalance(eth.coinbase)
145000000008589934592
>
```

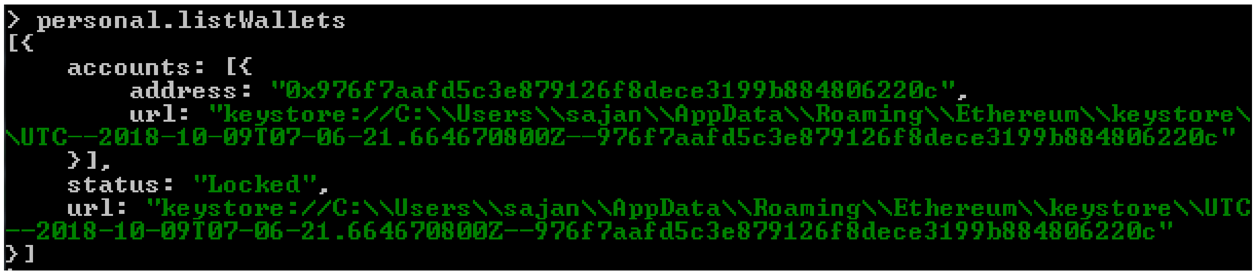
<Step. 6> Unlock the account / Send the transaction.

6-1) Check the status of sender's account

To check the status of one's account, use the command line-

>> personal.listWallets

The JSON object returned by this command will contain a status field that will denote the status of the account



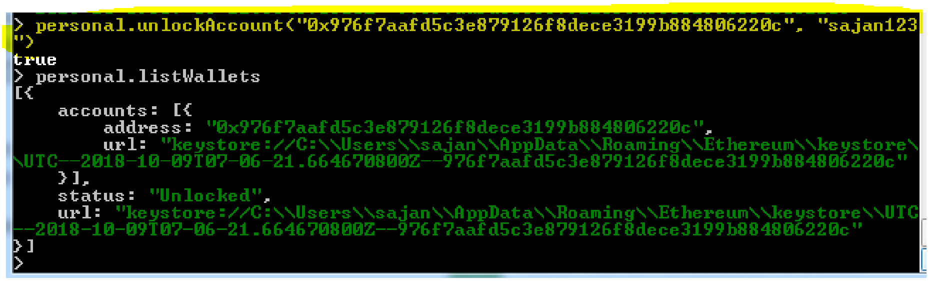
```
> personal.listWallets
[{
    accounts: [{
        address: "0x976f7aafd5c3e879126f8dece3199b884806220c",
        url: "keystore://C:\\Users\\sajan\\AppData\\Roaming\\Ethereum\\keystore\
\UTC--2018-10-09T07-06-21.664670800Z--976f7aafd5c3e879126f8dece3199b884806220c"
    }],
    status: "Locked",
    url: "keystore://C:\\Users\\sajan\\AppData\\Roaming\\Ethereum\\keystore\\UTC
--2018-10-09T07-06-21.664670800Z--976f7aafd5c3e879126f8dece3199b884806220c"
}]
```

6-2) Before sending Ether, you have to unlock the sender's account.

In order to unlock an account, use the command

>> personal.unlockAccount(ACCOUNT_ADDRESS, PASSWORD)

This command will return true if successful and and the status of the account will be unlocked for the current session. Check status to confirm.

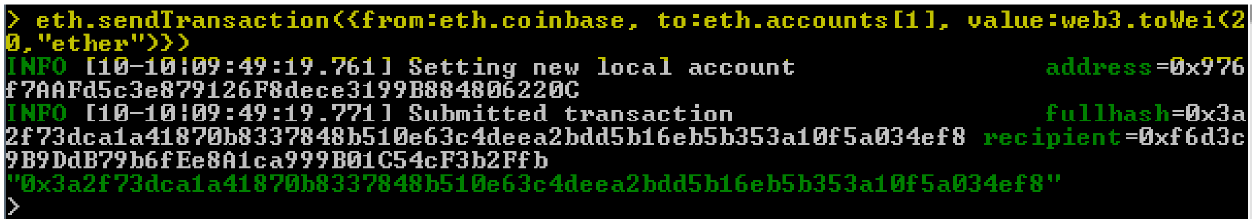


```
> personal.unlockAccount("0x976f7aafd5c3e879126f8dece3199b884806220c", "sajan123
")
true
> personal.listWallets
[{
    accounts: [{
        address: "0x976f7aafd5c3e879126f8dece3199b884806220c",
        url: "keystore://C:\\Users\\sajan\\AppData\\Roaming\\Ethereum\\keystore\
\UTC--2018-10-09T07-06-21.664670800Z--976f7aafd5c3e879126f8dece3199b884806220c"
    }],
    status: "Unlocked",
    url: "keystore://C:\\Users\\sajan\\AppData\\Roaming\\Ethereum\\keystore\\UTC
--2018-10-09T07-06-21.664670800Z--976f7aafd5c3e879126f8dece3199b884806220c"
}]
>
```

6-3) Send 20 Ethers to the receiver's address

Use eth.sendTransaction() to send ether from one address to another in the format below-

>> eth.sendTransaction({from:SENDER_ADDR, to:RECEIVER_ADDR, value: web3.toWei(AMOUNT,”ether”)})



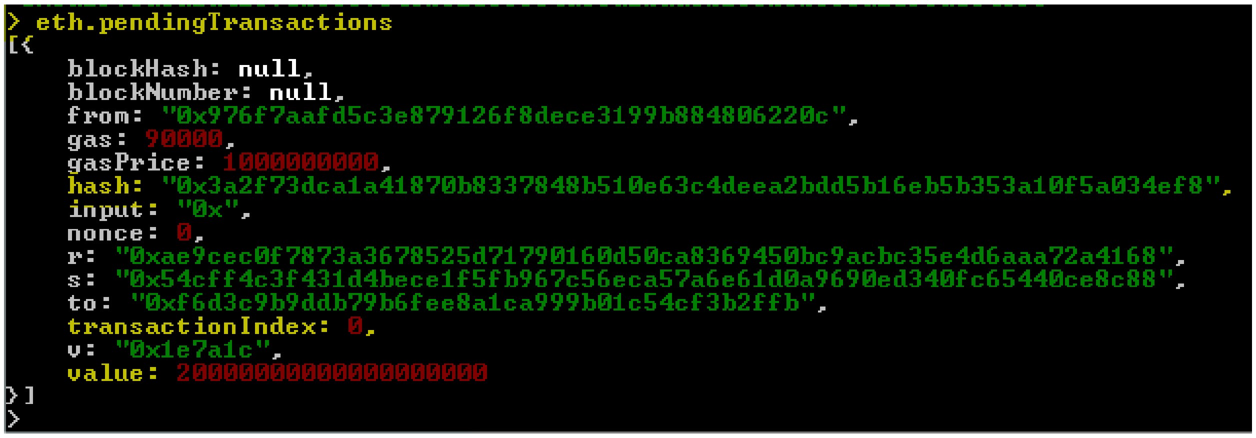
```
> eth.sendTransaction({from:eth.coinbase, to:eth.accounts[1], value:web3.toWei(2
0,"ether")})
INFO [10-10|09:49:19.761] Setting new local account                address=0x976
f7AAFd5c3e879126F8dece3199B884806220C
INFO [10-10|09:49:19.771] Submitted transaction                    fullhash=0x3a
2f73dca1a41870b8337848b510e63c4deea2bdd5b16eb5b353a10f5a034ef8 recipient=0xf6d3c
9B9DdB79b6fEe8A1ca999B01C54cF3b2Ffb
"0x3a2f73dca1a41870b8337848b510e63c4deea2bdd5b16eb5b353a10f5a034ef8"
>
```

If the command executes successfully, it will return a corresponding transaction hash

6-4) Check the pending transactions.

The transaction that are waiting to be written on the blockchain can be queried using the command-

>> eth.pendingTransactions



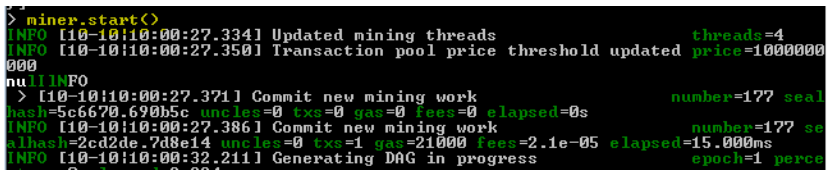
```
> eth.pendingTransactions
[{
    blockHash: null,
    blockNumber: null,
    from: "0x976f7aafd5c3e879126f8dece3199b884806220c",
    gas: 90000,
    gasPrice: 1000000000,
    hash: "0x3a2f73dca1a41870b8337848b510e63c4deea2bdd5b16eb5b353a10f5a034ef8",
    input: "0x",
    nonce: 0,
    r: "0xae9cec0f7873a3678525d71790160d50ca8369450bc9acbc35e4d6aaa72a4168",
    s: "0x54cff4c3f431d4bece1f5fb967c56eca57a6e61d0a9690ed340fc65440ce8c88",
    to: "0xf6d3c9b9ddb79b6fee8a1ca999b01c54cf3b2ffb",
    transactionIndex: 0,
    v: "0x1e7a1c",
    value: 20000000000000000000
}]
>
```

As seen from the above screenshot, the eth.pendingTransactions command will return a JSON object containing information such as from address, to address, value (in Wei) sent, transaction hash, etc.

6-5) Check the pending transactions again after mining.

The above pending transaction is waiting to be written into the blockchain. Once mining starts, the above transaction will be written in the blockchain. We start mining using the command-

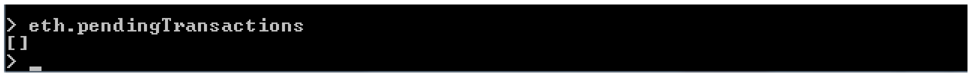
>> miner.start()



```
> miner.start()
INFO [10-10|10:00:27.334] Updated mining threads                   threads=4
INFO [10-10|10:00:27.350] Transaction pool price threshold updated price=1000000
000
nullINFO
 > [10-10|10:00:27.371] Commit new mining work                   number=177 seal
hash=5c6670.690b5c uncles=0 txs=0 gas=0 fees=0 elapsed=0s
INFO [10-10|10:00:27.386] Commit new mining work                   number=177 se
alhash=2cd2de.7d8e14 uncles=0 txs=1 gas=21000 fees=2.1e-05 elapsed=15.000ms
INFO [10-10|10:00:32.211] Generating DAG in progress               epoch=1 perce
```

And then check the mining transaction once again using the command-

>> eth.pendingTransactions

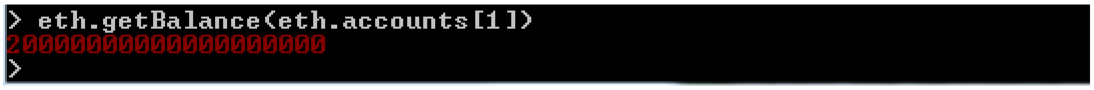


```
> eth.pendingTransactions
[]
> _
```

6-6) Look up the balance of receiver's account to see if sender successfully transfer 20 Ethers.

We can confirm the balance of the recipient account using the command-

>> eth.getBalance(ACCOUNT_ADDR)



```
> eth.getBalance(eth.accounts[1])
20000000000000000000
>
```

<Step. 7> Search for blocks

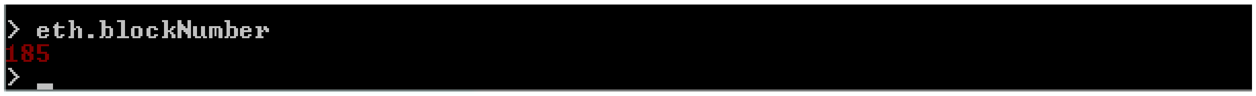
Try to search for blocks contained to the private chain

7-1) Search the total number of blocks connected to the main chain.

We can search for the latest block number in the main chain using the command

>> eth.blockNumber

This will return an integer which is the highest number of block connected in the chain.

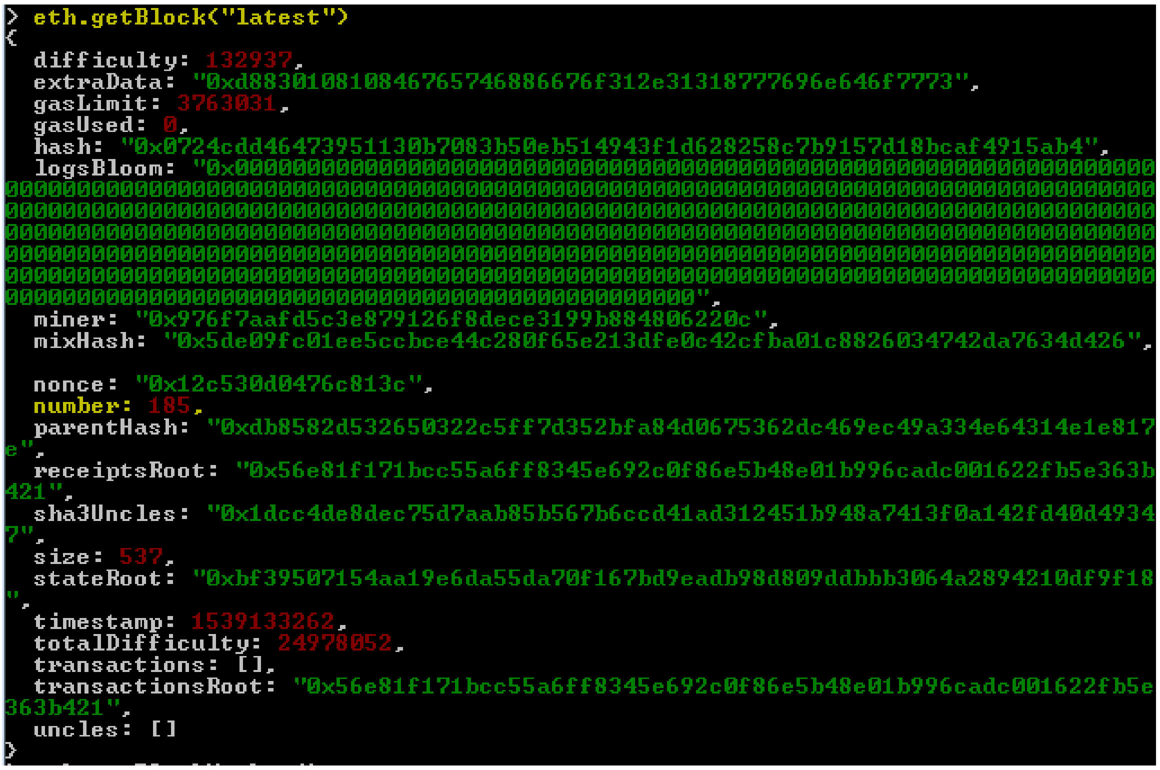


```
> eth.blockNumber
185
> _
```

Another way to query information (and thus blocknumber) of the latest block is to use the eth.getBlock command with "latest" parameter

>> eth.getBlock("latest")

This will return a JSON object containing information about the latest block added to the blockchain. The filed "number" will give the value of the block number for this latest block

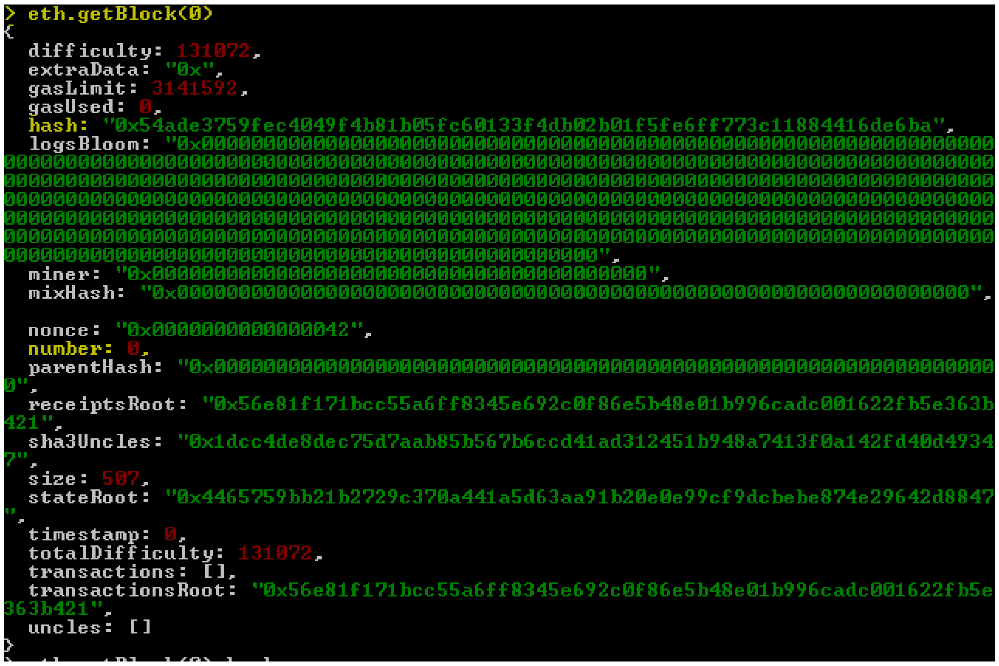


```
> eth.getBlock("latest")
{
  difficulty: 132937,
  extraData: "0xd883010810846765746886676f312e31318777696e646f7773",
  gasLimit: 3763031,
  gasUsed: 0,
  hash: "0x0724cdd46473951130b7083b50eb514943f1d628258c7b9157d18bcaf4915ab4",
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000",
  miner: "0x976f7aafd5c3e879126f8dece3199b884806220c",
  mixHash: "0x5de09fc01ee5ccbce44c280f65e213dfe0c42cfba01c8826034742da7634d426",

  nonce: "0x12c530d0476c813c",
  number: 185,
  parentHash: "0xdb8582d532650322c5ff7d352bfa84d0675362dc469ec49a334e64314e1e817
e",
  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b
421",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d4934
7",
  size: 537,
  stateRoot: "0xbf39507154aa19e6da55da70f167bd9eadb98d809ddbbb3064a2894210df9f18
",
  timestamp: 1539133262,
  totalDifficulty: 24978052,
  transactions: [],
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e
363b421",
  uncles: []
}
```

7-2) Search for the block of Height 0

Use the eth.getBlock command to get the block information, 0 is passed as an argument to the command

>>eth.getBlock(BLOCK_NUM_OR_BLOCK_HASH)

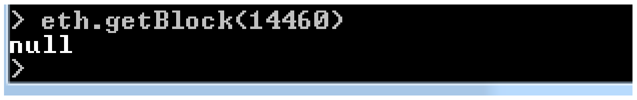


```
> eth.getBlock(0)
{
  difficulty: 131072,
  extraData: "0x",
  gasLimit: 3141592,
  gasUsed: 0,
  hash: "0x54ade3759fec4049f4b81b05fc60133f4db02b01f5fe6ff773c11884416de6ba",
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000",
  miner: "0x0000000000000000000000000000000000000000",
  mixHash: "0x0000000000000000000000000000000000000000000000000000000000000000",

  nonce: "0x0000000000000042",
  number: 0,
  parentHash: "0x000000000000000000000000000000000000000000000000000000000000000
0",
  receiptsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b
421",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d4934
7",
  size: 507,
  stateRoot: "0x4465759bb21b2729c370a441a5d63aa91b20e0e99cf9dcbebe874e29642d8847
",
  timestamp: 0,
  totalDifficulty: 131072,
  transactions: [],
  transactionsRoot: "0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e
363b421",
  uncles: []
}
```

7-3) Search for the block of Height 14460

Use the eth.getBlock command to get the block information, 0 is passed as an argument to the command

>>eth.getBlock(BLOCK_NUM_OR_BLOCK_HASH)



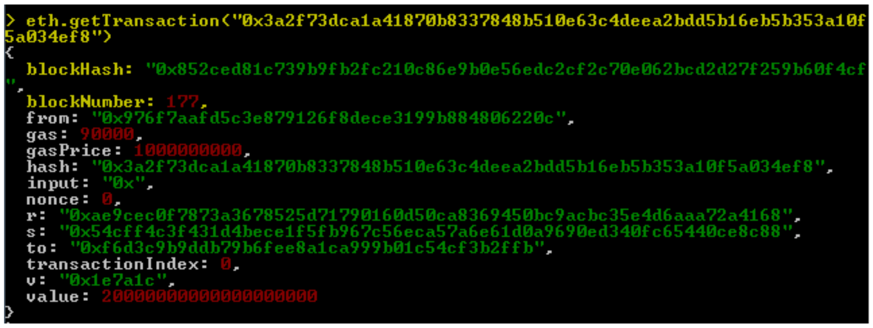
```
> eth.getBlock(14460)
null
>
```

This is so because, there is block of height 14460 in our chain. The highest block number in our chain is 185, so querying for the block of height 14460 returns null.

7-4) The block including the transaction which is used in transferring your Ether.

To find the transaction information, use the eth.getTransaction command passing the corresponding transaction hash as parameter.

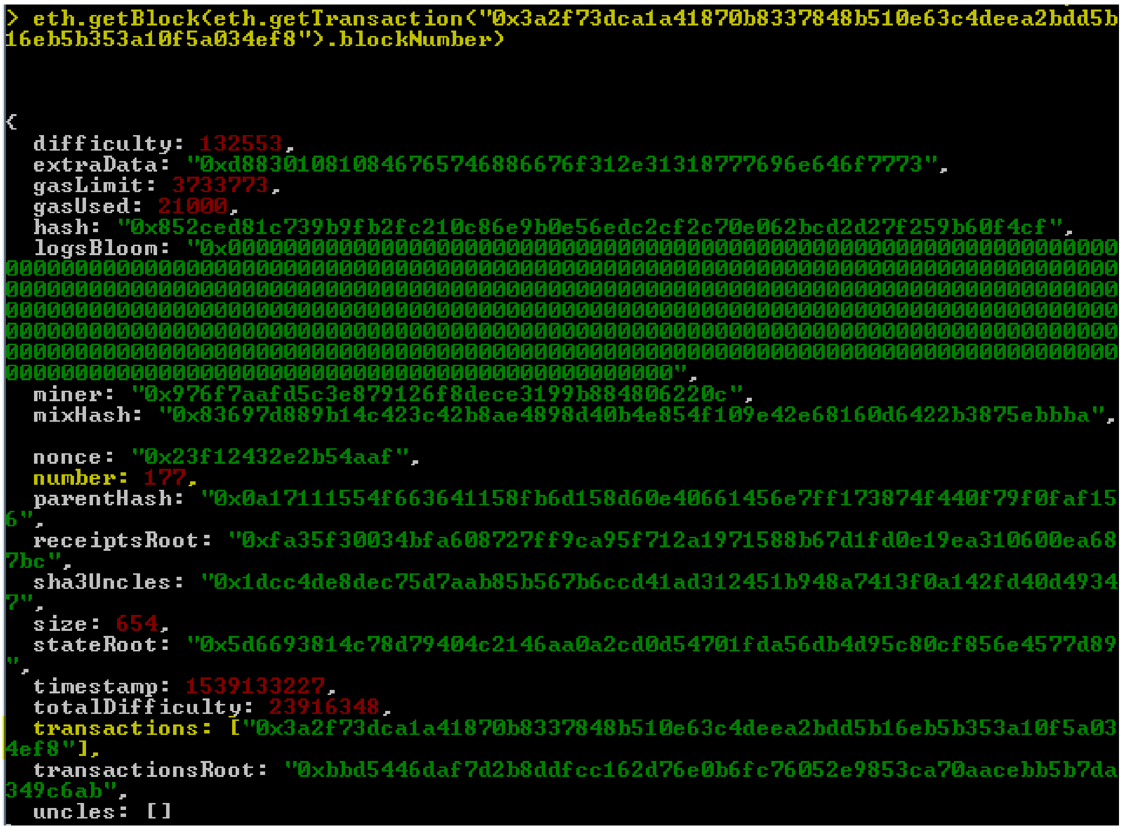
>> eth.getTransaction(TXN_HASH)



```
> eth.getTransaction("0x3a2f73dca1a41870b8337848b510e63c4deea2bdd5b16eb5b353a10f
5a034ef8")
{
  blockHash: "0x852ced81c739b9fb2fc210c86e9b0e56edc2cf2c70e062bcd2d27f259b60f4cf
",
  blockNumber: 177,
  from: "0x976f7aafd5c3e879126f8dece3199b884806220c",
  gas: 90000,
  gasPrice: 1000000000,
  hash: "0x3a2f73dca1a41870b8337848b510e63c4deea2bdd5b16eb5b353a10f5a034ef8",
  input: "0x",
  nonce: 0,
  r: "0xae9cec0f7873a3678525d71790160d50ca8369450bc9acbc35e4d6aaa72a4168",
  s: "0x54cff4c3f431d4bece1f5fb967c56eca57a6e61d0a9690ed340fc65440ce8c88",
  to: "0xf6d3c9b9ddb79b6fee8a1ca999b01c54cf3b2ffb",
  transactionIndex: 0,
  v: "0x1e7a1c",
  value: 2000000000000000000000
}
```

This will return the basic transaction information about the corresponding transaction. If we wish to go one step ahead and find the block information containing this transaction, then we can nest the information from above command into getBlock command as-

>>eth.getBlock(eth.getTransaction(“0x3a2f73dca1a41870b8337848b510e63c4deea2bdd5b16eb5b353a10f5a034ef8”).blockNumber)



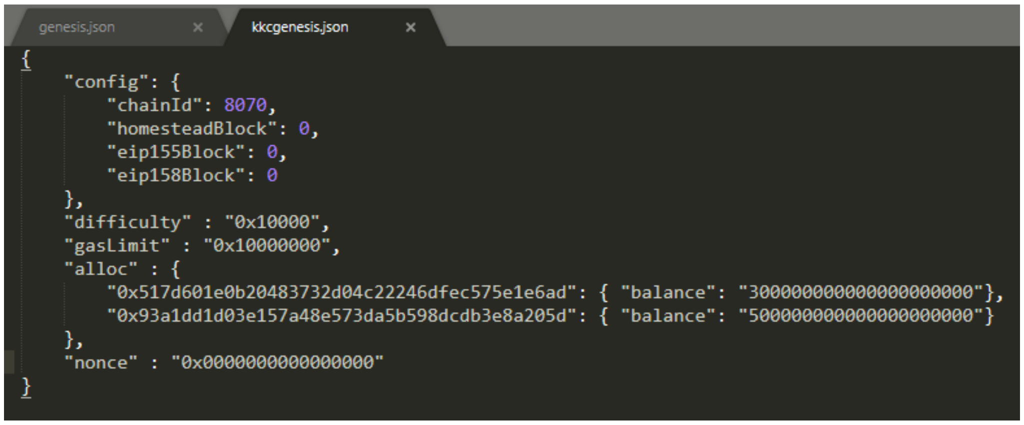
```
> eth.getBlock(eth.getTransaction("0x3a2f73dca1a41870b8337848b510e63c4deea2bdd5b
16eb5b353a10f5a034ef8").blockNumber)


{
  difficulty: 132553,
  extraData: "0xd883010810846765746886676f312e31318777696e646f7773",
  gasLimit: 3733773,
  gasUsed: 21000,
  hash: "0x852ced81c739b9fb2fc210c86e9b0e56edc2cf2c70e062bcd2d27f259b60f4cf",
  logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000",
  miner: "0x976f7aafd5c3e879126f8dece3199b884806220c",
  mixHash: "0x83697d889b14c423c42b8ae4898d40b4e854f109e42e68160d6422b3875ebbba",

  nonce: "0x23f12432e2b54aaf",
  number: 177,
  parentHash: "0x0a17111554f663641158fb6d158d60e40661456e7ff173874f440f79f0faf15
6",
  receiptsRoot: "0xfa35f30034bfa608727ff9ca95f712a1971588b67d1fd0e19ea310600ea68
7bc",
  sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d4934
7",
  size: 654,
  stateRoot: "0x5d6693814c78d79404c2146aa0a2cd0d54701fda56db4d95c80cf856e4577d89
",
  timestamp: 1539133227,
  totalDifficulty: 23916348,
  transactions: ["0x3a2f73dca1a41870b8337848b510e63c4deea2bdd5b16eb5b353a10f5a03
4ef8"],
  transactionsRoot: "0xbbd5446daf7d2b8ddfcc162d76e0b6fc76052e9853ca70aacebb5b7da
349c6ab",
  uncles: []
```

As we can see from the screenshot above, the block number 177 contains the only transaction so far we have executed and other information about this block.

<Step. 8> Join other's private network

8-1) Connect the node from your PC

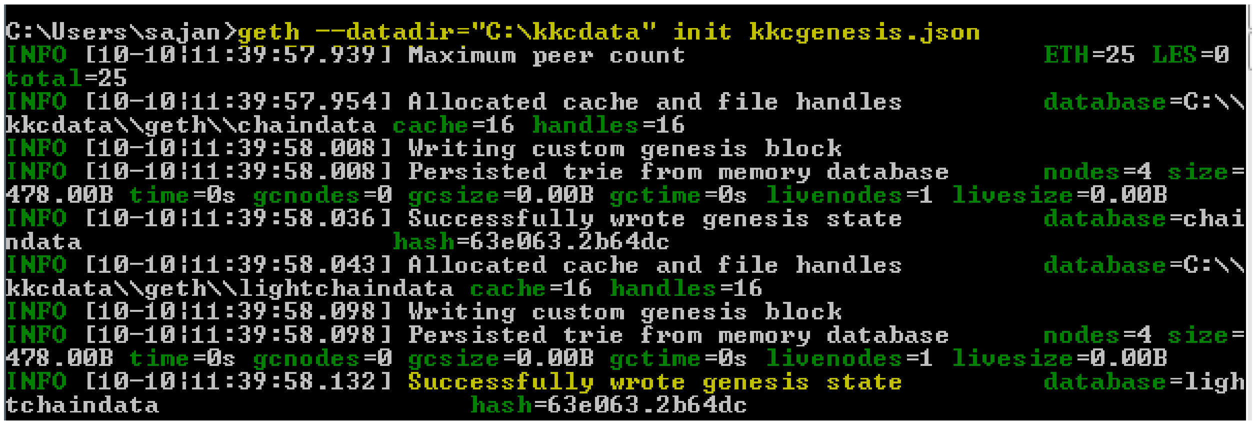
The given genesis file was saved as "kkcgenesis.json"



```
{
    "config": {
        "chainId": 8070,
        "homesteadBlock": 0,
        "eip155Block": 0,
        "eip158Block": 0
    },
    "difficulty" : "0x10000",
    "gasLimit" : "0x10000000",
    "alloc" : {
        "0x517d601e0b20483732d04c22246dfec575e1e6ad": { "balance": "300000000000000000000"},
        "0x93a1dd1d03e157a48e573da5b598dcdb3e8a205d": { "balance": "500000000000000000000"}
    },
    "nonce" : "0x0000000000000000"
}
```

Next we would like to initialize this genesis file and store the corresponding data in the folder "C:\kkcdata"

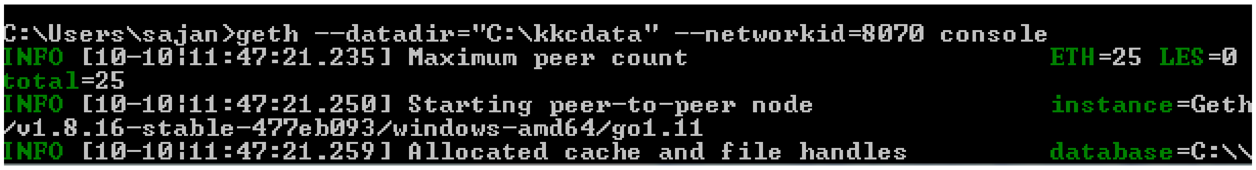
>> geth –datadir="C:\kkcdata" init kkcgenesis.json



```
C:\Users\sajan>geth --datadir="C:\kkcdata" init kkcgenesis.json
INFO [10-10|11:39:57.939] Maximum peer count                       ETH=25 LES=0
total=25
INFO [10-10|11:39:57.954] Allocated cache and file handles         database=C:\\
kkcdata\\geth\\chaindata cache=16 handles=16
INFO [10-10|11:39:58.008] Writing custom genesis block
INFO [10-10|11:39:58.008] Persisted trie from memory database      nodes=4 size=
478.00B time=0s gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-10|11:39:58.036] Successfully wrote genesis state         database=chai
ndata                    hash=63e063.2b64dc
INFO [10-10|11:39:58.043] Allocated cache and file handles         database=C:\\
kkcdata\\geth\\lightchaindata cache=16 handles=16
INFO [10-10|11:39:58.098] Writing custom genesis block
INFO [10-10|11:39:58.098] Persisted trie from memory database      nodes=4 size=
478.00B time=0s gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [10-10|11:39:58.132] Successfully wrote genesis state         database=ligh
tchaindata                    hash=63e063.2b64dc
```

Run the console with the network id as the chain id in the given genesis file

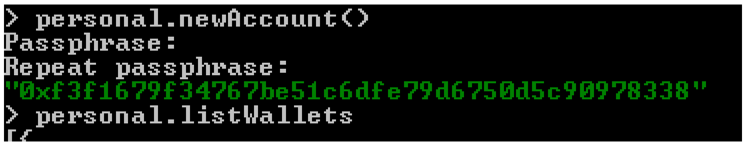
>> geth --datadir="C:\kkcdata" --networkid=8070 console



```
C:\Users\sajan>geth --datadir="C:\kkcdata" --networkid=8070 console
INFO [10-10|11:47:21.235] Maximum peer count                       ETH=25 LES=0
total=25
INFO [10-10|11:47:21.250] Starting peer-to-peer node               instance=Geth
/v1.8.16-stable-477eb093/windows-amd64/go1.11
INFO [10-10|11:47:21.259] Allocated cache and file handles         database=C:\\
```

Create a new account address

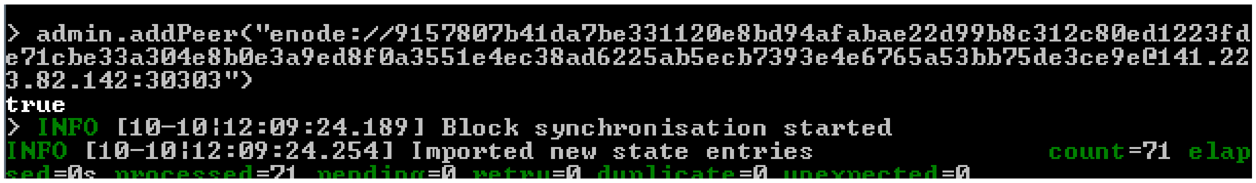
>> personal.newAccount()



```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0xf3f1679f34767be51c6dfe79d6750d5c90978338"
> personal.listWallets
[{
```

Use the admin.addPeer command to add the enode-

>> admin.addPeer("enode://ENODE_ADDR@IP_ADDR:PORT")



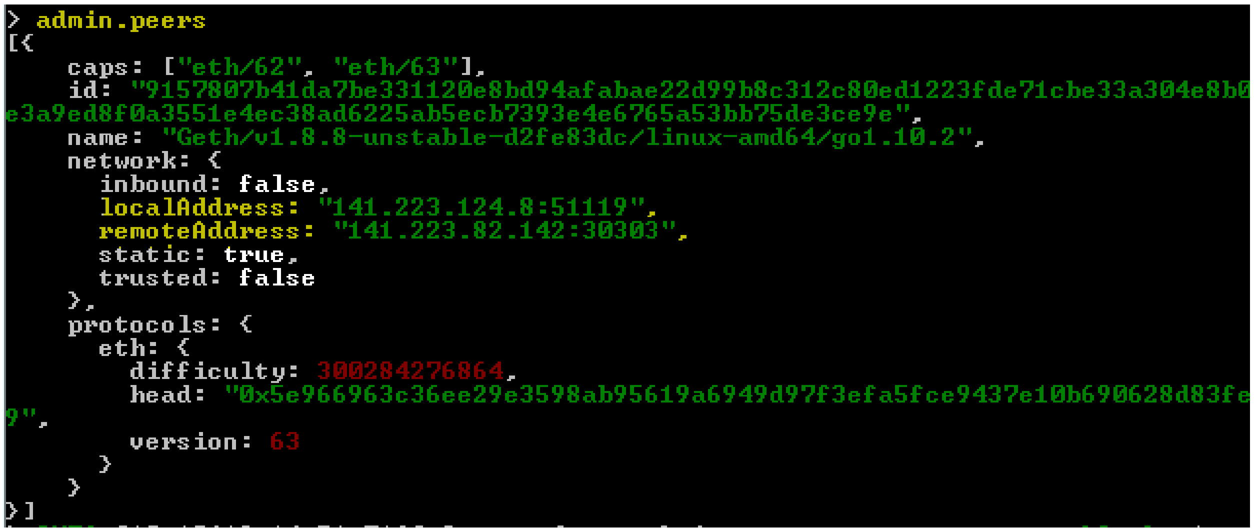
```
> admin.addPeer("enode://9157807b41da7be331120e8bd94afabae22d99b8c312c80ed1223fd
e71cbe33a304e8b0e3a9ed8f0a3551e4ec38ad6225ab5ecb7393e4e6765a53bb75de3ce9e@141.22
3.82.142:30303")
true
> INFO [10-10|12:09:24.189] Block synchronisation started
INFO [10-10|12:09:24.254] Imported new state entries               count=71 elap
sed=0s processed=71 pending=0 retry=0 duplicate=0 unexpected=0
```

This command will return true if the enode was successfully added.

8-2) Print out the information of connected peers

The information about connected peers can be retrieved using the command-

>> admin.peers

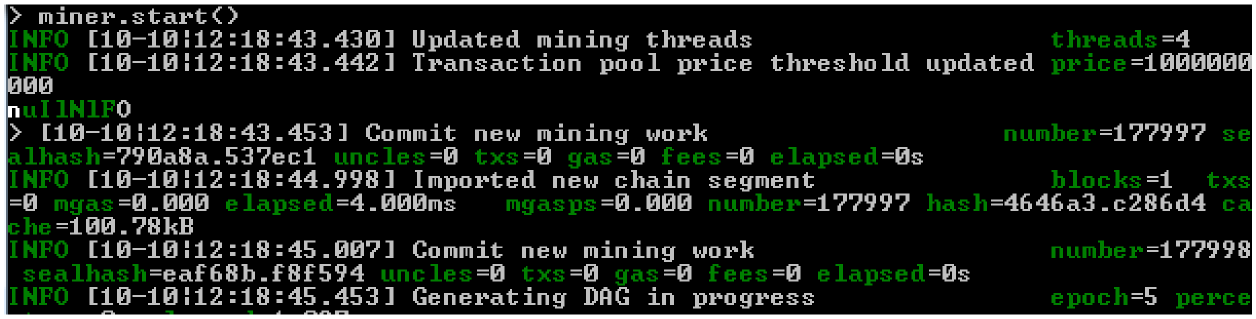


```
> admin.peers
[{
    caps: ["eth/62", "eth/63"],
    id: "9157807b41da7be331120e8bd94afabae22d99b8c312c80ed1223fde71cbe33a304e8b0
e3a9ed8f0a3551e4ec38ad6225ab5ecb7393e4e6765a53bb75de3ce9e",
    name: "Geth/v1.8.8-unstable-d2fe83dc/linux-amd64/go1.10.2",
    network: {
      inbound: false,
      localAddress: "141.223.124.8:51119",
      remoteAddress: "141.223.82.142:30303",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        difficulty: 300284276864,
        head: "0x5e966963c36ee29e3598ab95619a6949d97f3efa5fce9437e10b690628d83fe
9",
        version: 63
      }
    }
}]
```

8-3) Try to mine new block

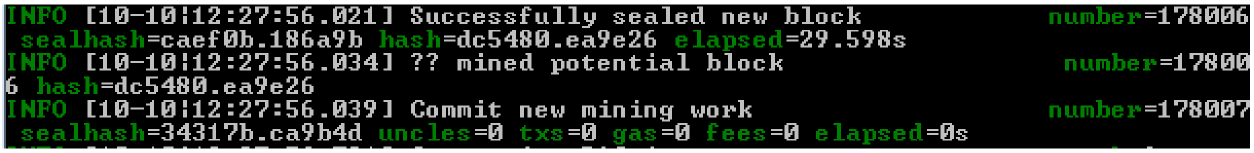
To start mining, use the command-

>> miner.start()



```
> miner.start()
INFO [10-10|12:18:43.430] Updated mining threads                   threads=4
INFO [10-10|12:18:43.442] Transaction pool price threshold updated price=1000000
000
nuIlNlFO
> [10-10|12:18:43.453] Commit new mining work                   number=177997 se
alhash=790a8a.537ec1 uncles=0 txs=0 gas=0 fees=0 elapsed=0s
INFO [10-10|12:18:44.998] Imported new chain segment               blocks=1  txs
=0 mgas=0.000 elapsed=4.000ms   mgasps=0.000 number=177997 hash=4646a3.c286d4 ca
che=100.78kB
INFO [10-10|12:18:45.007] Commit new mining work                   number=177998
 sealhash=eaf68b.f8f594 uncles=0 txs=0 gas=0 fees=0 elapsed=0s
INFO [10-10|12:18:45.453] Generating DAG in progress               epoch=5 perce
```

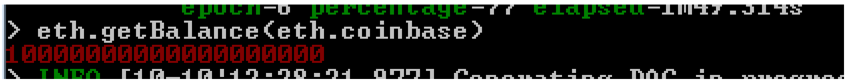
After a while, PoW solution to blocks were found-



```
INFO [10-10|12:27:56.021] Successfully sealed new block            number=178006
 sealhash=caef0b.186a9b hash=dc5480.ea9e26 elapsed=29.598s
INFO [10-10|12:27:56.034] ?? mined potential block                  number=17800
6 hash=dc5480.ea9e26
INFO [10-10|12:27:56.039] Commit new mining work                   number=178007
 sealhash=34317b.ca9b4d uncles=0 txs=0 gas=0 fees=0 elapsed=0s
```

You can check the balance in the etherbase address using-

>> eth.getBalance(eth.coinbase)

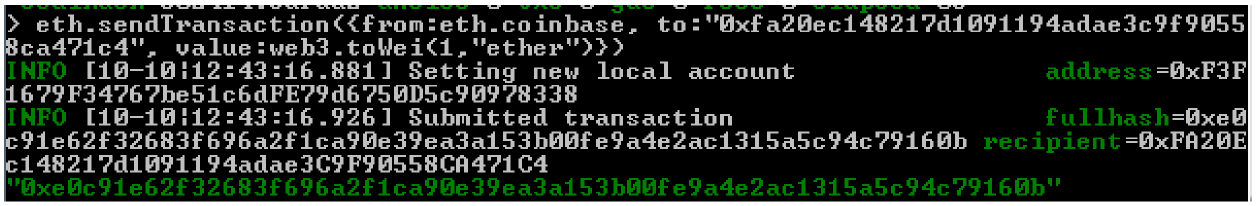


```
> eth.getBalance(eth.coinbase)
100000000000000000000
```

8-4) Send Ether as much as you want to 0xfa20ec148217d1091194adae3c9f90558ca471c4

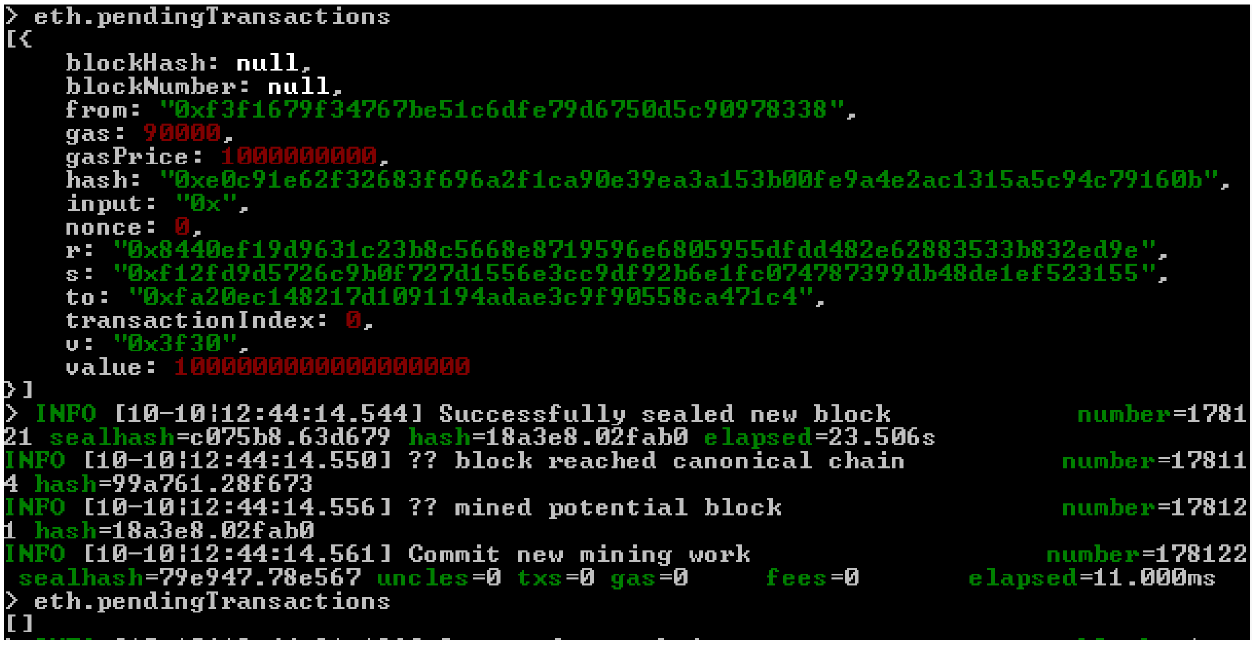
Use the following command to send ether-

>> eth.sendTransaction({from:SENDER_ADDR, to:RECEIVER_ADDR, value: web3.toWei(AMT, “ether”)})



```
> eth.sendTransaction({from:eth.coinbase, to:"0xfa20ec148217d1091194adae3c9f9055
8ca471c4", value:web3.toWei(1,"ether")})
INFO [10-10|12:43:16.881] Setting new local account                address=0xF3F
1679F34767be51c6dFE79d6750D5c90978338
INFO [10-10|12:43:16.926] Submitted transaction                    fullhash=0xe0
c91e62f32683f696a2f1ca90e39ea3a153b00fe9a4e2ac1315a5c94c79160b recipient=0xFA20E
c148217d1091194adae3C9F90558CA471C4
"0xe0c91e62f32683f696a2f1ca90e39ea3a153b00fe9a4e2ac1315a5c94c79160b"
```

After a while, the block solution was found and the block was written on the block chain



```
> eth.pendingTransactions
[{
    blockHash: null,
    blockNumber: null,
    from: "0xf3f1679f34767be51c6dfe79d6750d5c90978338",
    gas: 90000,
    gasPrice: 1000000000,
    hash: "0xe0c91e62f32683f696a2f1ca90e39ea3a153b00fe9a4e2ac1315a5c94c79160b",
    input: "0x",
    nonce: 0,
    r: "0x8440ef19d9631c23b8c5668e8719596e6805955dfdd482e62883533b832ed9e",
    s: "0xf12fd9d5726c9b0f727d1556e3cc9df92b6e1fc074787399db48de1ef523155",
    to: "0xfa20ec148217d1091194adae3c9f90558ca471c4",
    transactionIndex: 0,
    v: "0x3f30",
    value: 1000000000000000000
}]
> INFO [10-10|12:44:14.544] Successfully sealed new block            number=1781
21 sealhash=c075b8.63d679 hash=18a3e8.02fab0 elapsed=23.506s
INFO [10-10|12:44:14.550] ?? block reached canonical chain          number=17811
4 hash=99a761.28f673
INFO [10-10|12:44:14.556] ?? mined potential block                  number=17812
1 hash=18a3e8.02fab0
INFO [10-10|12:44:14.561] Commit new mining work                   number=178122
 sealhash=79e947.78e567 uncles=0 txs=0 gas=0     fees=0       elapsed=11.000ms
> eth.pendingTransactions
[]
```