LearnCoin: An Educational Implementation of Cryptocurrencies

Brian Sang, Solon High School

Cryptocurrencies such as Bitcoin and Litecoin have recently been the subject of much speculation, investment, and development. Due to the open source nature of these two cryptocurrencies, it is feasible for users to create their own alternative cryptocurrencies (known as “altcoins”) based on previously written source code. Services that offer users the ability to create their own altcoins for a small fee are currently available. However, using these services may not offer as much insight into the inner workings of the Bitcoin proof-of-work system. By providing an instructional and educational resource for the creation of one’s own cryptocurrency, we hope to promote further understanding of the proof-of-work system.

LearnCoin is based off of SmallChange, another cryptocurrency which is itself based off of the original Litecoin source code. Users will be able to modify LearnCoin into their own cryptocurrency. Ideally users should have at least two computers running Ubuntu 12.04 to begin, although other versions of Ubuntu or Linux distributions will probably work as well.

A Brief Explanation of the Bitcoin Protocol

In the most commonplace method of securing digital transactions today, a central institution, e.g. a bank or government body, sees and verifies all transactions.

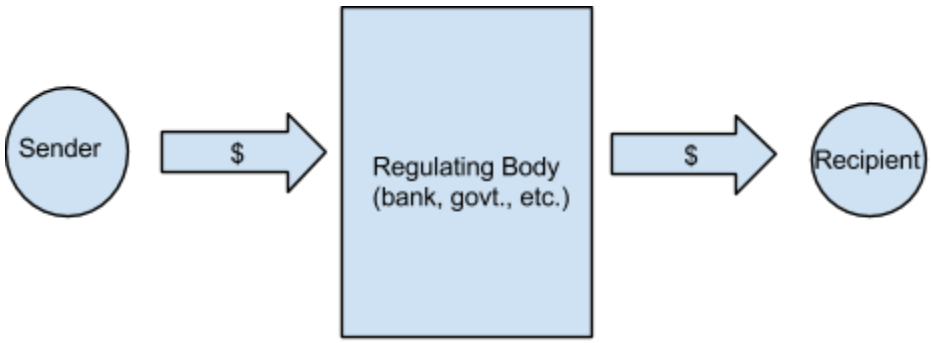


Figure 1.1 Two users have the Regulating Body as a third party to verify their transaction

Assuming the central institution is large, powerful, and trustworthy enough, this method does ensures conformity and security for all transactions and users. However, as we do not live in a perfect world, it does pose the following problems:

1. How can we trust this central authority in being able to see all of our transactions and identities and possibly use this information against us?

2. Likewise, how can we trust this central authority to not modify our transactions?

3. Who controls this central authority? Does this authority represent common people?

4. By making itself so prominent, would such a central authority make itself the target of attacks? If it is not powerful enough to repel these attacks, would its security be compromised?

The Bitcoin Protocol attempts to address these concerns with a distributed, peer-to-peer network of users. That is, there is *no* central institution to verify transactions in the Bitcoin protocol. Instead, the legitimacy of transactions is to be determined by the majority acceptance of the users.

How it Works

The Bitcoin network essentially functions as a method of timestamping transactions to ensure that attackers are unable to spend the same money twice. Computers connected to the Bitcoin network (“nodes”) are linked to other nodes. Using their CPU power, nodes then attempt to produce a proof-of-work - a solution to a computationally complex problem (think of it like a math puzzle).

During this time, any transactions that occur between two nodes are broadcasted to the rest of the network on a best-effort basis. That is, the information of the transaction is relayed to as many nodes as possible.

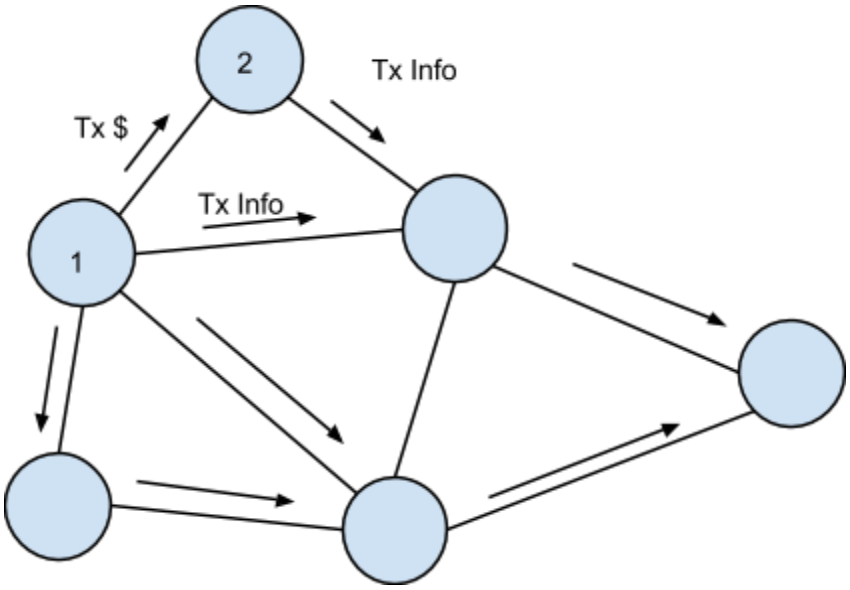


Figure 1.2 A transaction between nodes 1 and 2 having its information sent to other nodes on the network

Once a node finds a solution to the aforementioned problem, it creates a “block” containing its solution as well as a list of all transactions it has heard about. It then broadcasts this block to the rest of the network on a best-effort basis. Each node then verifies that all transactions in the block are valid and not already spent. The solution is easily verifiable by the rest of the nodes to be correct, and the newly “mined” block is accepted by the other nodes and added to their copy of the “blockchain” - a list of all blocks, with each node having a copy, created by the network serving as a public record of transactions. As an incentive for donating CPU power, the first transaction in every block is a reward of coins given to the node that correctly solved the problem and thus “mined” or created the block.

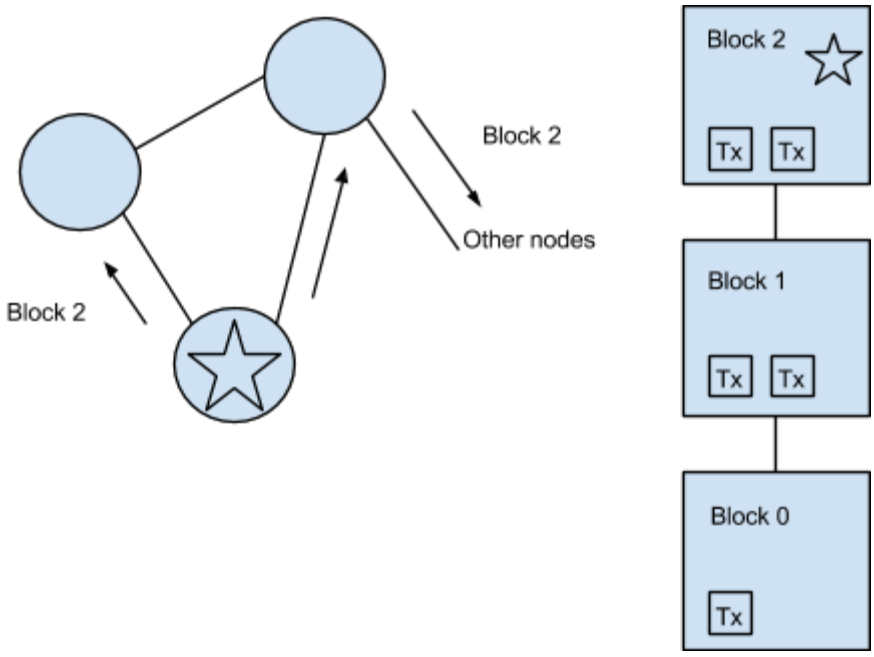


Figure 1.3 The starred block solves the mathematical problem and creates Block 2, sending it to other nodes who upon verifying it add it to their own copies of the blockchain.

How does this protect against attackers? Firstly, the computationally difficult problem means that any attacker trying to modify past blocks would be unable to catch up with the speed and power of the honest nodes, assuming that a majority of nodes are honest. This is because the protocol represents the majority of the honest nodes as the longest blockchain created. Attackers attempting to create illicit blocks without properly solving the computationally difficult problem would find their blocks promptly rejected by other nodes on the network. Therefore, in order to modify past blocks, an attacker would not only have to resolve the computationally difficult problem for that block, but also outpace the honest nodes in creating new blocks to form a longer blockchain. This would only be possible if an attacker had a majority of the CPU power in the network.

Secondly, attackers who attempt to “double spend” their coins would be foiled as not only are nodes will only accept blocks if the transactions are valid and not doubly spent (due to the delay between blocks, normal methods of double spending attacks cannot work). Additionally, even if an attacker were to obtain a majority of the network’s CPU power, the attacker should find it more favorable to play by the rules due to the incentive for mining blocks. By attacking the blockchain, an attacker would thus devalue the currency as a whole (due to increased security risks) and harm their own wealth. By mining blocks, the attacker not only supports the growth of the network (and thus supports the stability of their own wealth), but also receives reimbursement in the process.

If there is a group of isolated nodes on the network who work separately and create a fork of the main blockchain, when any node in the isolated group reconnects with the larger main network it will choose to comply with the larger main blockchain as it represents the majority work and majority decision. Any transactions then within the smaller isolated group will be added to the next block of the main network.

Necessary Packages

The following packages must be installed:

libboost-dev	Boost C++ Libraries development files
libboost-system-dev	Operating system (e.g. diagnostics support) library
libboost-program-options-dev	program options library for C++
libboost-thread-dev	portable C++ multi-threading
libboost-filesystem-dev	filesystem operations in C++
libcurl-4-openssl-dev	development files and documentation for libcurl (OpenSSL)
libdb5.1-dev	Berkeley v5.1 Database Libraries
libdb5.1++-dev	Berkeley v5.1 Database Libraries for C++
git	fast, scalable, distributed revision control system
qt-sdk	Complete Qt Software Development Kit
libminiupnpc-dev	UPnP IGD client lightweight library development files
build-essential	Informational list of build-essential packages
openssl	Secure Socket Layer (SSL) binary and related cryptographic tools

Descriptions taken from packages.ubuntu.com

They can be installed by using the command `apt-get install <packagename>` (for Debian/Ubuntu users) while having superuser permissions. Alternatively they can be found at packages.ubuntu.com. Installing these packages will allow us to successfully compile LearnCoin.

Copying the Source Code

Git is free software to keep track of and manage source code. By linking the source code to an online git hosting service, we will be able to more easily control updates to our source code for multiple platforms. It will also allow easy transmission of source code files through the internet.

A copy of the LearnCoin source code can be obtained from https://github.com/Baisang/LearnCoin.git. It is recommended to create a separate git repository either through GitHub or another host such as Bitbucket, as some files will have to be changed and updated in the process. Before beginning, be sure to create an account at one of these websites.

Begin by cloning the LearnCoin source code into your own directory:

```
~$ git clone https://github.com/Baisang/LearnCoin.git
~$ cloning into LearnCoin
```

If one wishes to rename LearnCoin into for example NewCoin, the command

```
~$ mv LearnCoin NewCoin
```

will rename the directory LearnCoin is in.

Now we will have to remove LearnCoin's ties to its original git repository. This can be easily done with the command:

```
~$ rm -r -f .git
```

which will remove all files within the hidden folder git (hidden folders are specified with a . before their name) and in all subdirectories within it.

To initialize a git repository for the directory and to copy it to the cloud

```
~$ git init
initializing git repository in ~/NewCoin
```

```
~$ git add -A *
```
(adds all files to repository, due to the -A flag)

```
~$ git  commit -m "first"
```
(commit records or takes a "snapshot" of what we added previously. The -m flag allows us to attach a message. Usually this message will indicate what the change was.)

```
~$ git remote add origin https://github.com/yourusername/NewCoin.git
```

(this will link the repository to our online account on a git repository website. GitHub is used as an example - others are widely available.)

```
~$ git push -u origin master
```

(this will push our current files out to the online git repository we created. The -u flag will allow us to later “pull” our files without having to specify which branch we are pulling from. Of course we are pushing our files out as the master branch.)

At this point the server for your online repository host will likely prompt for username and password.

Modifying LearnCoin Source

Names

To begin, we must replace all instances of LearnCoin with NewCoin (or any other name for the coin) in the following files:

LearnCoin-qt.pro
/src/irc.cpp
/src/init.cpp
/src/main.cpp
/src/util.cpp
/src/bitcoinrpc.cpp
/src/net.cpp
/src/rpcdump.cpp

One can view files that contain the word “LearnCoin” by using the grep command. The following command, when used in the /NewCoin/ folder contained in the root directory, will return a list of all files containing “LearnCoin”:

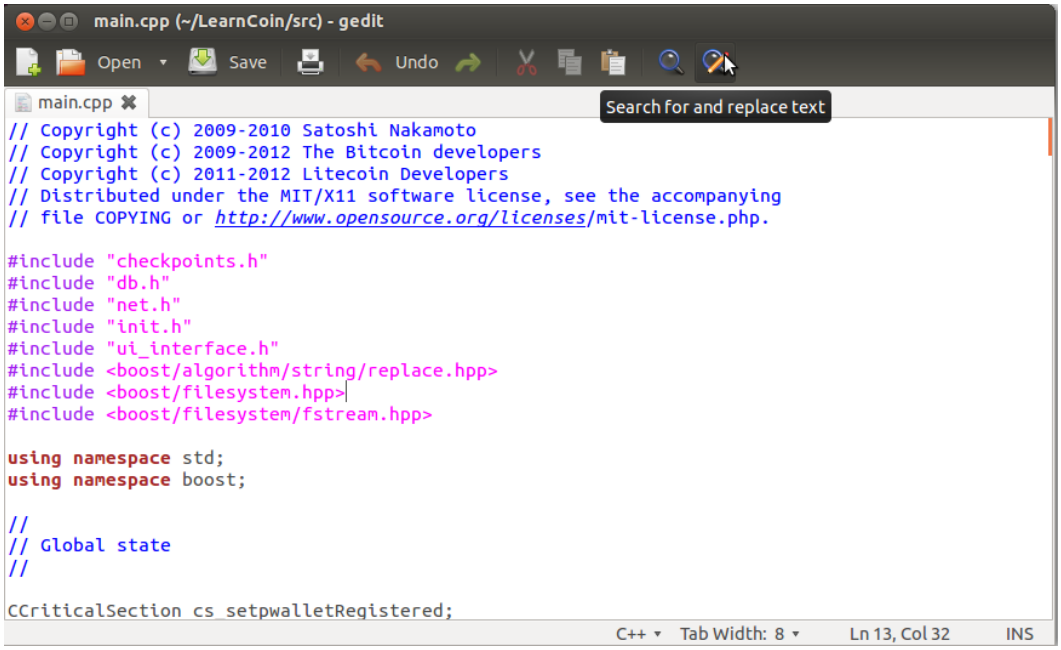
```
grep -i -r “LearnCoin” ~/NewCoin/
```

-i tells grep to ignore case, while -r tells it to search recursively in all subdirectories of ~/NewCoin/

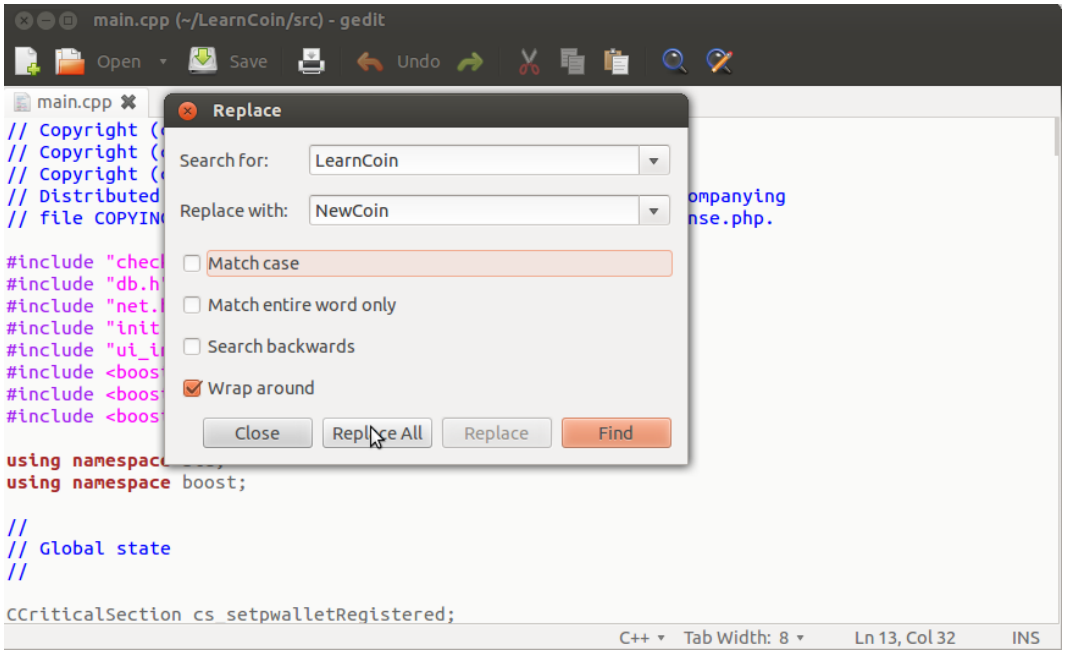
To replace instances of LearnCoin in these files, open the file with gedit (should be automatically installed in Ubuntu):

```
~/LearnCoin/src$ gedit main.cpp
```

Use the “Search for and replace text” tool



Be sure that “Match Case” is left unchecked



Also, change

/src/base58.h Line 280

```
PUBKEY_ADDRESS = 19, // addresses start with L
```

to the corresponding base 58 representation of your coin's first letter, available here: https://en.bitcoin.it/wiki/Base_58_Encoding

Ports

Users who wish to have their coin used in a real environment should also change the ports associated with LearnCoin to new ones for their own cryptocurrency. When an application is said to use a specific port, it will listen for connections through that port. As often ports are reserved for specific applications, it will be easy to uniquely identify communications through a port as belonging to a specific application. It is recommended to use a port unused by another popular application. A high-numbered port is recommended. A list of well-known ports can be found here: http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers. LearnCoin already uses ports

Different ports will have to be used for RPC (Remote Procedure Call - for miners) and P2P (peer to peer - how different machines running the coin will connect to each other). Additionally a testnet port can be specified.

/src/protocol.h Line 22 Testnet and P2P

```
return testnet ? 36327 : 36329; //testnet : P2P
```

/src/bitcoinrpc.cpp Line 2892 RPC

```
ip::tcp::endpoint endpoint(bindAddress, GetArg("-rpcport", 36328));
```

/src/bitcoinrpc.cpp Line 3168 RPC

```
if (!d.connect(GetArg("-rpcconnect", "127.0.0.1"), GetArg("-rpcport",
"36328")))
```

The following two are comments so it is not completely necessary to change them, however it is highly recommended.

/src/init.cpp Line 235 P2P and testnet

```
"  -port=<port>            " + _("Listen for connections on <port>
(default: 36329 or testnet: 36327)") + "\n" +
```

/src/init.cpp Line 277 RPC

```
"  -rpcport=<port>         " + _("Listen for JSON-RPC connections on
<port> (default: 36328)") + "\n" +
```

Coins

To change the amount of coins rewarded per block:

/src/main.cpp Line 831

```
int64 nSubsidy = 5 * COIN; //Amt of coins per block
```

Notice that the below lines are commented out:

```
//    if(nHeight < 17280) // no block reward within the first 3 days
//          nSubsidy = 0;
```

This ensures that the first blocks mined on the LearnCoin network will reward coins, rather than waiting 3 days to give rewards to block miners. By default LearnCoin gives 5 coins per block. Further below, within the same method:

```
static const int64 nTargetTimespan = 0.35 * 24 * 60 * 60; //
LearnCoin: 0.35 days
static const int64 nTargetSpacing = 15; // LearnCoin: 15 seconds
```

nTargetTimespan is the time LearnCoin takes to readjust the difficulty of mining. The difficulty is of course a function of the current amount of LearnCoin in circulation - the more LearnCoin in circulation, the more difficult to mine.

nTargetSpacing is the ideal time it should take for a block to be found. Modifying this will also modify the difficulty to allow for blocks to be mined more quickly or more slowly.

And

```
if(nHeight > 10519200) // no block reward after 5 years
      nSubsidy = 0;
```

This will mean that blocks will not reward 5 coins after 5 years of LearnCoin running. Any coins received by the miner will be from collecting transaction fees.

To alter the maximum number of coins

/src/main.h Line 43

```
static const int64 MAX_MONEY = 42007680 * COIN; // maximum of 42 007
680 coins
```

/src/main.h Line 550

```
return dPriority > COIN * 5760 / 250; // 5760 blocks found a day (4
blocs/min * 60min/hr * 24hr/day).
```

Target blocks per day can be calculated by using your predetermined nTargetSpacing value and seeing how many blocks would be found in a day based on that value.

Merkle Tree

The Bitcoin protocol stores the history of transactions in a Merkle Tree, a binary tree of transaction hashes. This allows for easy traversal throughout the tree and thus easy verification of transaction history within a single block.

To create the first block, the genesis block, a root of its Merkle Tree will have to be generated. We begin by updating the Unix time associated with LearnCoin. Unix time is measured in seconds since 1/1/1970. To get the current Unix time from the command line:

```
~$date +%s
1400539822
```

Place this value in the following locations:

/src/main.cpp Line 2033

```
block.nTime     = 1400539822;
```

/src/main.cpp Line 2039 This is the location for using the Testnet - we will not cover usage of Testnet but it can be useful if seeking to have a more controlled environment once the coin is established.

```
block.nTime     = 1400539822;
```

Typically one also changes Line 2021 to a news headline/other message that corresponds to the date:

```
const char* pszTimestamp = "First day of Project";
```

Now that we have made our changes, we upload them to the cloud using git commands:

```
~/NewCoin$ git add -A *
~/NewCoin$ git commit -m “changesportcointime”
~/NewCoin$ git push origin master
```

To build the Merkle root

```
~/NewCoin$ cd src
~/NewCoin/src$ make -f makefile.unix
```

We have just built a command-line only version of our coin. To generate the Merkle root, we run using the following command:

```
~/NewCoin/src$ ./NewCoin
```

And we get the following error message:

```
NewCoin: main.cpp:2047: bool LoadBlockIndex(bool): Assertion
`block.hashMerkleRoot ==
uint256("0xef7e256a2cf2d38576225af2775a1e8160928a0c78245ab3615615d1ff
16870e")' failed.
Aborted (core dumped)
```

Our generated Merkle root can be found within NewCoin's application data directory in the debug log, however. To navigate to this file:

```
~/NewCoin/src$ cd
~$ cd .NewCoin
~$ gedit debug.log
```

One should see something like this:

NewCoin version v0.6.3.0-unk-beta (2014-05-08 14:44:24 -0400)
Startup time: 05/21/14 18:04:26
Default data directory /home/username/.NewCoin
Used data directory /home/username/.NewCoin
Bound to [::]:36329
Bound to 0.0.0.0:36329
Loading block index...
dbenv.open LogDir=/home/username/.NewCoin/database ErrorFile=/home/username/.NewC$
bc440ed918d9deb0e8c1dc135293e8509dc1e062d4a532ea1152dc82c181c16a
14dc23f54de47df797172d66531e7a115b782e65f69dda7bf5a4e98fac0ae086
ef7e256a2cf2d38576225af2775a1e8160928a0c78526ab3615615d1ff16870e

The last line contains the valid Merkle root:

ef7e256a2cf2d38576225af2775a1e8160928a0c78526ab3615615d1ff16870e

Insert this value in /src/main.cpp Line 2047

```
assert(block.hashMerkleRoot ==
uint256("0xef7e256a2cf2d38576225af2775a1e8160928a0c78526ab3615615d1ff
16870e"));
```

Be sure to keep the “ marks and the 0x in front of the Merkle root intact.

Genesis Block

To begin creating the Genesis Block, ensure that the following line in main.cpp matches:

/src/main.cpp Line 2050
```
if (true && block.GetHash() != hashGenesisBlock)
```

By setting the first part of the and statement to true, the program will attempt to mine the genesis block when it runs as block.getHash() will not equal hashGenesisBlock at this point in time.

Now, recompile the code with the above changes:

```
~/NewCoin/src$ make -f makefile.unix
```

And run again

```
~/NewCoin/src$ ./NewCoin
```
After some time another error message will be displayed. At this point the genesis block has been successfully mined. Once again its information can be located within the .LearnCoin/debug.log folder:

block.nTime = 1400539822
block.nNonce = 2087447301
block.GetHash =
21d36dc831c042ce8260d01777140c0ed5f667d5c32cd79f7147431ba46fe368
CBlock(hash=21d36dc831c042ce8260, PoW=00000211c28c2ba5ea6f, ver=1, hashPrevBlock=0000000000000000000, hashMerkleRoot=ef7e256a2c, nTime=1400539822, nBits=1e0ffff0, nNonce=2087447301, vtx=1)
  CTransaction(hash=ef7e256a2c, ver=1, vin.size=1, vout.size=1, nLockTime=0)
        CTxIn(COutPoint(0000000000, -1), coinbase 04ffff001d0104144669727374206461792066620507266a656374)
        CTxOut(error)
  vMerkleTree: ef7e256a2c

For reference, I suggest inserting this information within the block comment on line 2009 of main.cpp.

Replace the following values:

/src/main.cpp Line 32

```
uint256
hashGenesisBlock("0x21d36dc831c042ce8260d01777140c0ed5f667d5c32cd79f7
147431ba46fe368");
```

/src/main.cpp Line 2035

```
block.nNonce = 2087447301;
```

/src/main.cpp Line 2050

```
if (false && block.GetHash() != hashGenesisBlock)
```

/src/checkpoints.cpp Line 27

```
      (          0,
uint256("0x21d36dc831c042ce8260d01777140c0ed5f667d5c32cd79f7147431ba4
6fe368"))
```

The checkpoints file ensures that users running the coin software are on the same blockchain. Currently the only checkpoint that needs to be added is the genesis block, block 0. Later on one can add more checkpoints for different blocks.

Once again, after these modifications, we update our online repository:

```
~/NewCoin$ git add -A *
~/NewCoin$ git commit -m “changesmerklegenesis”
~/NewCoin$ git push origin master
```

And rebuild

```
~/NewCoin/src$ make -f makefile.unix
```

On a second computer, clone the repository

```
~$ git clone https://github.com/yourusername/NewCoin.git
```

And build as before.

Mining and Transactions

On both computers, navigate to the ~/.NewCoin/ directory, and create a new file called NewCoin.conf containing the following information:

rpcuser=someusername
rpcpassword=somepassword
addnode=<the ip address of the other computer>

One can use either local IP addresses or public ones. This will connect the two computers at start. The username and password should be unique to allow for differentiation.

On both computers, run the NewCoin program

```
~/NewCoin/src$ ./NewCoin &
```

The & will allow us to run other commands as NewCoin will run in the background. A list of commands can be found here: https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_Calls_list
In the meantime, use the getinfo command:

```
~/NewCoin/src$ ./NewCoin getinfo
{
      "version" : 60300,
      "protocolversion" : 60001,
      "walletversion" : 60000,
      "balance" : 0,
      "blocks" : 0,
      "connections" : 1,
      "proxy" : "",
      "difficulty" : 0.00024414,
      "testnet" : false,
      "keypoololdest" : 1400530204,
      "keypoolsize" : 104,
      "paytxfee" : 0.00000000,
      "mininput" : 0.00010000,
      "errors" : ""
}
```

The output on both computers should match and verify that each has connected to the other.

To begin mining

```
~/NewCoin/src$ ./NewCoin setgenerate true 4
```

This will cause the computer to begin mining NewCoin using 4 threads of its CPU.

```
~/NewCoin/src$ ./NewCoin getmininginfo
{
      "blocks" : 0,
      "currentblocksize" : 1000,
      "currentblocktx" : 0,
      "difficulty" : 0.00024414,
      "errors" : "",
      "generate" : true,
      "genproclimit" : 4,
      "hashespersec" : 7159,
      "networkhashps" : 18068,
      "pooledtx" : 0,
      "testnet" : false
}
```

will return information about the how the computer is currently mining.

Coins earned from mining are automatically rewarded to an empty String account (“”) in one’s wallet (stored as a wallet.dat file in the application data directory).

To view the list of accounts

```
~/NewCoin/src$ ./NewCoin listaccounts
{
      "" : 500.40000000,
      "test1" : 12.90000000
}
```

In this example, the empty String account has 500.4 coins, while another account in the wallet test1 has 12.9. To add accounts to the wallet

```
~/NewCoin/src$ ./NewCoin getnewaddress <accountname>
8mVLPUs4BqQciJKxvc5oBZUaKcyqjto2wW
```

will return a NewCoin address and associate it with the specified account name.

To move coins from the “” account to another

```
~/NewCoin/src$ ./NewCoin move “” <accountname> <amount>
```

will return true if successful.

To send to another address

```
~/NewCoin/src$ ./NewCoin sendtoaddress <address> <amount>
```

will return the transaction ID for the transaction.

A look through the debug.log file during these processes can glean much more information. In this example, 20 coins are to be sent from Brahms to Beethoven. In this version of LearnCoin each block rewards 4 coins.

Sender (Brahms):
ThreadRPCServer method=sendfrom **//Call to send 20 LearnCoin**
SelectCoins() best subset: 4.00 4.00 4.00 4.00 4.00 total 20.00 **//Why groups of 4 ea.? Think of it like having a $4 bill (assuming one of those exists). When blocks are created the creator is rewarded with a $4 "bill" essentially. Remember though that as these are mere digital constructs bills can be changed on demand.**
SelectCoins() best subset: 4.10 4.00 4.00 4.00 4.00 total 20.10 **//.1 LearnCoin transaction fee**
CommitTransaction:
CTransaction(hash=0c30d8cde6, ver=1, vin.size=5, vout.size=1, nLockTime=0)
    CTxIn(COutPoint(59a19546ae, 0), scriptSig=3046022100d30d5c8dc2205b)
    CTxIn(COutPoint(830c3fe0e2, 0), scriptSig=3046022100d21fc0d80935ec)
    CTxIn(COutPoint(40edc4fcab, 0), scriptSig=3044022061708eadcd1eadb6)
    CTxIn(COutPoint(541f7666c0, 0), scriptSig=304502210091bebd3b364e57)
    CTxIn(COutPoint(9246593db1, 0), scriptSig=3046022100b3f172582af82b)
    CTxOut(nValue=20.00000000, scriptPubKey=OP_DUP OP_HASH160 1cdc63629ce8)
AddToWallet 0c30d8cde6 new
WalletUpdateSpent found spent coin 4.00bc
59a19546aeedaab6e3112d430cc3d49531bb3ab0d5e0e8a7407fc59a0f11431a
WalletUpdateSpent found spent coin 4.10bc
830c3fe0e298dad57e1714cd9ea95dc24ea4dc643190cca45015605d3bb75821
WalletUpdateSpent found spent coin 4.00bc
40edc4fcabd6edd25b3b7ac8ab55950f88afa0159cbc597e26eedad8c016ccd7
WalletUpdateSpent found spent coin 4.00bc
541f7666c06dd6bf933fa22618d50fa661e9a4fae7a0633f62504a089d167ce6
WalletUpdateSpent found spent coin 4.00bc
9246593db13ec4d08bc59174d7b05cb67a491f465e3527b49f71a51f6cd4e4ee
CTxMemPool::accept() : accepted 0c30d8cde6 (poolsz 1)
Relaying wtx 0c30d8cde6
received getdata for: tx 0c30d8cde68c12c381cf **//Transaction ID - only valid for wallets that are affected by the transaction**
Flushing wallet.dat **//Updating wallet information**

Flushed wallet.dat 35ms
received block 07fe58566451c393ac5d **//Another node found the block, receive block info**
AddToWallet 0c30d8cde6 update **//Updates with another confirm on the transaction**
SetBestChain: new best=07fe58566451c393ac5d height=178 work=187697968 date=05/15/14 19$ **//New longest chain**
ProcessBlock: ACCEPTED

Recipient (Beethoven):
CTxMemPool::accept() : accepted 0c30d8cde6 (poolsz 1) **//Transaction information**
AddToWallet 0c30d8cde6 new **//Adding values to wallet**
received getdata for: tx 0c30d8cde68c12c381cf **/Adding values to wallet**
Flushing wallet.dat **//Updating wallet information**
Flushed wallet.dat 51ms
received block 07fe58566451c393ac5d **//Another node found the block, receive block info**
AddToWallet 0c30d8cde6 update **//Updates with another confirm on the transaction**
SetBestChain: new best=07fe58566451c393ac5d height=178 work=187697968 date=05/15/14 19$ **//New longest chain**
ProcessBlock: ACCEPTED

In this case, a third node, Bach, was the one who solved the problem first and added the transaction to the block and thus the block to the block chain.

Block finder (Bach):
CTxMemPool::accept() : accepted 0c30d8cde6 (poolsz 1) **//Got transaction information**
Running BitcoinMiner with 2 transactions in block
BitcoinMiner:
proof-of-work found **//Found proof-of-work to create block**
  hash: 0000088a3855757a24c6cc11a1b5a536022ad01c124e471e3e8376c44da7f93f
target: 00000ffff0000000000000000000000000000000000000000000000000000000
CBlock(hash=4893f6f9a5a258ac0f22, PoW=0000088a3855757a24c6, ver=1, hashPr$
  CTransaction(hash=a08fb63ea6, ver=1, vin.size=1, vout.size=1, nLockTime$
        CTxIn(COutPoint(0000000000, -1), coinbase 042c1275530102062f503253482$
        CTxOut(nValue=4.10000000, scriptPubKey=02588922471cf99120852301090984)
  CTransaction(hash=0c30d8cde6, ver=1, vin.size=5, vout.size=1, nLockTime$
        CTxIn(COutPoint(59a19546ae, 0), scriptSig=3046022100d30d5c8dc2205b)
        CTxIn(COutPoint(830c3fe0e2, 0), scriptSig=3046022100d21fc0d80935ec)
        CTxIn(COutPoint(40edc4fcab, 0), scriptSig=3044022061708eadcd1eadb6)
        CTxIn(COutPoint(541f7666c0, 0), scriptSig=304502210091bebd3b364e57)
        CTxIn(COutPoint(9246593db1, 0), scriptSig=3046022100b3f172582af82b)
        CTxOut(nValue=20.00000000, scriptPubKey=OP_DUP OP_HASH160 1cdc63629ce$
  vMerkleTree: a08fb63ea6 0c30d8cde6 fa8e4d3b68
generated 4.10 **//4.0 + .10 transaction fee "reward"**
keypool keep 62

AddToWallet a08fb63ea6 new
SetBestChain: new best=4893f6f9a5a258ac0f22 height=167 work=176163456 $
ProcessBlock: ACCEPTED

Further Reading

Users who wish to have their coin available for mass public use should set up a “seed node” which will have many connections to other nodes to facilitate peer to peer connections.

Though now slightly outdated, another resource available regarding the creation of one’s own cryptocurrency can be found here:
http://dogecoin.ga/how_to_create_scrypt_based_altcoins.html
This resource was intended as an update to supercede and accompany the above instructional. Much of my process parallels the above resource, except skipping functions such as testnet and GUI wallets instead focusing more on creating transactions.

Additionally, the original Bitcoin paper published by Satoshi Nakamoto gives a more in depth explanation of the protocol: https://bitcoin.org/bitcoin.pdf
The Bitcoin wiki also offers many technical articles about the protocol:
https://en.bitcoin.it/wiki/Main_Page

Acknowledgements

I would like to acknowledge Professor Mike Lin of Cleveland State University for his support, as well as a fellow student Savva Madar for helping me get introduced with Bitcoin and cryptocurrencies. Without both of their assistance this project would not have been possible at all.