**CSED 490U Blockchain & Cryptocurrency**

**Assignment 5**

****

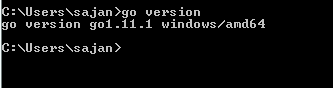
**Submitted by- Sajan Maharjan**

**POVIS id-** [**thesajan@postech.ac.kr**](mailto:thesajan@postech.ac.kr)

**Registration Number- 20182095**

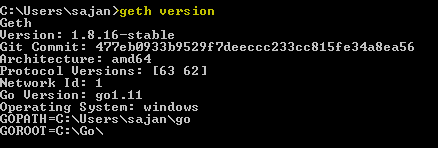
<Step. 1> Install Go language

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



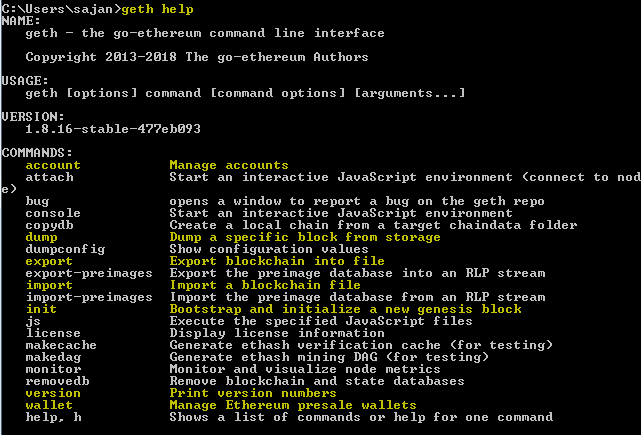
<Step. 2> Install go-ethereum (geth)

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



<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.

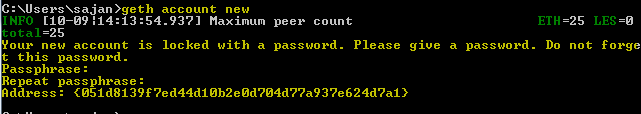


The above snapshot shows “geth” command line options for managing accounts, dumping a specific block, importing and exporting private keys, initializing genesis blocks etc. Besides these command line options there are also other options that enable us to connect to the ethereum network, ethashing option,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

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

>> geth account new

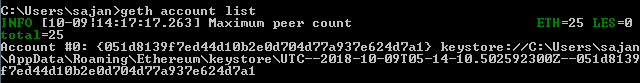
command line can be used to create a new account, which will then prompt the user to enter corresponding password to create a new account address. eg.



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

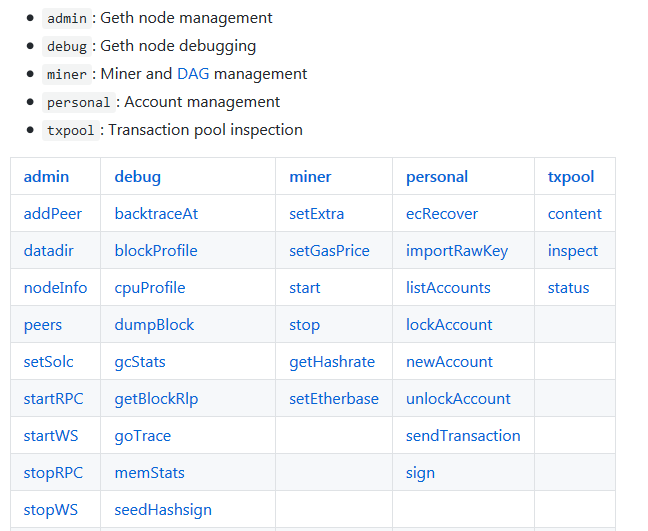


To run geth on the testnet, use the option –testnet.

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

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

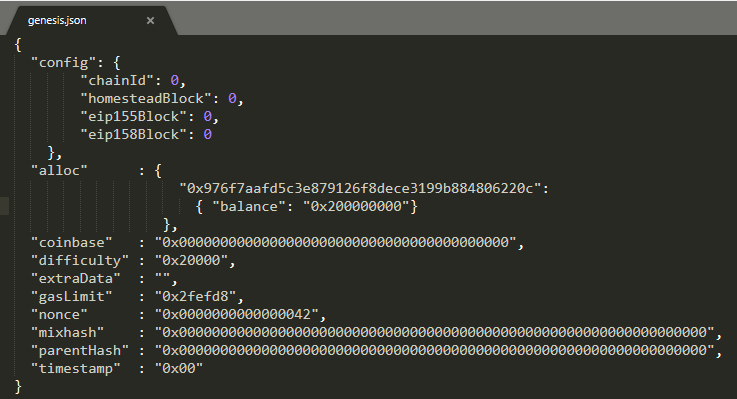
When you run this command, it will import block headers from the testnet and once its done, you can run various management APIs such as- admin, debug, miner, txpool. The following are the management APIs available:



<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-



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/*

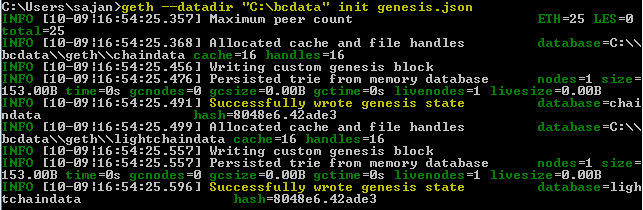
We can specify a custom directory to store blockchain data using –datadir option.

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

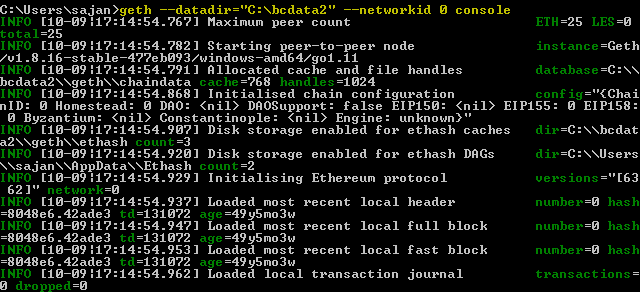


4-4) Run geth engine with console

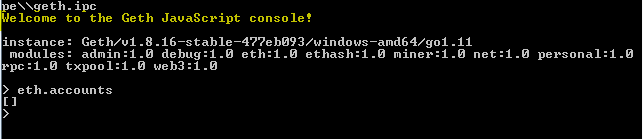
To start the geth console, connecting to the private network we just created, we can use the command-

>> 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-



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



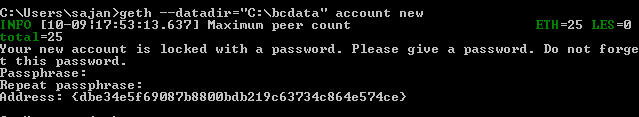
<Step. 5> Generate addresses / Mine Ether / Check the balance

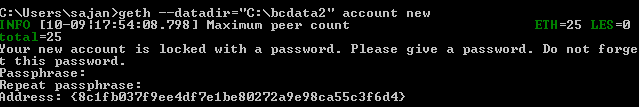
5-1) Create two addresses which can be used for sender and receiver in transmission testing.

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

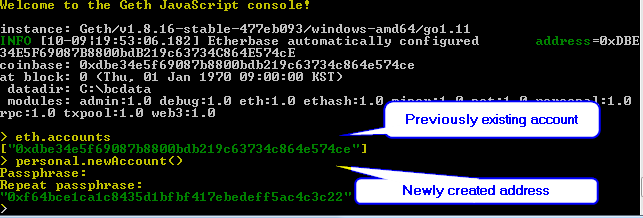




We can see that two new addresses have been created and each of these addresses are being stored in a different directory. We can also create an account from within the geth console. Let us create another account for each of these directories using the geth console-

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

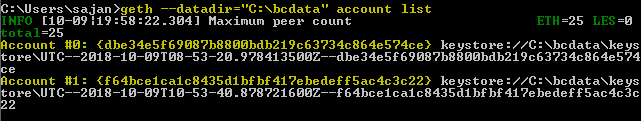
>>personal.newAccount()

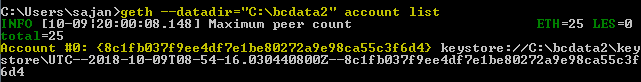
5-2) Look up your account list.

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.





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





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

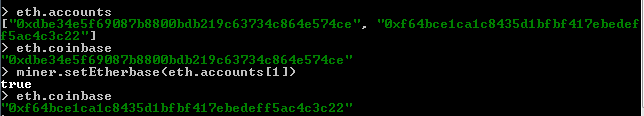
>> eth.coinbase



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.



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)



>> miner.start()



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)



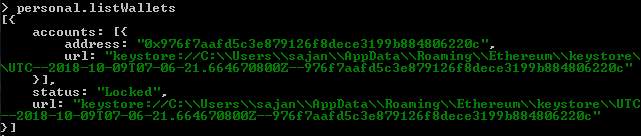
<Step. 6> Unlock the account / Send the transaction.

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

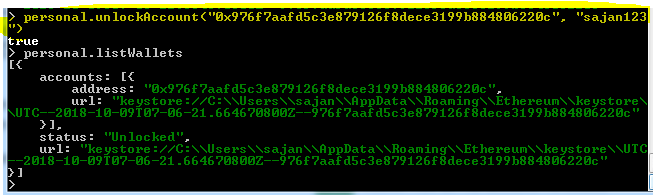


6-2) Before sending Ether, you have to unlock the sender’s account.

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.



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”)})

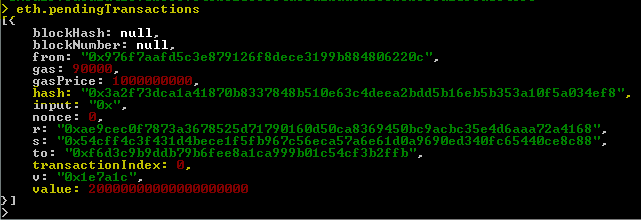


If the command executes successfully, it will return a corresponding transaction hash

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

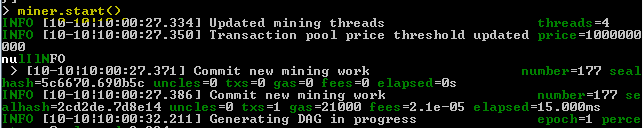


As seen from the above screenshot, the eth.pendingTransactions command will return a JSON object containing information such as from address, to address, value (in Wei) sent, transaction hash, etc.

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()



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

>> 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)



<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

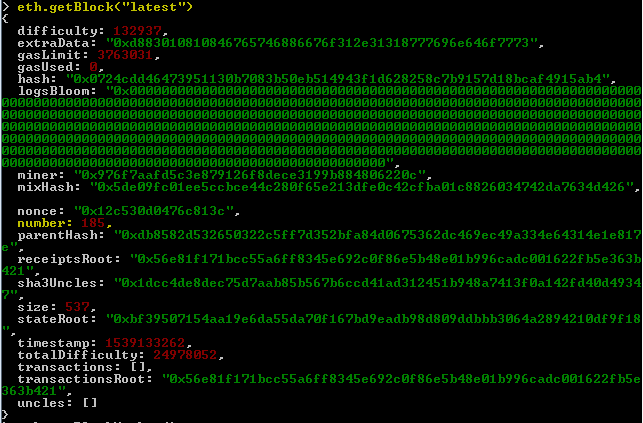
This will return an integer which is the highest number of block connected in the chain.



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



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)



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)



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.

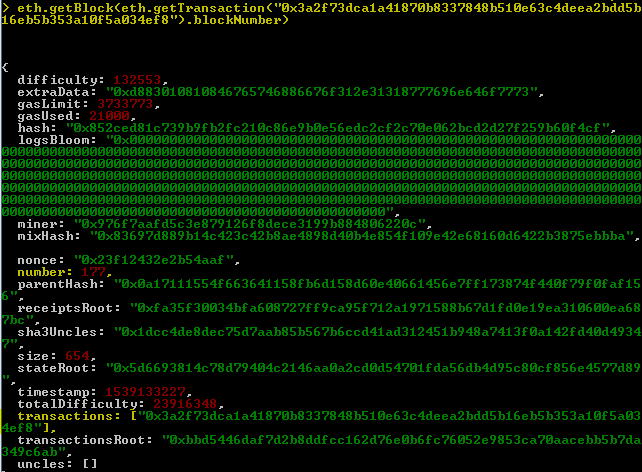
To find the transaction information, use the eth.getTransaction command passing the corresponding transaction hash as parameter.

>> eth.getTransaction(TXN\_HASH)



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)

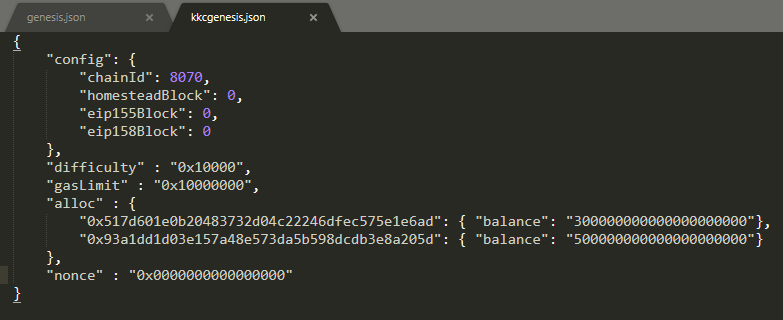


As we can see from the screenshot above, the block number 177 contains the only transaction so far we have executed and other information about this block.

<Step. 8> Join other’s private network

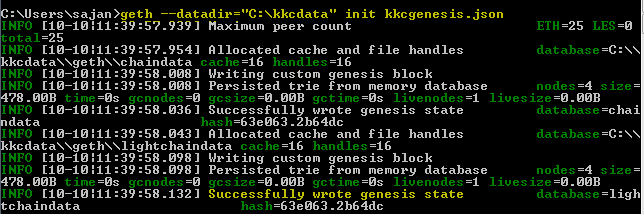
8-1) Connect the node from your PC

The given genesis file was saved as “kkcgenesis.json”



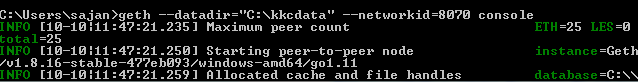
Next we would like to initialize this genesis file and store the corresponding data in the folder “C:\kkcdata”

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



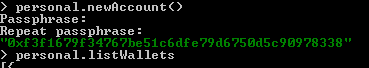
Run the console with the network id as the chain id in the given genesis file

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



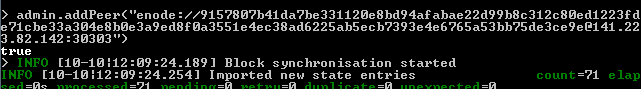
Create a new account address

>> personal.newAccount()



Use the admin.addPeer command to add the enode-

>> admin.addPeer(“enode://ENODE\_ADDR@IP\_ADDR:PORT”)



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



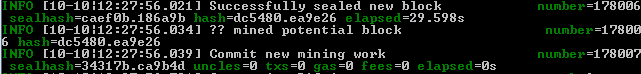
8-3) Try to mine new block

To start mining, use the command-

>> miner.start()



After a while, PoW solution to blocks were found-



You can check the balance in the etherbase address using-

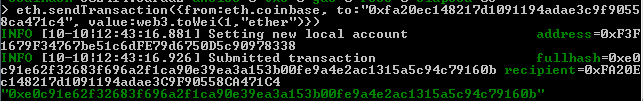
>> eth.getBalance(eth.coinbase)



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”)})



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

