

Installing Ethereum on Ubuntu (Linux Recommended)

To create an Ethereum Private Network, we first need to have Ethereum installed in our system. In this section of Ethereum Private Network Tutorial, you will learn how to install Ethereum on Ubuntu.

To install Ethereum, run the following commands in a terminal:

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository -y ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install ethereum
Done! This will install Ethereum on your system.
```

Let's start with the Private Network creation.

Demo: Creating Ethereum Private Network and making a Transaction

In this Ethereum Private Network Tutorial, we will send ethers from one account to another and so, we need accounts. Let's now see how to create accounts for our Blockchain.

Creating Accounts for Ethereum Private Network

Before creating new accounts, let us create a new directory for our workplace. Refer to the below commands to do this:

```
$ mkdir private-ethereum
$ cd private-ethereum
```

To make a transaction, we need at least two accounts: A receiver and a sender.

To create two accounts, run the following command twice:

```
$ geth --datadir ./datadir account new
```

Enter the **passphrase** for each account when asked. Do not forget this passphrase!

Once these commands run successfully, two accounts will be created and the account address will be displayed on the screen.

```
desteuque@desteuque-VirtualBox:~$ mkdir private-ethereum
desteuque@desteuque-VirtualBox:~$ cd private-ethereum
desteuque@desteuque-VirtualBox:~/private-ethereum$ geth --datadir ./datadir account new
INFO [09-26|13:01:20.225] Maximum peer count          ETH=50 LES=0 total=50
INFO [09-26|13:01:20.225] Smartcard socket not found, disabling  err="stat /run/pcscd/pcscd.comm:
no such file or directory"
Your new account is locked with a password. Please give a password. Do not forget this password.
Password:
Repeat password:

Your new key was generated

Public address of the key: 0xf5F80372781ce9c2c63a82AcCA57Da9367FDBfb8
Path of the secret key file: datadir/keystore/UTC--2019-09-26T11-01-28.084120091Z--f5f80372781ce9c2
c63a82acca57da9367fdbfb8

- You can share your public address with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your funds!
- You must BACKUP your key file! Without the key, it's impossible to access account funds!
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!

desteuque@desteuque-VirtualBox:~/private-ethereum$ geth --datadir ./datadir account new
INFO [09-26|13:01:41.723] Maximum peer count          ETH=50 LES=0 total=50
INFO [09-26|13:01:41.723] Smartcard socket not found, disabling  err="stat /run/pcscd/pcscd.comm:
no such file or directory"
Your new account is locked with a password. Please give a password. Do not forget this password.
Password:
Repeat password:

Your new key was generated

Public address of the key: 0x9aa4993D0F5dD3b495545336112968974F4Dcf20
```

Save these addresses somewhere because we will be using these further.

Creating Genesis File

A Genesis file contains the properties that define the Blockchain. A Genesis file is the start-point of the Blockchain and so, it is mandatory to create the Genesis file to create a Blockchain. Now, let's create the Genesis file.

First, create a file named **genesis.json**

\$ nano genesis.json

And now copy and paste the following code in that file:

```
{
  "config": {
    "chainId": 2019,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0
  },
  "difficulty": "400",
  "gasLimit": "2000000",
  "alloc": {
    "f5F80372781ce9c2c63a82AcCA57Da9367FDBfb8" : {
      "balance": "10000000000000000000000000000"
    },
    "9aa4993D0F5dD3b49554533611296B974F4Dcf20" : {
      "balance": "12000000000000000000000000000"
    }
  }
}
```

Note: In the above code, replace the address under **alloc** section with the address of the accounts that you created in the previous step.

```
GNU nano 3.2 genesis.json
{
  "config": {
    "chainId": 2019,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0
  },
  "difficulty": "1",
  "gasLimit": "2000000",
  "alloc": {
    "f5F80372781ce9c2c63a82AcCa57Da9367FDBfb8": {
      "balance": "1000000000000000000"
    },
    "9aa4993D0F5dD3b49554533611296B974F4Dcf20": {
      "balance": "1200000000000000000"
    }
  }
}
```

[Read 19 lines]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

Save it and exit.

Let me explain the contents of the Genesis file in brief:

chainId – This is the chain identification number that is used to distinguish between Blockchains

homesteadBlock, eip155Block, eip158Block, byzantiumBlock – these properties are related to chain forking and versioning. We don't need these for our tutorial, so let's set them to 0.

difficulty – This number decides how difficult the blocks will be to mine. For Private networks, it's good to set a lower number as it lets you mine blocks quickly, which results in fast transactions.

gasLimit – This number is the total amount of gas that can be used in each block. We don't want our network to hit the limit, so we have set this high.

alloc – This part is used to allocate ethers to already created accounts.

The Genesis file is ready. Now, it's time to start the Blockchain.

Instantiating Data Directory

Before starting the Blockchain, we have to instantiate the data directory. The Data Directory is the directory where the data related to the Blockchain is stored. To instantiate the data directory, run the following command:

```
$ geth --datadir ./myDataDir init ./genesis.json
```

On successful instantiation, you should see the following output:

```
desteuque@desteuque-VirtualBox:~/private-ethereum$ geth --datadir ./myDataDir init ./genesis.json
INFO [09-26|13:27:42.317] Maximum peer count                ETH=50 LES=0 total=50
INFO [09-26|13:27:42.317] Smartcard socket not found, disabling err="stat /run/pcscd/pcscd.comm:
no such file or directory"
INFO [09-26|13:27:42.318] Allocated cache and file handles database=/home/desteuque/private
-ethereum/myDataDir/geth/chaindata cache=16.00MiB handles=16
INFO [09-26|13:27:42.337] Writing custom genesis block
INFO [09-26|13:27:42.337] Persisted trie from memory database nodes=3 size=409.00B time=148.53
2µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [09-26|13:27:42.338] Successfully wrote genesis state database=chaindata hash=81ee19...5
1a8c1
INFO [09-26|13:27:42.338] Allocated cache and file handles database=/home/desteuque/private
-ethereum/myDataDir/geth/lightchaindata cache=16.00MiB handles=16
INFO [09-26|13:27:42.352] Writing custom genesis block
INFO [09-26|13:27:42.352] Persisted trie from memory database nodes=3 size=409.00B time=71.152
µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
INFO [09-26|13:27:42.353] Successfully wrote genesis state database=lightchaindata hash=81e
e19...51a8c1
desteuque@desteuque-VirtualBox:~/private-ethereum$
```

With the Data Directory instantiated, we can now start the Blockchain.

Starting Ethereum Private Blockchain

To start the Blockchain, run the following command:

```
$ geth --datadir ./myDataDir --networkid 2019 console 2>> Eth.log
```

Done! Your private Ethereum Blockchain is up and running.

In the above command, we are sending all the logs in a separate file called **Eth.log**. Geth will automatically create a new file if it is not found.

The output of this code should look something like this:

```
desteuque@desteuque-VirtualBox:~/private-ethereum$ geth --datadir ./myDataDir --networkid 2019 console 2>> Eth.log
Welcome to the Geth JavaScript console!

instance: Geth/v1.9.5-stable-a1c09b93/linux-amd64/go1.11.5
at block: 0 (Thu, 01 Jan 1970 01:00:00 CET)
 datadir: /home/desteuque/private-ethereum/myDataDir
 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

> █
```

Now, we have entered into the **geth console** where we can run commands for our Blockchain.

Reading Logs

In the previous section, I mentioned that we are storing the logs in another file. In this section, I will tell you how to read the logs from this file.

We will read the logs from a separate terminal, so first let's open a new terminal. First, switch to the **private-ethereum** directory and then run the following command to read the logs:

```
$ tail -f Eth.log
```

```

desteuque@desteuque-VirtualBox:~/private-ethereum$ tail -f Eth.log
INFO [09-26|13:43:40.323] Loaded most recent local full block      number=0 hash
=d4e567...cb8fa3 td=17179869184 age=50y5mo2w
INFO [09-26|13:43:40.323] Loaded most recent local fast block       number=0 hash
=d4e567...cb8fa3 td=17179869184 age=50y5mo2w
INFO [09-26|13:43:40.323] Regenerated local transaction journal    transactions=
0 accounts=0
INFO [09-26|13:43:40.330] Allocated fast sync bloom                size=322.00Mi
B
INFO [09-26|13:43:40.386] New local node record                    seq=1 id=93aa
bfd0f160378c ip=127.0.0.1 udp=30303 tcp=30303
INFO [09-26|13:43:40.388] IPC endpoint opened                      url=/home/desteuque/private-ethereum/myDataDir/geth.ipc
INFO [09-26|13:43:40.407] Started P2P networking                   self=enode://
a4a7edef8c84eedadb61451ddcc87d64ee3da4251c2dd5302af666342456e97050b6231997a86db0
c4c3278016b984fe58f489de47b725f47df1058b6a1299d0@127.0.0.1:30303
INFO [09-26|13:43:40.583] Initialized fast sync bloom              items=12356 e
rrorrate=0.000 elapsed=251.655ms
WARN [09-26|13:43:40.607] Served eth_coinbase                      reqid=3 t=27.
862µs err="etherbase must be explicitly specified"
INFO [09-26|13:43:46.879] New local node record                    seq=2 id=93aa
bfd0f160378c ip=37.71.143.35 udp=49199 tcp=30303

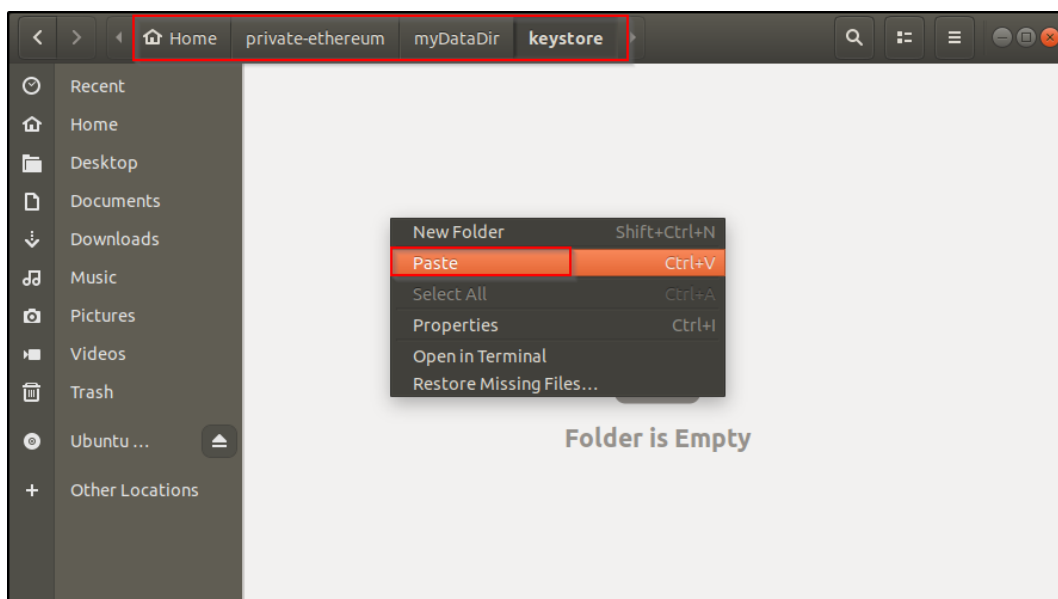
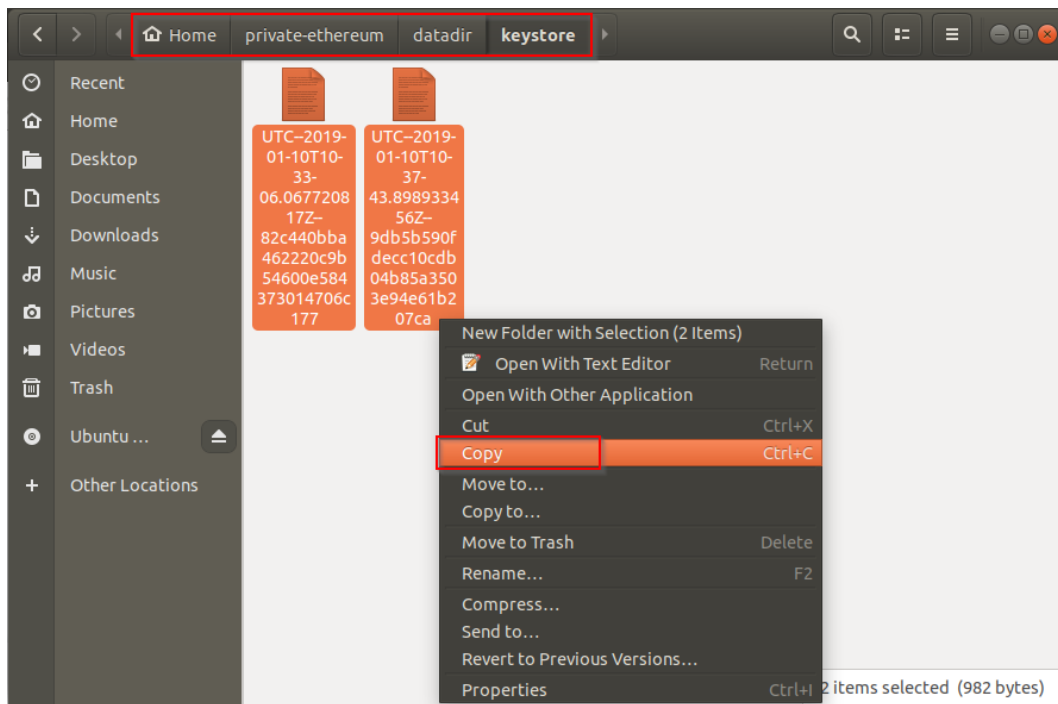
```

You can now see the logs in the terminal. These logs get dynamically updated whenever there is some activity in the Blockchain.

Importing accounts to Private Network

You may remember that we created two accounts to make transactions. But, we didn't add these accounts to our network. So, in this section of Ethereum Private Network tutorial, I will tell you how to import the accounts.

When we create an account, all the details of the account is stored in a **UTC file** in the directory mentioned during account creation (path: **./datadir/keystore**). To import the accounts, we need to copy these files and paste in the **keystore** directory under the Data Directory (path: **./myDataDir/keystore**)



That's all! The accounts are imported. Simple, isn't it? To verify the import, we will run the following command in the **geth console**.

```
> eth.accounts
```

This will show a list of all accounts available.

To check the balance of these accounts, we will use the following command:

```
> web3.fromWei(eth.getBalance(<address_of_account>), "ether")
```



```
> eth.accounts
[
  "0xf5f80372781ce9c2c63a82acca57da9367fdbfb8", "0x9aa4993d0f5dd3b49554533611296b974f4dcf20"
]
> web3.fromWei(eth.getBalance("0xf5f80372781ce9c2c63a82acca57da9367fdbfb8"), "ether")
0.1
> web3.fromWei(eth.getBalance("0x9aa4993d0f5dd3b49554533611296b974f4dcf20"), "ether")
0.12
```

We are ready with everything required to make a transaction.

Network Connectivity

With all nodes that you want to run initialized to the desired genesis state, you'll need to start a bootstrap node that others can use to find each other in your network and/or over the internet. The clean way is to configure and run a dedicated bootnode:

```
bootnode --genkey=boot.key
bootnode --nodekey=boot.key
```

With the bootnode online, it will display an enode URL that other nodes can use to connect to it and exchange peer information. Make sure to replace the displayed IP address information (most probably [::]) with your externally accessible IP to get the actual enode URL.

Note: You can also use a full fledged Geth node as a bootstrap node.

Starting Up Your Member Nodes

With the bootnode operational and externally reachable (you can try `telnet <ip> <port>` to ensure it's indeed reachable), start every subsequent Geth node pointed to the bootnode for peer discovery via the `--bootnodes` flag. It will probably also be desirable to keep the data directory of your private network separated, so do also specify a custom `--datadir` flag.

```
geth --datadir path/to/custom/data/folder --networkid 15 --
bootnodes <bootnode-enode-url-from-above>
```

Since your network will be completely cut off from the main and test networks, you'll also need to configure a miner to process transactions and create new blocks for you.

Running A Private Miner

Mining on the public Ethereum network is a complex task as it's only feasible using GPUs, requiring an OpenCL or CUDA enabled ethminer instance. For information on such a setup, please consult the EtherMining subreddit and the Genoil miner repository.

In a private network setting however, a single CPU miner instance is more than enough for practical purposes as it can produce a stable stream of blocks at the correct intervals without needing heavy resources (consider running on a single thread, no need for multiple ones either). To start a Geth instance for mining, run it with all your usual flags, extended by:

```
$ geth <usual-flags> --mine --minerthreads=1 --  
etherbase=0x0000000000000000000000000000000000000000
```

Which will start mining blocks and transactions on a single CPU thread, crediting all proceedings to the account specified by --etherbase. You can further tune the mining by changing the default gas limit blocks converge to (--targetgaslimit) and the price transactions are accepted at (--gasprice).

Making A Transaction

In this Ethereum Private Network tutorial, we will send some ethers from one account to another.

The syntax to send ethers is as follows:

```
> eth.sendTransaction({from:"address", to:"address", value:  
web3.toWei(amount, "ether")})
```

We will send 1000 ethers from account 1 to account 2 using the following command:

```
> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1],  
value: web3.toWei(1000, "ether")})
```

Didn't work? This is because the account is locked by default and doesn't allow transactions.

```
> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1], value: web3.toWei(1000, "ether")})  
Error: authentication needed: password or unlock  
    at web3.js:3143:20  
    at web3.js:6347:15  
    at web3.js:5081:36  
    at <anonymous>:1:1
```

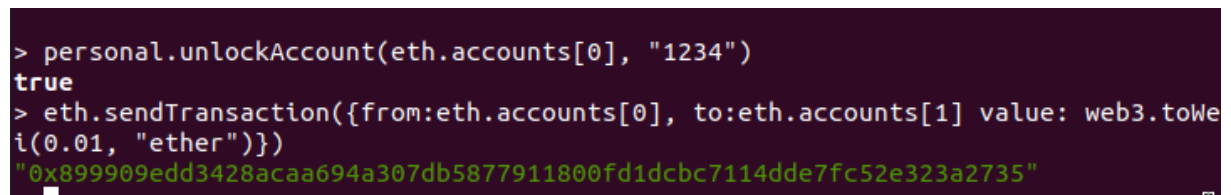
So, first, we need to unlock the sender account. Remember the passphrase you used while creating the account? Well, you must, because you will have to use that to unlock the account. We will unlock the account with the following command:

```
personal.unlockAccount(eth.accounts[0], "<password>")
```

Now we will send ethers successfully:

```
> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1],  
value: web3.toWei(1000, "ether")})
```

This should return a Transaction ID.



```
> personal.unlockAccount(eth.accounts[0], "1234")  
true  
> eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1] value: web3.toWei(0.01, "ether")})  
"0x899909edd3428acaa694a307db5877911800fd1dcbc7114dde7fc52e323a2735"
```

Done! You have successfully made a transaction!

But before checking the balance in both the accounts you need to mine a block.

Please now start to mine it by launching the command `miner.start()`. Wait a few moments to be sure that the block was mined and check again your account.

If you are not seeing an increase in value is because you have set up your bootnode incorrectly.