1. Set up a private network with Ethereum on a single machine

A. Software needs to be installed

1. It can be installed with the built images which are downable here: https://geth.ethereum.org/docs/install-and-build/installing-geth

B. Installation verification

1. Open a shell or terminal, try to execute `geth --help`. If the help information of `geth` command is shown, it means you have installed the software correctly.
2. If the installation has been finished but the terminal does not show the help information when you run `geth --help`, it may be caused by the settings of your environment variables. Check the installation location of geth and there should be a directory which contains executable files such as "geth", "bootnode", "wnode", etc. Adding the directory in the PATH variables should make the terminal find these commands.

C. Creating/using a genesis block

1. Genesis block is the first block in a block chain. The peers and miners in the same network should have the identical genesis block for all the following blocks to be mined. The genesis block can be created by a text file. As an example, we have a file genesis.json attached with the tutorial. There are many fields defined in the file. While, for now, we only need to keep in mind the field `chainId`. Each chain has its only `chainId`. In the public network of Ethereum, the public chain has the `chainId` equal to 1. For private and test networks, we can use any integer except 1. If you want to run a private network on the local machine, please randomly choose an integer as your `chainId`. For other fields in genesis.json, please refer to https://lightrains.com/blogs/genesis-json-parameter-explained-ethereum for more information if you are interested. Now, please choose your own `chainId` and save the genesis.json file.

D. Creating a peer in the private network by specifying the genesis block

1. Some rules for peers in a private network:
   - **a)** Each peer should have its own directory/folder by using `--datadir`.
   - **b)** Each running peer should have its own port number (by using `--port`) and remote process communication port number (by using `--http.port`).
   - **c)** In case you want to have more than one terminals being associated with the same running peer, the running peer should have a unique inter-process communication pipe (by using `--ipcpath`)
2. Create a new peer (not running automatically).
   - **a)** Create a new directory called `ethereum` (other names for the new directory are also OK, but please keep the consistency for the rest steps in the tutorial). Put

genesis.json file to the new directory. Now, the file genesis.json should be at `~/ethereum/genesis.json`.

b) Create the new peer by running the following command:

```
geth --datadir "~/ethereum/peer1" init ~/ethereum/genesis.json
```

c) A new directory `peer1` should be created. You can also use other directory names if you want, but please keep the consistency in the following steps of the tutorial. In the directory `peer1`, there are two folders. Folder `geth` saves the configurations and `keystore` saves the wallets of accounts that the peer may have. Since we have not yet created any wallets, the keystore folder should be empty.

3. Run the peer

a) The peer that we have created is not running right now. To run the peer, please use the following command with the `chaindId` replaced by the `chaindId` in your genesis.json:

```
geth --datadir "~/ethereum/peer1" --nodiscover --networkid 24601 --port 12341 --http --http.port 9001 --authrpc.port 8551 --http.corsdomain "*" --ipcpath "~/ethereum/peer1/geth1.ipc" console
```

b) Some information starts to show on the screen. The information is self-explanatory, so you can briefly go through it. Please locate the line as follows:

```
INFO [mm-dd|hh:mm:ss.msc] Started P2P networking self="peer1-id@ip-address:port?discport=0"
```

The highlighted part should be the day, month, hour, minutes, second, millisecond, the running ethereum node id, the ip of the running node, and the port number of the running node.

Remember the `peer1-id@ip-address:port?discport=0`, we will use it later. Note that the "peer1-id", "ip-address", and "port" may be with different values for different persons.

c) The value after `--datadir` should be the directory that you have just created. The value of `--networkid` should be identical with the `chainId` in your genesis.json file. The `--port` parameter is for the communication purpose of other peers in the same private network. If you do not use `--port`, the default value of port will be 30303. We will create the second peer in the same private network later. Option `--http` is to enable the Remote Procedure Call for this running peer. In particular, `--http.port` specifies what the port number that the peer listens on to receive the remote process callings (default value will be 8545) and `--http.corsdomain` plays as a filter for the domain names or IP addresses that the remote procedure callings come from. The wild card `"*"` after `--http.corsdomain` simply means that the peer accepts remote procedure calling from any remote hosts (and thus, it is risky). The string parameter after `--ipcpath` is the location of the pipe for inter process communication purpose. In the example, the pipe file is at `~/ethereum/peer1/geth1.ipc`. We can connect and control the same peer

from another terminal by using command `geth attach ~/ethereum/peer1/geth1.ipc` in a new terminal/shall.

d) After executing the command, we should see the following output:

```
Welcome to the Geth JavaScript console!

instance: Geth/v1.9.25-stable-e7872729/linux-amd64/go1.15.6
at block: 0 (Wed Dec 31 1969 16:00:00 GMT-0800 (PST))
 datadir: /home/david/ethereum/peer1
 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
```

e) We see there are some modules have been automatically loaded in the console, i.e., `admin`, `debug`, `eth`, `ethash`, `miner`, `net`, `personal`, `rpc`, `txpool`, and `web3`. In the given console, we can run some functions provided by these modules. For example, we can run `admin.nodeInfo` to see the information about the node. Since we have no other peers running in the same private network, there would be no other peers listed after running command `admin.peers` in the console.

E. Run a second peer

1. Run a second peer with the following commands in a new terminal.

a) First, run the following command in the new terminal:

```
geth --datadir "~/ethereum/peer2" init ~/ethereum/genesis.json
```

Again, you can change the name of the directory to be created `peer2`, but just need to keep consistency in the following steps of the tutorial. Then, run command:

```
geth --datadir "~/ethereum/peer2" --nodiscover --networkid 24601 --port 12342 --http --http.port 9002 --authrpc.port 8552 --http.corsdomain "*" --ipcpath "~/ethereum/peer2/geth2.ipc" console
```

b) Still, form the output, we need to locate something like

```
INFO [mm-dd|hh:mm:ss.msc] Started P2P networking self="peer2-id@ip-address:port?discport=0"
```

Remember the `peer2-id@ip-address:port?discport=0`, we will use it later. Note that the "peer2-id", "ip-address", and "port" may be with different values for different persons.

c) Now, the second peer is running in the second terminal. However, these two peers cannot see each other yet. Thus, if we run command `admin.peers` now in one of the two consoles, we still cannot see the other peer in the private network, even though both of the two peers have the same `networkid` equal to `chainId` in your genesis.json. While, `networkid` can be also revealed by running `admin.nodeInfo` in the console of the running peer.

2. To trigger the node discovery, we need to run `admin.addPeer()` function.

a) We can choose either one of these two peers to add the other. Assume that we are in the console of the first peer. We need to run command with `peer2-id`:

```
admin.addPeer("peer2-id@ip-address:port?discport=0")
```
The I"peer2-id", "ip-address", and "port" should be the same with the values you have found in Section E1-b).

b) Now, in either of the two consoles, when we run `admin.peers`, we should see the other peer get printed out.

F. Create your first account and mine the blocks

1. We have no account on either of the peers. We have to create an account before mining, so the rewards of mining blocks can be received by the account.
2. Run command
   `personal.newAccount()`
   in the first terminal. According to the name of the above command, we should realize that there is actually a function called `newAccount()`, requiring no argument, defined in the module `personal`.
3. The console will ask you to enter the password twice. There is no way to recover your password if you forget it, so please remember your password firmly. We suggest using `EECE571` as the password for test purposes.
4. Assume the command `personal.newAccount()` has been executed in the console of the first peer, now in the console we can call function
   `miner.start(1)`
   to start the mining process with 1 worker. You can also increase the number if you want. Apparently, you may not want to occupy too many workers if you are mining with CPU, since your computer will be very slow due to the limited number of cores on your CPU. However, the more workers we use, the faster the blocks can be generated in your block chain. It will return you a `null` after you run the command, but it is OK.
5. Now, run command
   `eth.blockNumber`
   in the same terminal, it will show you the number of blocks in the current block chain. The number of blocks in the current block chain should increase as your mining process is going.
6. Run command `eth.blockNumber` in the other console, it will show you the number of blocks as well. The value should be the same with that in your first console.
7. Using
   `miner.stop()`
   to stop the mining process, and the results of `eth.blockNumber` would not change any more in both consoles.

G. Check your balance after mining

1. Assume you have run `miner.start(3)` in a console for a while, you can check your balance in the same console by running
   `eth.getBalance(eth.accounts[0])`
   Here, `eth.accounts` will return a list of all accounts that we have created for the peers. Since we have just created one account, `eth.accounts[0]` returns the only account that we have with the peer. Function `eth.getBalance()` takes an account as the input argument to check its balance.

H. Extensions

1. More peers can be created and added into the private network. Note that each peer should be placed in a new directory and run with a unique port number, remote process communication port number, and inter process communication path.
2. The new created peer cannot be automatically known by the existing peers. In the next tutorial, we will introduce the `bootnode` which is a server process automatically notifying the existing peers for a new joining peer.
3. Each peer could be a miner and all these peers can mine the blocks together. Note that before the miner is running, as least one account should be created.

I. If you have any problems when you try to reproduce the experiment, please contact me for help. Email: zwang@ece.ubc.ca

J. In the next tutorial, we will learn how to enable the peers in a private network to discover each other automatically and how to create a transaction to transfer Ethers between accounts.