How To Clone Scrypt Based Altcoins for Fun and Profit

Wait a second, why would you want to give out the secrets?!? Because its not a secret anymore and besides, why shouldn't everyone and their neighbors be able to create a plethora of these useless yet exciting math bits? The information in this article took me a few weeks to compile and what works for me is not guaranteed to work for you. Please use this guide as a starting point to learn a bit about C programming and compiling software.

I will NOT do tech support–just because you can't get something to work doesn't entitle you to bother me about it. Go read, dig, and read some more. Nearly everything in this guide is posted in some form or another on bitcointalk.org's altcoin forum. The rest of it I meticulously tracked down through trial and error and a healthy dose of Googling. Things are meant to break, you'll need to figure out why and make it work. By the end of this guide you should have a working coin, p2p(irc) network, and clients for Linux (easy), Mac (a bit harder), and Windows (ugh).

What do I need?

- Source Code for a Scrypt Coin
- Working knowledge of and access to Linux or Mac command line–I'm not going to show Cygwin, but its similar.
- 2 or more computers or virtual machines–I will be using my laptop and a Debian Wheezy based VPS.
- Text Editing Software–I'm using TextWrangler for Mac, but Netbeans, EmeraldEditor, or nano will work fine.
- Time and Patience…

Happy Fiddling!

Github, Source Code, and Linux

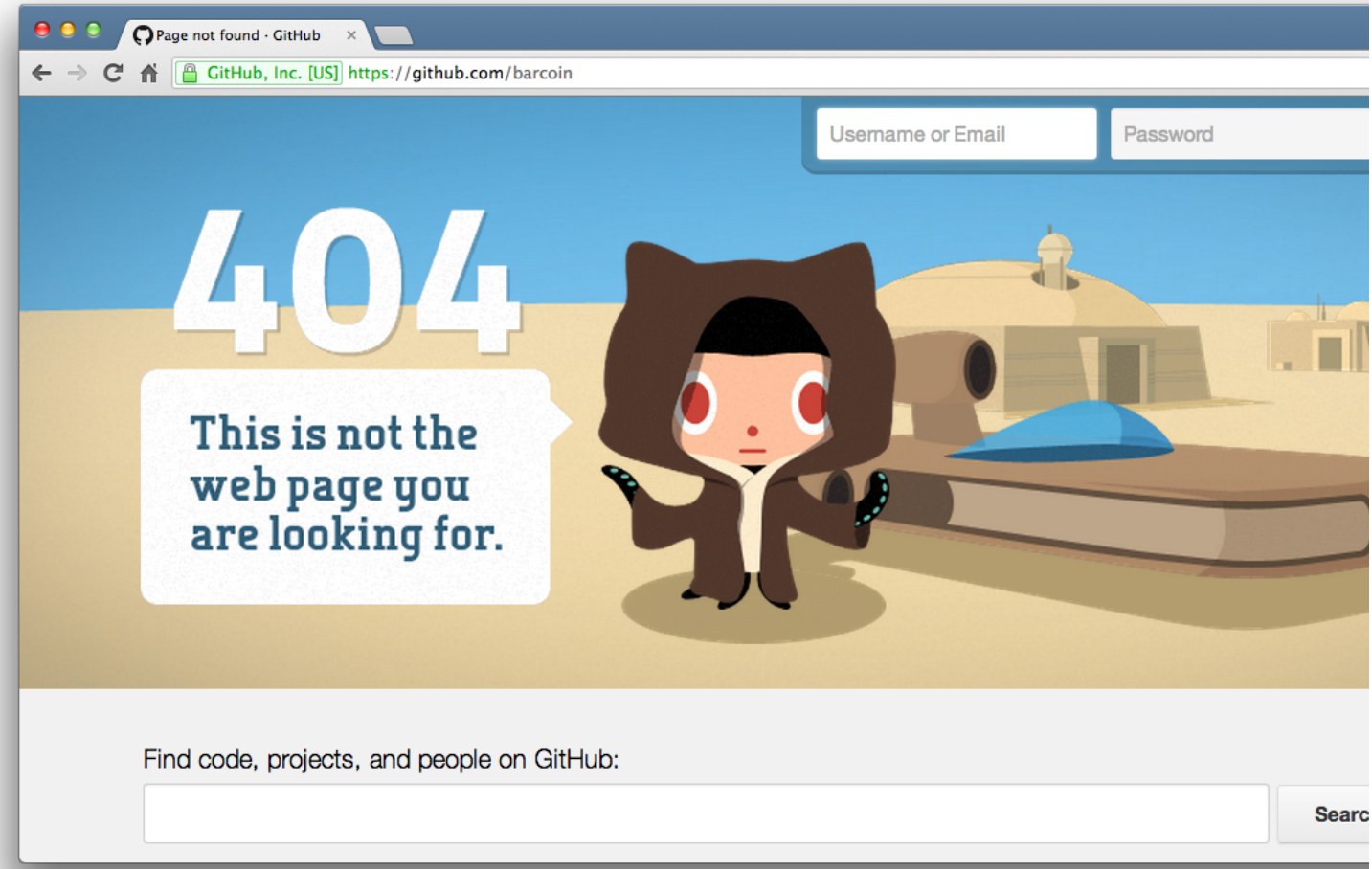
Source Code

First things first. You'll need some source code. Since I doubt you can write it from scratch (I couldn't), you should be happy to know there are a bazillion different options in the scrypt-coin family for you to clone and alter. My first coins were based on the most excellent research coin, SmallChange, by lightenup. His git diff output: https://github.com/bfroemel/smallchange/commit/947a0fafd8d033f6f0960c4ff0748f76a3d58326 [https://github.com/bfroemel/smallchange/commit/947a0fafd8d033f6f0960c4ff0748f76a3d58326] is nearly enough information to completely create your own alt-coin and as such should be lauded. Yes, I realize his code is simply the Litecoin source with cut and paste changes, but hey–that's what we're working on here and he added some excellent comments throughout the code.

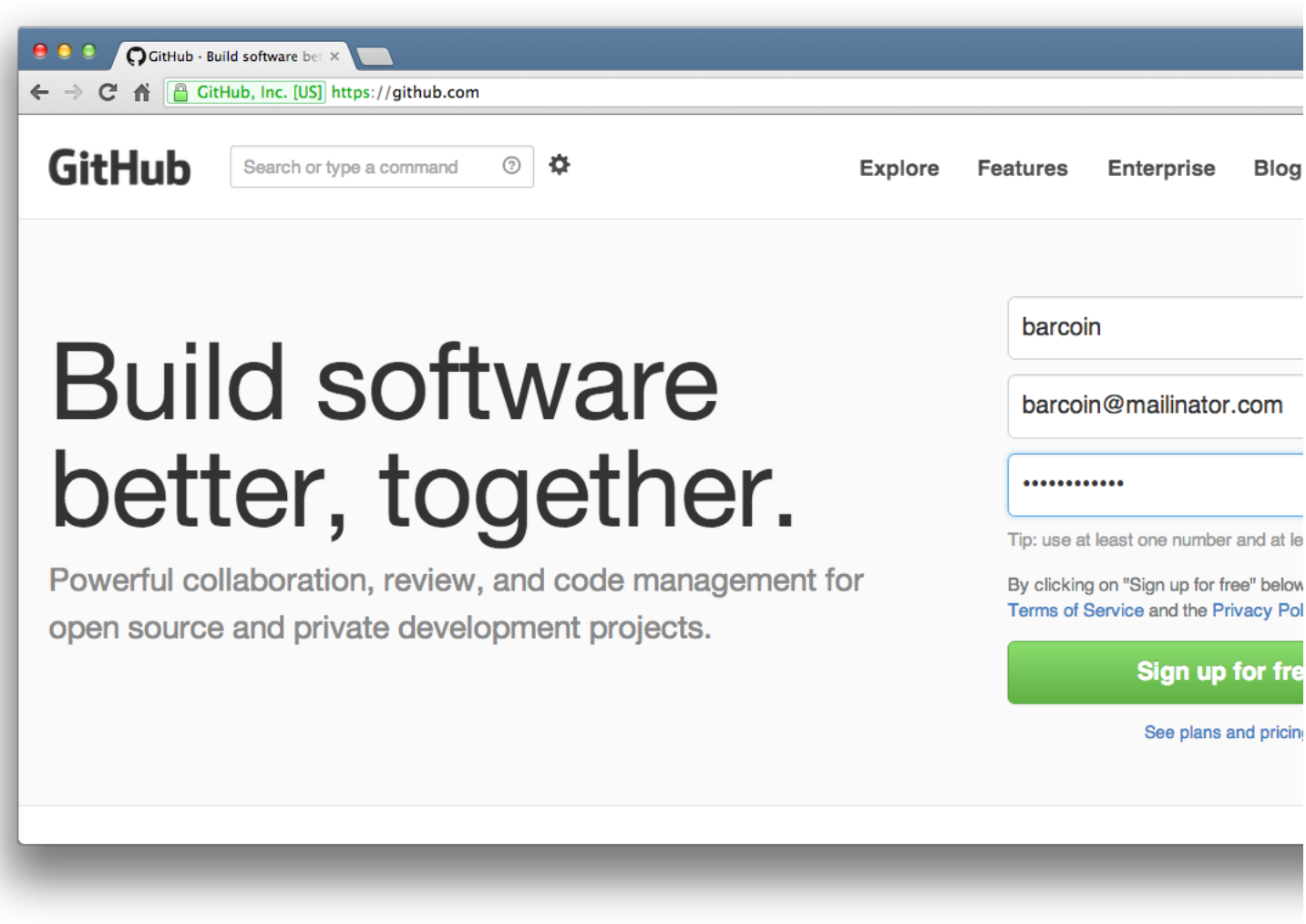
For the purposes of this tutorial and to preserve a “clean” copy of the SMC code, I have created foocoin: https://github.com/foocoin/foocoin [https://github.com/foocoin/foocoin] This guide will show you how to turn 'foo'coin in to 'bar'coin and you can take it from there. I've already changed enough to make this coin compile-able if you follow this guide. If you'd prefer to start with the original SmallChange code, it is here: https://github.com/bfroemel/smallchange.git [https://github.com/bfroemel/smallchange.git] or you could use the Litecoin, Franko, YAC, CHN, MIN, whatever source–we'll change enough of it to make it work.

Set up a Github account

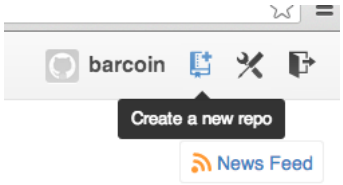
Now would be a good time to come up with a brilliantly unique name for your new coin. You can visit https://github.com/insertcoinnamehere [https://github.com/insertcoinnamehere] and check to see if your coin's name is available. If it's not, just add -project or something witty and credibility boosting, maybe -nonprofit or -consortium would give your coin some gusto:



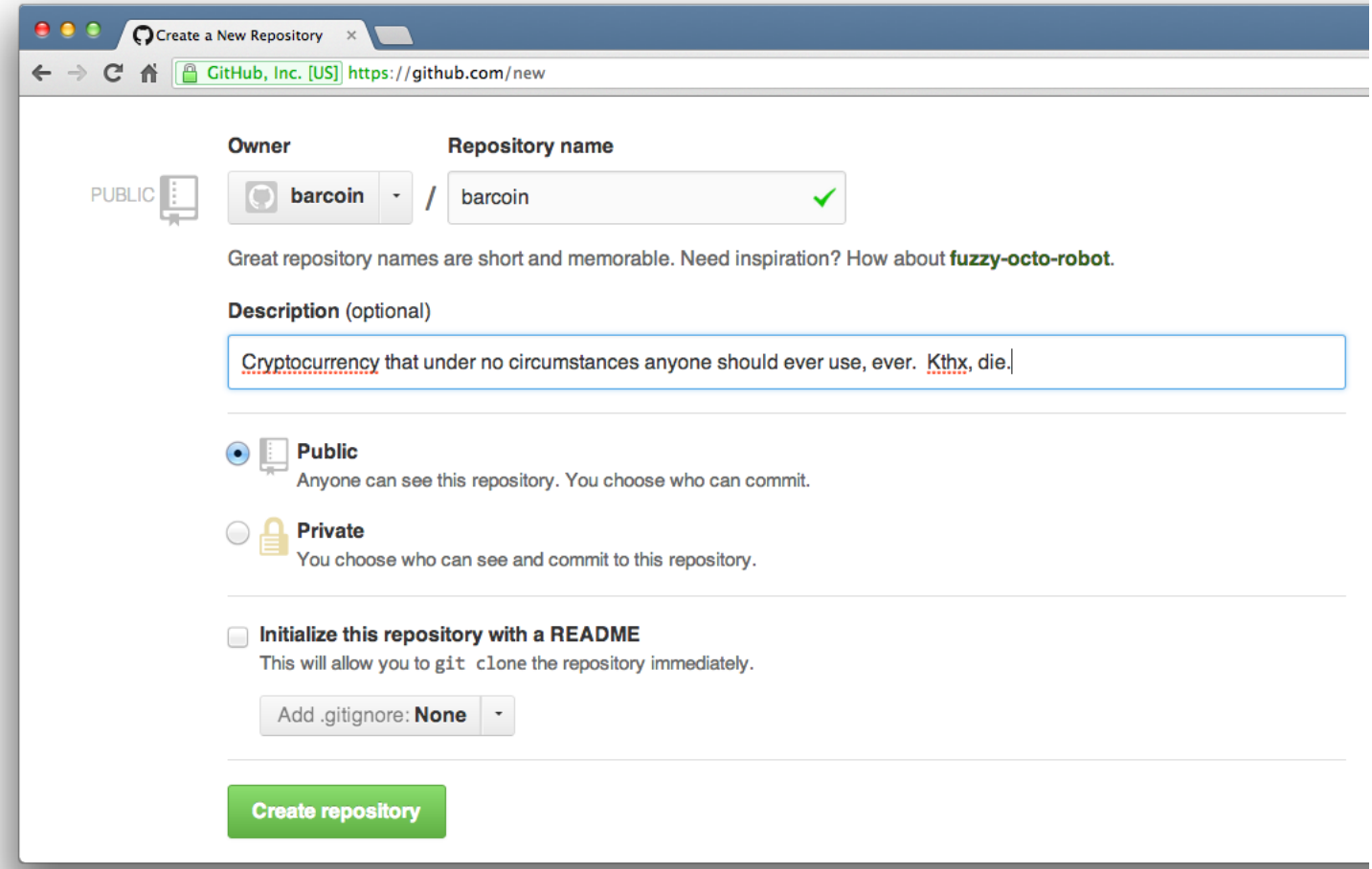
Hey, look, barcoin isn't taken–we'd better snatch it right up! Simply go to https://github.com [https://github.com] and fill-in the blanks:



Now, lets make one of those sexy source code URL's that everyone can git pull from. What's git pull? Don't worry, you'll get the hang of it–look, you're learning new things already. In the upper right hand corner next to your username, click “Create Repository:”



Now fill in the blanks:



Click the green button and voila, you have Github. Now take note of this information:

Create a new repository on the command line

```
touch README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/barcoin/barcoin.git
git push -u origin master
```

Because we're going to come back to that. Better yet, Bookmark it in your browser and open a new tab.

Linux, Mac, Cygwin

Yes, you need one of them. For this tutorial, I will be using a MacBook with OSX 10.8.something and a Debian Wheezy VPS. I suppose you can use Cygwin, but I prefer the *nix's to Windows any day and you may as well learn one more thing too, right? Shoot, if you need a good LiveCD with the dependencies already built in that is set up to be run in a VMWare session, try CDEbian. Otherwise this guide uses Debian which means most mainstream clones should work (i.e.: Ubuntu, XUbuntu, Mint).

Setting up a PC or VM with Linux or OSX is outside the scope of this tutorial, but suffice to say I can vouch for VMWare and its ability to run multiple VMs simultaneously and its ability to virtualize OSX 10.6.x. You will need to install some dependencies on which ever OS you choose to be able to build the daemon and -Qt wallet.

Dependencies for OSX

The easiest way I've found to install dependencies on OSX is to use MacPorts or Homebrew. Personally, I like MacPorts better than Homebrew, but its simply because MacPorts installs in /opt (where I think it should go) and because MacPorts offers universal builds by default. If you prefer Homebrew, the formulas are the same, but I'd recommend building with the –32-bit flag. For the purpose of this tutorial, we're going to use MacPorts.

One needs to install the following dependencies:

- boost (C++ libraries)
- db48 (Berkeley DB 4.8)
- qt4-mac (Open Source QT 4.8.4, includes qmake)
- openssl (ssl dev libraries)
- git (to move source back and forth to the repository)
- miniupnpc (UPNP dev libraries, optional–honestly I say skip this crap)

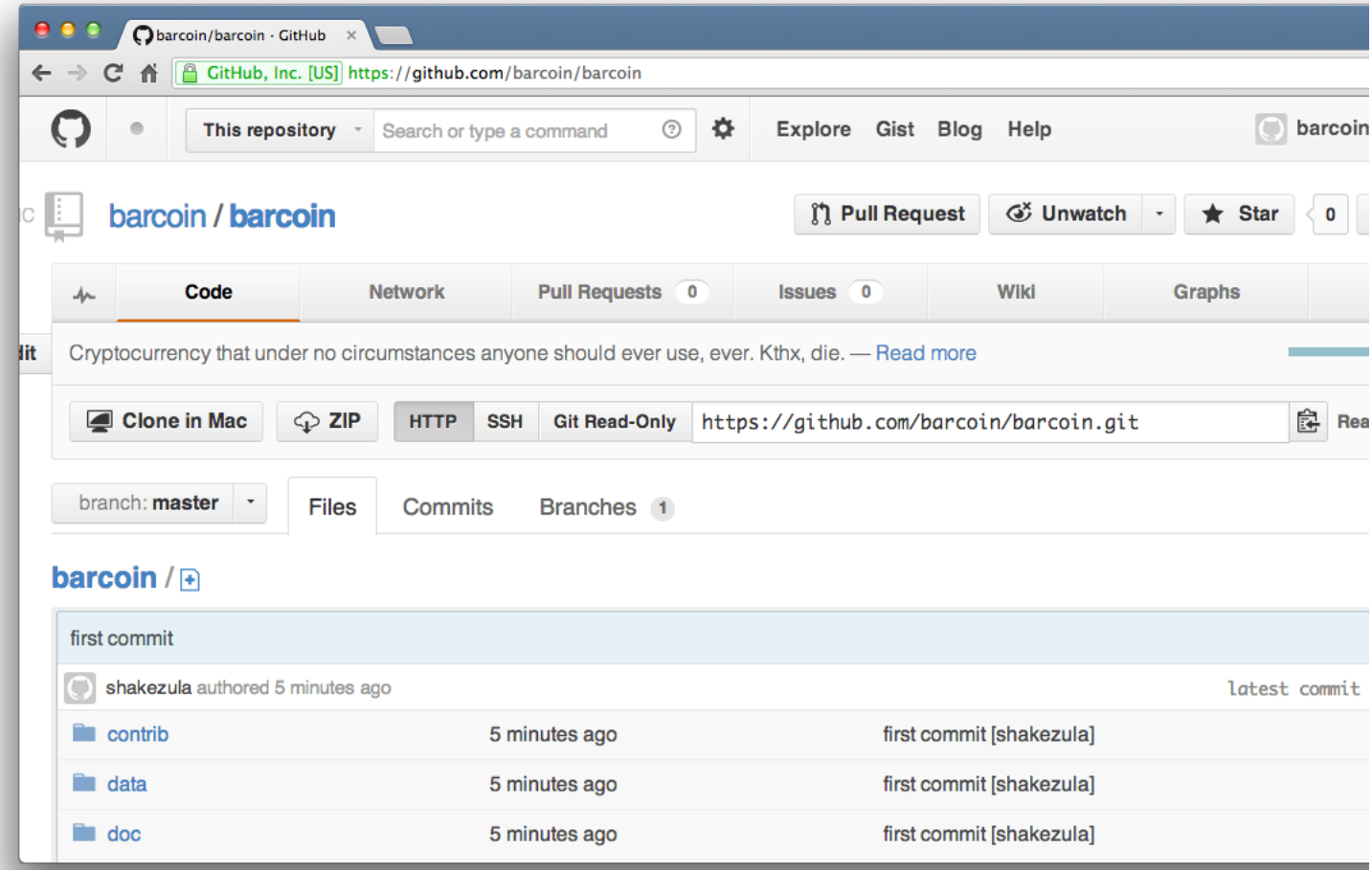
After installation of the basic MacPorts for your version of OSX, this can be accomplished with this command:

```
%sudo port install boost db48 qt4-mac openssl miniupnpc git
```

Once all of the dependencies are built and installed, the next step is to clone the source from git. In this example, I will be cloning foocoin, rename it, re-git initialize it, and push the initial copy out to Github to ensure git is working:

```
%git clone https://github.com/foocoin/foocoin.git
cloning in to foocoin
%mv foocoin barcoin
%cd barcoin
%rm -rf .git
%git init
initializing git repository in ~/barcoin
%git add -A *
%git commit -m "first commit"
%git remote add origin https://github.com/barcoin/barcoin.git
%git push -u origin master
username for git@github.com: barcoin
password for barcoin@github.com: **********
```

Now what did we just do? We cloned the existing source, deleted its link to git, reinitialized the folder for Github, added all the existing folders and files in to the repository, committed our changes (made them permanent and put them in the “Master” branch, renamed it in to our new *bigger *better *faster coin, set it back up to link to Github–but to the *new* coin's account, and pushed a copy out to Github. Now if you go and look at your page it should look like so:



Oh, look at all that fresh source code just awaiting to be tweaked.

Dependencies for Linux

On Debian based Linux, dependencies and build requirements can be installed in a single command like so:

```
$sudo apt-get install build-esssential libboost-dev libdb48-dev libopenssl-dev libdb++-dev git qt-sdk
```

This will install all the needed packages as apt is very smart. Once that's complete, the same bits as above should be applied:

```
$git clone https://github.com/foocoin/foocoin.git
cloning in to foocoin
$mv foocoin barcoin
$cd barcoin
$rm -rf .git
$git init
initializing git repository in ~/barcoin
$git add -A *
$git commit -m "first commit"
$git remote add origin https://github.com/barcoin/barcoin.git
$git push -u origin master
username for git@github.com: barcoin
password for barcoin@github.com: **********
```

Dependencies for Windows

Ugh, I knew you'd ask. Windows is tricker than it should be when it comes to building with the GNU toolchain. First, you'll need mingw32 installed and dependencies built by hand for each of the listed items above. You'll also need to customize your -qt.pro file with the location of those dependencies. To simplify this, I've already compiled and assembled the needed dependencies in to a nice Github repository for you. If either downloaded and extracted as c:\deps or git cloned to C:\, this pacakge: https://github.com/foocoin/deps.git [https://github.com/foocoin/deps.git] will give you everything you need to build foo(bar)coin using the source you've already got. More about building the long way when we get to the Windows client compilation bit a little further along in the project.

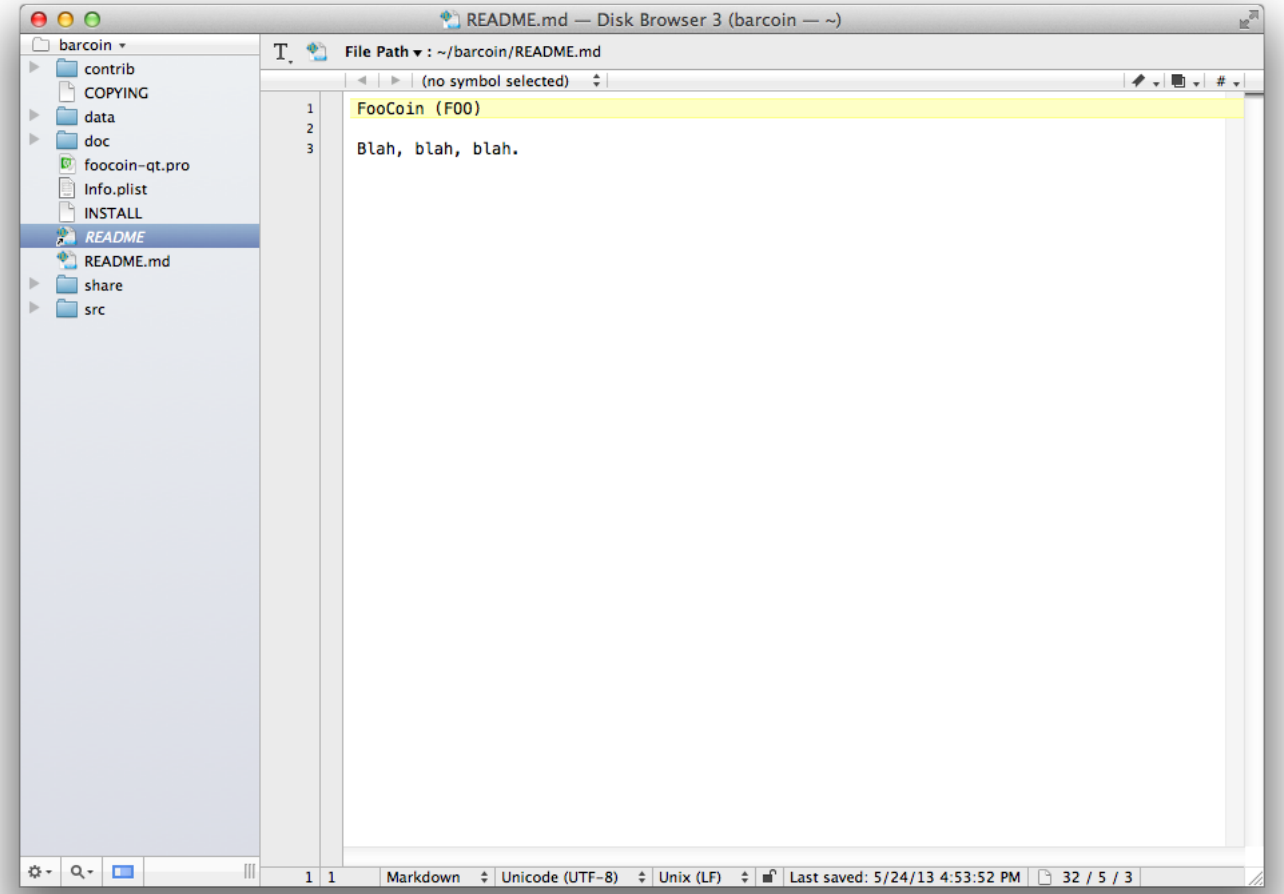
Now you're ready to Cut and Paste!

Search and Replace

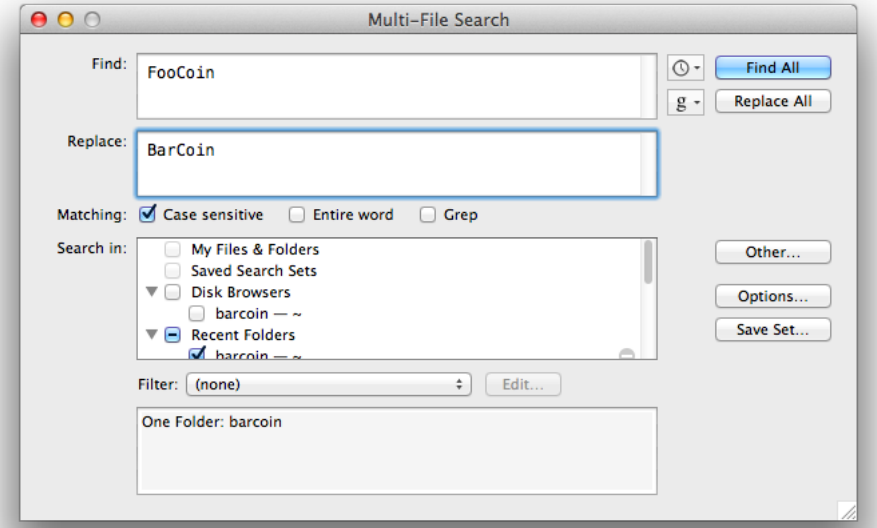
Ahh, now we've come to the creative part. The bit where you change the things you want to change to make your coin yours. As this is a *cloning* tutorial, I am not going to focus on the intricacies of programming (I'm not qualified). I'm simply going to show you where to make the changes you need to make to get a coin up and running. For this step, I really prefer TextWrangler on my Mac. It allows for multiple file searching and replacing which makes this portion of the process go quite quickly. If you're going to set up a VM to build -Qt wallets for Mac anyway, you should/could simply install the dependencies above and build within OSX completely. TextWrangler is free.

Names

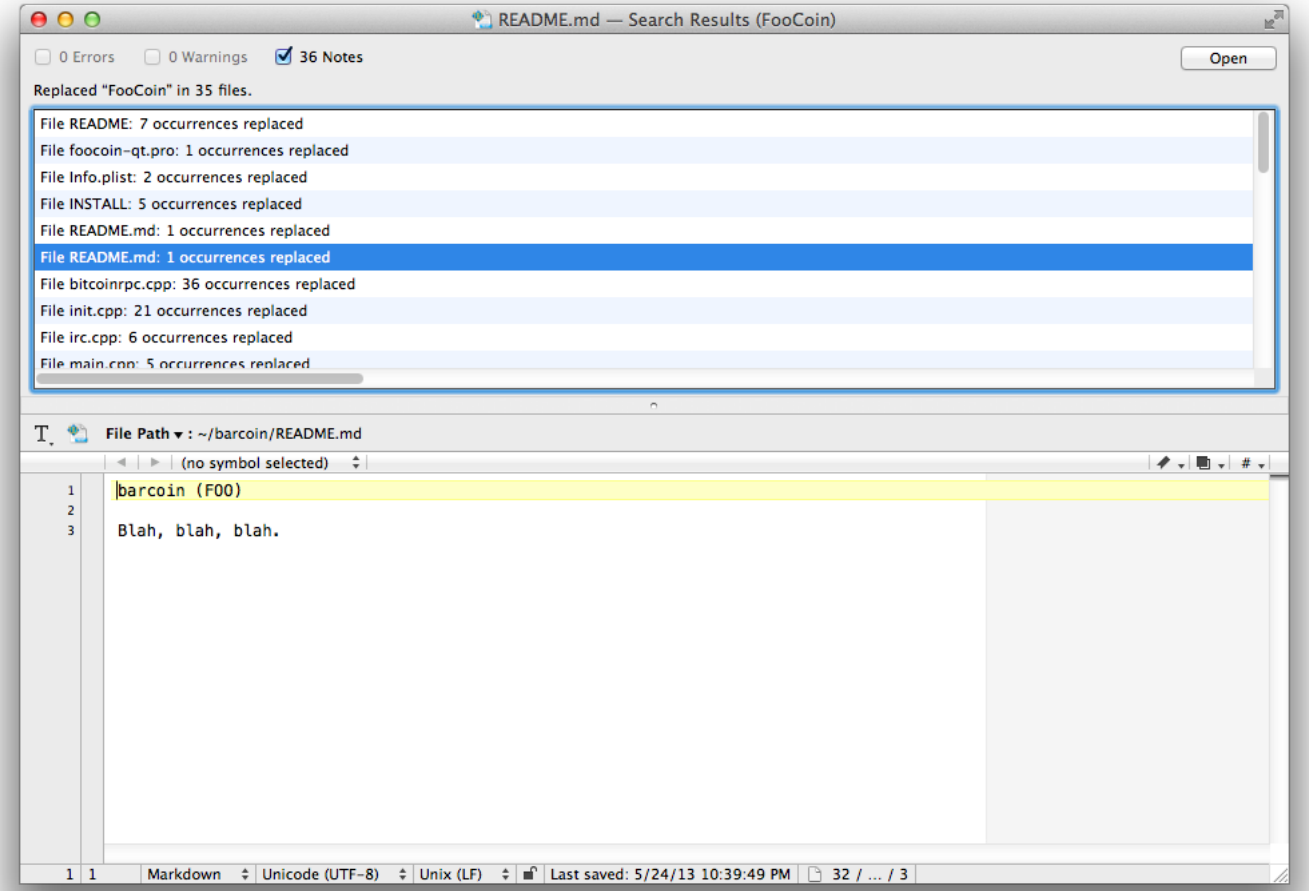
TextWrangler will allow you to open an entire folder of files. Just open the program and choose File, the Open, highlight the “barcoin” folder and click Open:



Ahh, nice, all of the code in one easy to use interface. Be aware, that editing these files most certainly can be done via nano or Netbeans, or whatever other text editor, even Notepad I suppose. I just like this one, 'cuz of this next feature. Now we need to replace all instances of “FooCoin, foocoin, and FOOCOIN” in our source with “BarCoin, barcoin, and BARCOIN.” Note the 3 different case settings–most code has all three in it. To change this in TextWrangler, choose Search, then Multi File Search and select the “barcoin” directory:



Do this for all three case settings, or if you prefer e.e.cummings style, replace them all without the “Case Sensitive” box checked in one fail swoop. TextWrangler will show you the whole list of changed files and allow you to browse the changes once completed:



You will also want to replace all instances of “FOO” with “BAR.” This is the 3 letter designation for your coin, like BTC or PPC. Finally, you will need to manually change the name of foocoin-qt.pro in the main source folder. Hey…this is starting to come together, no?

Ports and Network Changes

Ok, now we need to give the new coin a unique port range to use. You'll need two ports, one for RPC connections (for miners to connect to) and one for P2P Connections. You can find a good list of reserved ports here: http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers [http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers] Most any ports will work assuming they are 1: Over port 1024 and 2: not used by something else. I'd suggest something in the high numbers, good examples include 56679 and 56680 or 12365 and 12366.

For this example we're going to use 55883 for RPC and 55884 for P2P. In the foocoin sources, these ports are already set, so go ahead and modify them using your text editor of choice.

Change the RPC/P2P Port in the following files:

src/bitcoinrpc.cpp: (RPC PORT)

```
LINE 2893:  ip::tcp::endpoint endpoint(bindAddress, GetArg("-rpcport", 55883));
LINE 3169:  if (!d.connect(GetArg("-rpcconnect", "127.0.0.1"), GetArg("-rpcport", "55883")))
```

src/init.cpp: (P2P PORT + Testnet Port)

```
LINE 235 "  -port= " + _("Listen for connections on  (default: 55884 or testnet: 45884)") + "\n" +
```

You can set the testnet port to any other random port, but remember what you set it to.

src/init.cpp: (RPC PORT)

```
LINE 271 "  -rpcport= " + _("Listen for JSON-RPC connections on  (default: 55883)") + "\n" +
```

src/protocol.h: (Testnet Port + P2P PORT)

```
LINE 22 return testnet ? 45883 : 55884;
```

You can also set an initial “seed node” or always on system that the new coin wallets coming online will check for additional addresses:

src/net.cpp:

```
LINE 1000  {"some website name", "somewebsite.org or ip x.x.x.x"},
```

Coins Per Block/Block Intervals/Max Number of Coins

These changes are also pretty simple. Change the following lines of code in the following files:

src/main.cpp: (Number of coins per block awarded)

```
LINE 831 int64 nSubsidy = 1 * COIN;
```

src/main.cpp: (How *should* blocks be found and how often difficulty retargets)

```
LINE 837 static const int64 nTargetSpacing = 120; // FooCoin: 2 minute blocks
LINE 836 static const int64 nTargetTimespan = 1 * 24 * 60 * 60; // FooCoin: 1 days
```

In this example, we want our coin to produce 1 coin blocks every 2 minutes and readjust difficulty once per day (1 day x 24 hours x 60 minutes x 60 seconds). You can adjust these, but know since this is a scrypt clone and we're not changing the starting difficulty this target rate will be skewed until the hash rate levels out on your new coin. This is tricky stuff and I don't quite understand all of it yet.

src/main.h: (Total number of Coins ever and Estimated # of Blocks per day)

```
LINE 43 static const int64 MAX_MONEY = 10000 * COIN; // maximum number of coins
LINE 550 return dPriority > COIN * 720 / 250; // 720 blocks found a day.
```

You'll need to do some math to figure out your blocks per day target based on how many coins you want to create over what timespan and how far apart your blocks are. I'm not doing all the work for you! This coin is set to give 1 coin blocks every 2 minutes, targeting 720 blocks per day through a maximum of 10,000 coins which means if mined hard, it will run out of coins in a week's time.

Address Starting Letter/Number

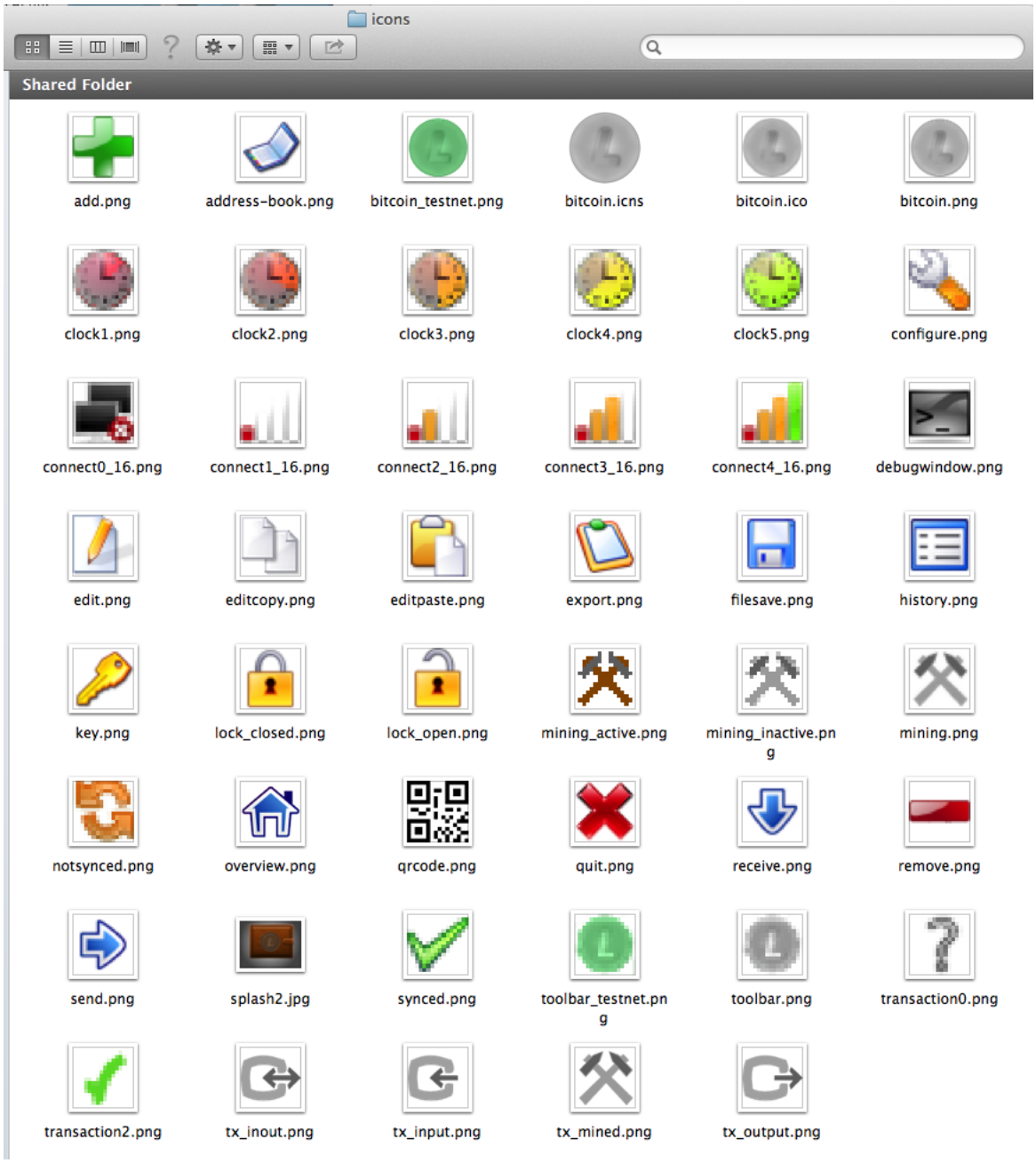
The first digit or letter of the new coin's address is determined by a base-58 code. You can see a list of all of the available options here: https://en.bitcoin.it/wiki/Base58Check_encoding [https://en.bitcoin.it/wiki/Base58Check_encoding] To change your coin's address edit this:

src/base58.h:

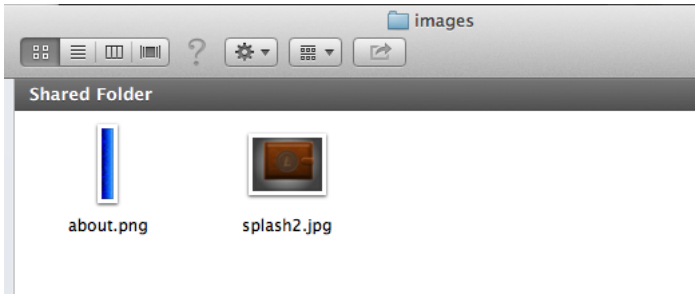
```
LINE 280 PUBKEY_ADDRESS = 38, //Set the address first bit here
```

Icons and Splash Images

You will find all of the supporting images and icons for the wallet in the **src/qt/res** folder. There are two folders with icons you should pay attention to:



and



Foo and bar coins both use the default Litecoin imagery. You should use an image editing program of your choice (I like Photoshop CS3, but GIMP is also nice) to edit the images. If you want rounded icons/images, use transparent .png files. Also, don't forget to generate an .ico (Windows/Linux) and an .icns (Mac) icon file for your program. A great website I like to use for this is here: http://iconverticons.com/ [http://iconverticons.com/]

Merkel Hash

The Merkel hash is the root of your coin's network. Its the hash that all of the blocks will be measured against and the basis for mining a genesis block. My methodology is to get a coin working on the testnet first and then the main network by building and testing in a staged progression. The Merkel hash is not actually the first thing you need to change though.

Epoch Time

Since Midnight UTC on New Years Day, 1970, Unix (or Epoch or POSIX) time has been used to coordinate various system calls and functions of Unix systems (and many others by default). Since this kind of time is simple seconds and doesn't account for leap seconds, its an easy way to calculate unique time-based values for programming. To that effect, the first thing one must change when building a new coin is the base time for the birth of the coin or the genesis of the coin.

This is set in two places in the code, one for the test net:

src/main.cpp:

```
LINE 2023 block.nTime    = 1300000000;
```

and one for the main net:

src/main.cpp:

```
LINE 2017 block.nTime    = 1300000000; //epochtime
```

You can get the current epoch time from: http://www.epochconverter.com/ [http://www.epochconverter.com/] or you can generate it from the command line of most *nix systems with this code:

```
$ date +%s
$ 1369590088
```

It is customary to also change this line of code to a headline from the day of coin creation in order to relate it to the block.nTime with some human-readable bit:

src/main.cpp:

```
LINE 2005 const char* pszTimestamp = "Traditionally one puts something timely here coinciding with the epoch";
```

Now, notice the other lines near the block.nTime, they are called block.nNonce. A 'nonce' is a unit of measurement that is unique and occurs after the nTime is set. The code uses nTime+nNonce to formulate and validate timestamps for blocks and transactions. This is a VERY rough overview of how this really works, but I hope it gives you an idea. We will come back to the nNonce in a moment when we mine a genesis block.

Generate a Merkel Hash

Thankfully, this forum post: https://bitcointalk.org/index.php?topic=189350.msg2035449#msg2035449 [https://bitcointalk.org/index.php?topic=189350.msg2035449#msg2035449] gives us a method to generate the Merkel Hash via the coin's test net feature. Right now would be a good time to do the following and copy your files out to your Github repository:

```
barcoin% git add -A *
barcoin% git commit -m "changes"
barcoin% git push origin master
```

Doesn't it feel good to know you're using Github like a pro?

First Build

Now that you have a fresh copy with all of your cut and pasting uploaded to Github, we're ready to build a copy of our command line only version of the coin:

```
barcoin% cd src/
barcoin/src% make -f makefile.osx USE_UPNP=- (or makefile.unix if you're on Linux/BSD/etc)
```

The code should build cleanly if you've only changed what you're supposed to and you ahve the right dependencies installed. You'll end up with a single executable called the name of your coin with a d on the end if you're on Linux (i.e. barcoin (osx/windows) barcoind (Linux). “Stripping” the extra code will create a smaller file if you so desire:

```
barcoin/src% strip barcoin (add the d on Linux, barcoind)
```

Now, we want to run barcoin from the command line using the -testnet switch like so:

```
barcoin/src% ./barcoin -testnet (add the d on Linux, ./barcoind)
```

It will immediately fail on first run, throwing out an error like so:

```
Assertion failed: (block.hashMerkleRoot == uint256("0x")), function LoadBlockIndex, file main.cpp, line 2031.
zsh: abort      ./barcoin
```

We now have a Merkel hash…wait, but where? Its is in your coin's “Application Data” directory. On Linux, that's in your home folder, then a .coinname like:

```
~./barcoin.
```

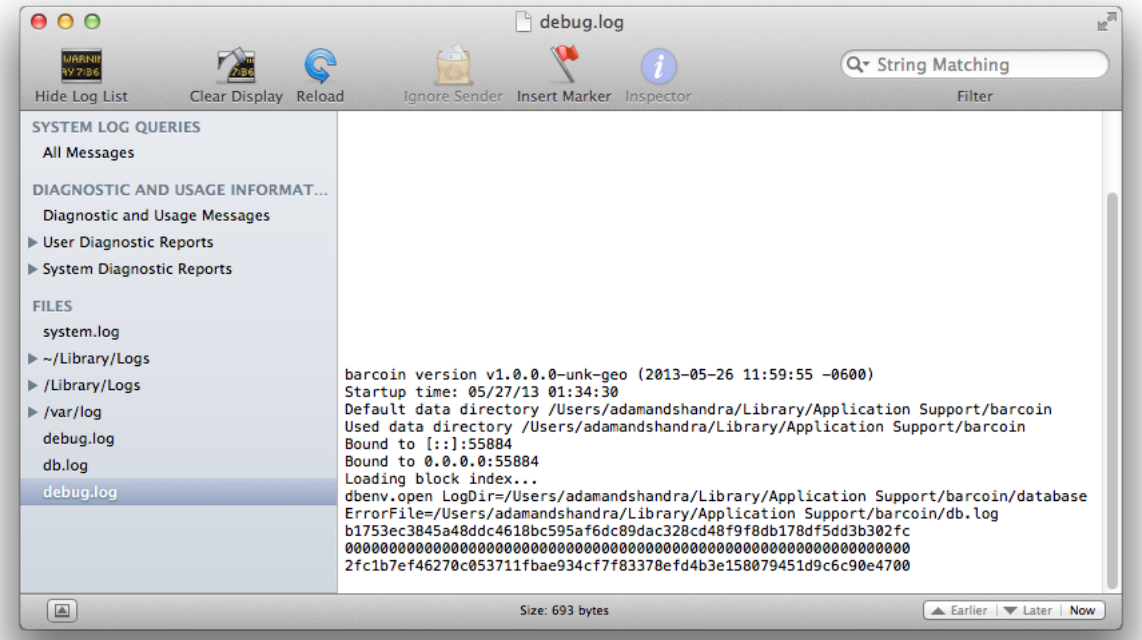
On OSX, it's going to be in your Library folder:

```
/Users/username/Library/Application Support/barcoin
```

If you want to see it graphically, hold the option button and click the Finder's Go menu, then choose Application Support and the barcoin folder. On Windows it will be in the Application Data Roaming folder:

```
c:\Users\username\AppData\Roaming\barcoin
```

In this folder you'll find a few files–this is also the folder you'll put your coin's .conf file when we're ready to mine it so remember how you got here. Now, open debug log and it will look like this:



Thanks to tyrion's amazingly helpful post, we can decipher this debug out put as so:

```
b1753ec3845a48ddc4618bc595af6dc89dac328cd48f9f8db178df5dd3b302fc Block hashed using the non-existent Merkel, based on the pzTimestamp from
0000000000000000000000000000000000000000000000000000000000000000 Genesis block, no good because all the nNonces are set to 0 in main.cpp
2fc1b7ef46270c053711fbae934cf7f83378efd4b3e158079451d9c6c90e4700 Valid Merkel Hash, generated using the epoch time in main.cpp
```

Now, take the valid Merkel Hash and insert it in to main.cpp:

src/main.cpp

```
LINE 2031 assert(block.hashMerkleRoot == uint256("0x2fc1b7ef46270c053711fbae934cf7f83378efd4b3e158079451d9c6c90e4700"));
```

Genesis Blocks

Dang, we're cooking with gas now eh? How does one mine a genesis block? Luckily the code is already in the source to do just that so don't fret. Who gets the initial coins? Well, no one really–there is a way to do it: https://bitcointalk.org/index.php?topic=189350.msg2038801#msg2038801 [https://bitcointalk.org/index.php?topic=189350.msg2038801#msg2038801] but personally I leave them to cyber space as a token of good karma to the bit gods at the church of development (FinShaggy, this means you, mate.)

Testnet Genesis Block

Ok, now you don't need to re-upload to Github just yet, because we need to generate genesis blocks for our network first. With the Merkel hash in place, this line:

src/main.cpp

```
LINE 2034 if (true && block.GetHash() != hashGenesisBlock)
```

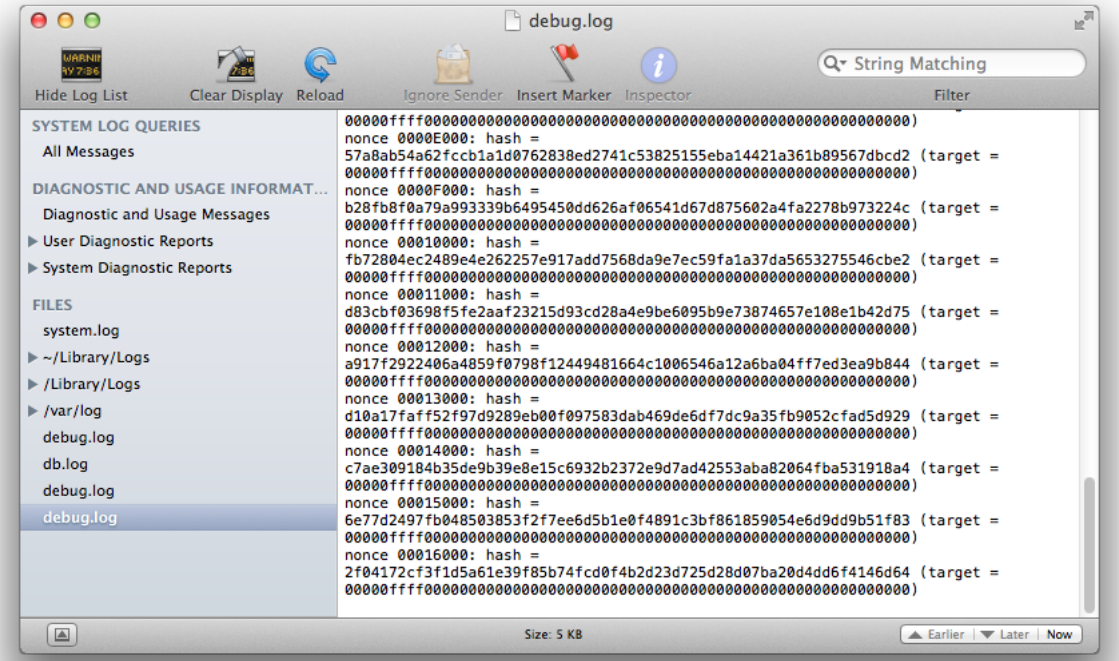
if set to true (as above) will mine a genesis block upon the next time the program is run–beginning with the nNonce in the code (0). Let's recompile the code with the new Merkel Hash:

```
barcoin/src$ make -f makefile.osx USE_UPNP=- (or .unix, whatever)
```

Recompilation should be pretty quick as most of the files have already been built. Once its done, start it again using this command:

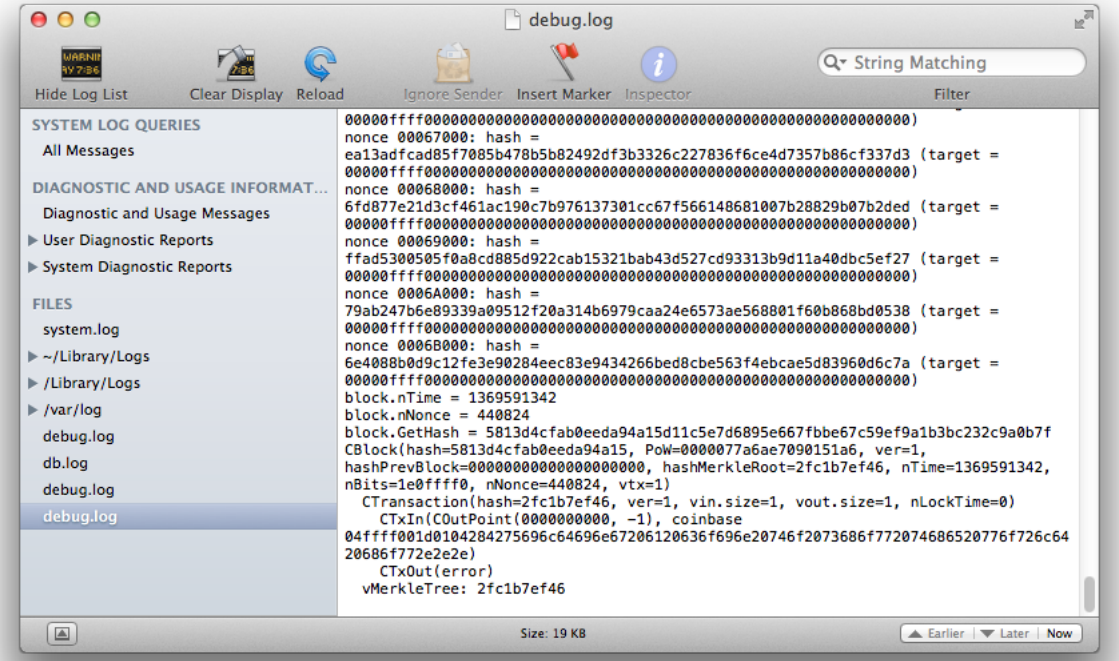
```
barcoin/src$ ./barcoin -testnet
```

You will hear your hard drive start to churn and it will seem like the coin has frozen in the window–but its not frozen, its mining a genesis block for the testnet based on your freshly working generated Merkel Hash. If you open the **debug.log** you'll see this in action:



Isn't that nifty? Its hashing a block happily, each nonce is ticking by. How long will this take? On an i7-2600 it can take 5-10 minutes. On a Core2Duo (like my iMac) it can take 20-30 minutes, maybe longer. Just let it do its thing, and go get some runts– eventually it will find one that it likes. This time it will write it in to the **testnet3** folder under your coin's conf folder in a file called **debug.log**:

```
Assertion failed: (block.GetHash() == hashGenesisBlock), function LoadBlockIndex, file main.cpp, line 2065.
zsh: abort      ./barcoin -testnet
```



Ah ha! See it there? There's a noonce and a genesis block hash, ripe for the plucking!

```
block.nNonce = 440824
block.GetHash = 5813d4cfab0eeda94a15d11c5e7d6895e667fbbe67c59ef9a1b3bc232c9a0b7f
```

Now, put these in to the main.cpp file:

src/main.cpp:

```
LINE 1984 hashGenesisBlock = uint256("0x5813d4cfab0eeda94a15d11c5e7d6895e667fbbe67c59ef9a1b3bc232c9a0b7f");
```

Yes, you need to leave the 0x in front of the hashes. You also need to enter the nNonce:

src/main.cpp:

```
LINE 2024 block.nNonce   = 440824;
```

Note that the sections of the main.cpp file we just edited correspond to the testnet and we haven't done the main net quite yet. This is because at this point, I usually get two systems up and running on the testnet to make sure they can mine coins and then I move on to the main net. Lets save our changes, and upload them to Github and then we'll move on to mining on the testnet:

```
barcoin% git add -A *
barcoin% git commit -m "changes"
barcoin% git push origin master
```

Mining Testnet Coins

First things first, rebuild your coin's executable on your local PC:

```
barcoin/src% make -f makefile.osx USE_UPNP=- (or .unix, whatever)
```

Now comes the part where you need two computers with distinct IP addresses. I find this easy to do with a Linux VPS and my home PC, so that's my example. Two machines on a LAN should work, and I believe 2 or more virtual machines should work too, as long as you're able to keep up with the IP addresses. Connect to your second machine and build the coin's file just as we did before–since you sent the code to Github, may as well use your new elite github skillz:

```
$ git clone https://github.com/barcoin/barcoin.git
cloning in to barcoin
```

```
$ cd barcoin/src
barcoin/src$ make -f makefile.unix (I'm on Linux here).
barcoin/src$ strip barcoind
```

Now I'm ready to run it–in testnet mode and with a connection to my “other” computer. This is kind of tricky, because you need to start the coin on both computers with the **-connect=x.x.x.x** variable, each with the IP of the other PC:

Home PC - iMac:

```
barcoin/src% ./barcoin -testnet -connect=9.5.6.5 &
```

VPS - Linux:

```
barcoin/src$ ./barcoin -testnet -connect=66.77.32.56 &
```

Add the & to the command will allow it to process in the background and allow you to continue to feed the coin commands without opening a second console window.

On the first run, it will complain about not having a .conf file:

```
error: You must set rpcpassword= in the configuration file:
/Users/username/Library/Application Support/barcoin/barcoin.conf
If the file does not exist, create it with owner-readable-only file permissions.
It is recommended you use the following random password:
rpcuser=barcoinrpc
rpcpassword=6WgBVzyEBaFJodzyKt69y8VaQBagPSGD5kHptnYGwjt5
(you do not need to remember this password)
If the file does not exist, create it with owner-readable-only file permissions.
```

Create this file, in whatever format you prefer, nano works great for this and assign an RPC user/password. If you want to use CGMiner/CPUMiner to mine solo later, make this something you'll remember. If you plan to only use the client's built in miner for solo mining, just cut and paste the auto generated info. This is the same file you may want to set up some of the bitcoin.conf commands in, here's a good reference: https://en.bitcoin.it/wiki/Running_Bitcoin#Bitcoin.conf_Configuration_File [https://en.bitcoin.it/wiki/Running_Bitcoin#Bitcoin.conf_Configuration_File]

- On OSX this file is: **/Users/username/Library/Application Support/barcoin/barcoin.conf**
- On Linux this file is **~/.barcoin/barcoin.conf**
- On Windows, this file is **c:\users\username\appdata\roaming\barcoin\barcoin.conf**

Side note: because I use a VPS for this, I don't really need to worry about port forwarding at that end. On the home PC, you will want to forward the port you chose for P2Pport in the cut and paste section to the PC you're using. For this example, that is port 55884.

Now start the coin again:

Home PC - iMac:

```
barcoin/src% ./barcoin -testnet -connect=9.5.6.5 &
```

VPS - Linux:

```
barcoin/src$ ./barcoin -testnet -connect=66.77.32.56 &
```

Now's a good time to brush up on the command line API calls syntax for interacting with the bitcoin client from this wiki page: https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_Calls_list [https://en.bitcoin.it/wiki/Original_Bitcoin_client/API_Calls_list]

First you'll want to send:

```
barcoin/src% ./barcoin getinfo
```

It should return something like this:

Home PC - iMac:

```
barcoin/src%./barcoin getinfo
{
    "version" : 1000000,
    "protocolversion" : 60001,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "blocks" : 0,
    "connections" : 1,
    "proxy" : "",
    "difficulty" : 0.00024414,
    "testnet" : true,
    "keypoololdest" : 1369621665,
```

```
    "keypoolsize" : 101,
    "paytxfee" : 0.00000000,
    "mininput" : 0.00010000,
    "errors" : ""
  }
```

The other side should look the same and the numbers should match. Note that testnet doesn't verify checkpoints, so they should connect pretty easily (oooh, that's the 1BTC question, but more on that in a bit…the other side:

VPS - Linux

```
/barcoin/src$./barcoind getinfo
{
    "version" : 1000000,
    "protocolversion" : 60001,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "blocks" : 0,
    "connections" : 1,
    "proxy" : "",
    "difficulty" : 0.00024414,
    "testnet" : true,
    "keypoololdest" : 1369622277,
    "keypoolsize" : 101,
    "paytxfee" : 0.00000000,
    "mininput" : 0.00010000,
    "errors" : ""
}
```

Lovely, they line up and each have a single connection. Now we can make one of them (or both) begin generating coins by using the following command:

```
barcoin/src% ./barcoin setgenerate true 16
```

The number is how many threads of your processor you want to devote, at the insanely low difficulty we're starting out with, this should be plenty to generate a few blocks. You won't see the results in real time, rather you'll need to issue the following command and evaluate the info:

```
barcoin/src% ./barcoin getmininginfo
{
    "blocks" : 0,
    "currentblocksize" : 1000,
    "currentblocktx" : 0,
    "difficulty" : 0.00024414,
    "errors" : "",
    "generate" : true,
    "genproclimit" : 16,
    "hashespersec" : 1432,
    "networkhashps" : -9223372036854775808,
    "pooledtx" : 0,
    "testnet" : true
}
```

Success! See that **hashespersec**? The internal scrypt miner is now doing its thing and making you some blocks. You'll have to issue the getmininginfo command a few times before it starts to count up in the block count. In just a few minutes you should be able to see:

```
barcoin/src$./barcoind getmininginfo
{
    "blocks" : 1,
    "currentblocksize" : 1000,
    "currentblocktx" : 0,
    "difficulty" : 0.00024414,
    "errors" : "",
    "generate" : true,
    "genproclimit" : 16,
    "hashespersec" : 1376,
    "networkhashps" : 32,
    "pooledtx" : 0,
    "testnet" : true
}
```

Woah doggie, we have blocks. Now verify that your other sees the blocks by doing a getinfo on your **other** computer:

barcoin/src%./barcoin getinfo

```
{
    "version" : 1000000,
    "protocolversion" : 60001,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "blocks" : 1,
    "connections" : 1,
    "proxy" : "",
```

```
    "difficulty" : 0.00024414,
    "testnet" : true,
    "keypoolooldest" : 1369621665,
    "keypoolsize" : 101,
    "paytxfee" : 0.00000000,
    "mininput" : 0.00010000,
    "errors" : ""
}
```

Well, whatta ya know? Whatta ya say we mine some mainnet coins?

Main Net Genesis Block

So really all we need to do now is update main.cpp with a new epoch time, in the main net section this time and mine a genesis block the similarly to the way we did it on testnet. First, stop the coind from running on both your local and remote computers by issuing the command:

```
barcoin/src% ./barcoind stop
Barcoin is stopping
```

Next, go back to your development PC and edit main.cpp with a new block.nTime:

src/main.cpp:

```
LINE 2017 block.nTime    = 1369623856; //epochtime
```

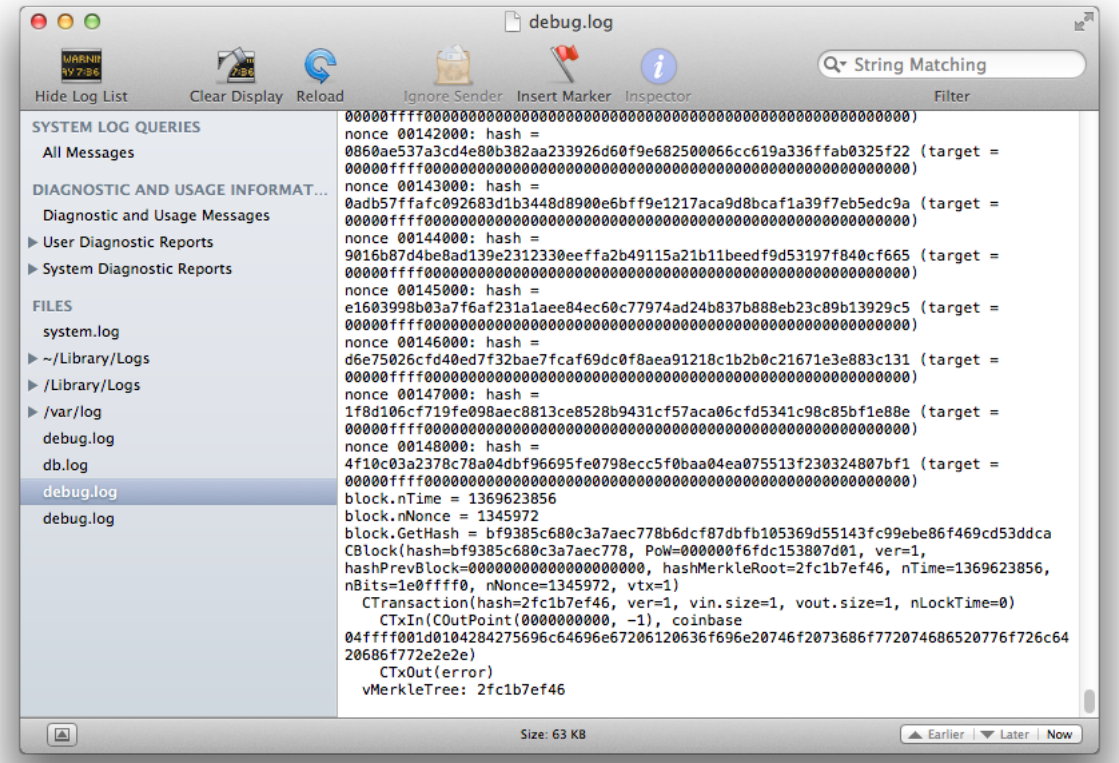
Now, recompile the coin again from the command line:

```
barcoin/src% make -f makefile.osx USE_UPNP=- (or .unix, whatever, ha!)
```

Now run the coin again, but this time don't include the -testnet switch:

```
barcoin/src% ./barcoin &
```

It will again seem to be frozen while it mines the genesis block and your processor will likely go to 100% utilization. Be patient, this took Satoshi 6 days or some shit, right? Again, if you have a Mac, watching it mine with the system log viewer is pretty fun–and then Success:



Now, we just do the same as we did on the testnet, and make these changes to main.cpp:

src/main.cpp:

```
LINE 32 uint256 hashGenesisBlock("0xbf9385c680c3a7aec778b6dcf87dbfb105369d55143fc99ebe86f469cd53ddca");
LINE 2019 block.nNonce   = 1345972;
LINE 2034 if (false && block.GetHash() != hashGenesisBlock)
```

Checkpoints

Changing line 2034 to false will keep clients from trying to hash their own genesis block should something be awry. One more file to change:

src/checkpoints.cpp

```
LINE 27 (         0, uint256("0xbf9385c680c3a7aec778b6dcf87dbfb105369d55143fc99ebe86f469cd53ddca"))
```

This is the “trick.” Remember I said, there was a trick? This is it. Hash 0 in this file needs to be set to the genesis block hash, so do it and rejoice as you've now nearly finished creating your clone! Should you want to make your coin “legit” you'll want to revisit this file in the future and add other checkpoints in to it–but that's a bit we'll save for the end of the guide. Lets send our changes to Github before we build:

```
barcoin% git add -A *
barcoin% git commit -m "changes"
barcoin% git push origin master
```

Ok, we're ready to rebuild on the first pc:

```
barcoin% cd src/
barcoin/src% make -f makefile.osx USE_UPNP=- (or .unix, blah...)
strip barcoin
```

Now on the second pc (assuming its Linux here):

```
~$ cd barcoin
barcoin$ git pull
updating git...wait wait...done!
barcoin$ cd src/
barcoin/src$ make -f makefile.unix USE_UPNP=-
strip barcoind
```

Ooo laa laa, we're done here. Now we can mine us some coinz!

Mining Main Net Coins

The process here is the same as the testnet, but without the -testnet switch. Start'er up:

Home PC - iMac:

```
barcoin/src% ./barcoin -connect=9.5.6.5 &
```

VPS - Linux:

```
barcoin/src$ ./barcoin -connect=66.77.32.56 &
```

Verify with getinfo:

```
barcoin/src%./barcoind getinfo
{
    "version" : 1000000,
    "protocolversion" : 60001,
    "walletversion" : 60000,
    "balance" : 0.00000000,
    "blocks" : 0,
    "connections" : 1,
    "proxy" : "",
    "difficulty" : 0.00024414,
    "testnet" : false,
    "keypoololdest" : 1369627515,
    "keypoolsize" : 101,
    "paytxfee" : 0.00000000,
    "mininput" : 0.00010000,
    "errors" : ""
}
```

Get a new address:

```
barcoin getnewaddress
GeeNLCk9KfQQ35wPpf2faYdqHEU5KW4Khn
```

Start one of them (or both of them mining) and verify it:

```
barcoin/src%./barcoind setgenerate true 16
barcoin/src%./barcoind getmininginfo
{
    "blocks" : 0,
    "currentblocksize" : 1000,
    "currentblocktx" : 0,
    "difficulty" : 0.00024414,
    "errors" : "",
    "generate" : true,
    "genproclimit" : 16,
    "hashespersec" : 1417,
    "networkhashps" : -9223372036854775808,
    "pooledtx" : 0,
    "testnet" : false
}
```

Oooooh myyyyyy goooooooooood, right? Its making blocks of our new Barcoin (or is it a BAR to consolidate your digital wealth? I mean, shoot, there's only a weeks worth at pump and dump mining rates right?) Soon you will see: **"blocks" : 1,** and then that number will start to climb. Now's the time you could set up the barcoin.conf client to accept connections from your LAN and point your dualie-7970 boxen at it or perhaps a minerd. Its ready to rock and roll at this point.

Things to remember:

- You're basically done here. The command line version can do everything the -Qt can.
- Blocks take 120 confirms, so you'll need to leave a system mining even at just a few hashses to keep your network going. I like to leave my VPS mining at just a few Kh/s and use it as the seed node so that the network is always confirming even if its very slow.
- You're basically done here. But no, you're not–lets make some GUI wallets.

Compiling the -Qt Wallets

Ok, so this will make or break your coin if you plan to distribute it. Before I go deep in to this, know that the source code for foocoin is customized to make building as easy as possible on Windows (the hardest system to build for). It is also fairly easy to build a Mac version, but at this point I'm having trouble redistributing the Mac versions with the other PC having the same dependencies installed. As for Linux, surprisingly enough, its the easiest to build for and if you installed all the dependencies from the top section of the guide you'll be able to knock it out with two commands.

Mac OSX -Qt

I'm starting with this one simply to go inline with dependencies order above. In order to keep things tidy on my iMac I created a virtual machine loaded with OSX 10.6.8, Snow Leopard. This was pretty straight forward using VMWare Fusion. After install and software updating, I installed XCode 3.2.6, which contains a working non-llvm version of gcc and its free from Apple here: http://connect.apple.com/cgi-bin/WebObjects/MemberSite.woa/wa/getSoftware?bundleID=20792 [http://connect.apple.com/cgi-bin/WebObjects/MemberSite.woa/wa/getSoftware?bundleID=20792] A simple install, no frills, make sure all the objects are checked for installation.

Next, I installed MacPorts this version: https://distfiles.macports.org/MacPorts/MacPorts-2.1.3-10.6-SnowLeopard.pkg [https://distfiles.macports.org/MacPorts/MacPorts-2.1.3-10.6-SnowLeopard.pkg] and then the dependencies listed in the first section ala:

```
 xcode%sudo port install boost db48 qt4-mac openssl miniupnpc git
```

After a bit of time, all goodies are installed so we'll clone the coin's software in the regular fashion:

```
xcode% git clone https://github.com/barcoin/barcoin.conf
xcode% cd barcoin
```

Now, something a tad different this time, we need to run qmake instead of make. Do that like so:

```
barcoin% qmake "USE_UPNP=-" barcoin-qt.pro
```

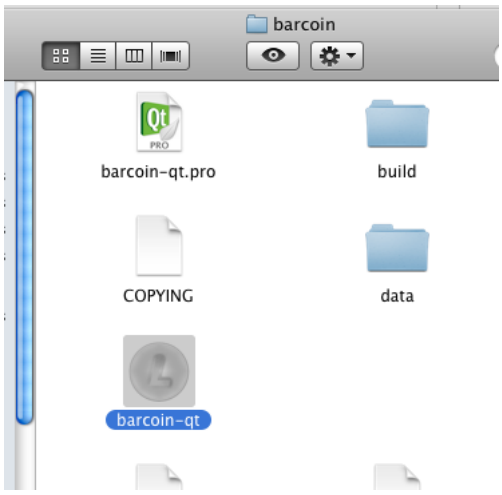
Yes, you need the " " around USE_UPNP=- and yes, this may produce some strange looking results, something like this:

```
Project MESSAGE: Building without UPNP support
Removed plural forms as the target language has less forms.
If this sounds wrong, possibly the target language is not set or recognized.
```

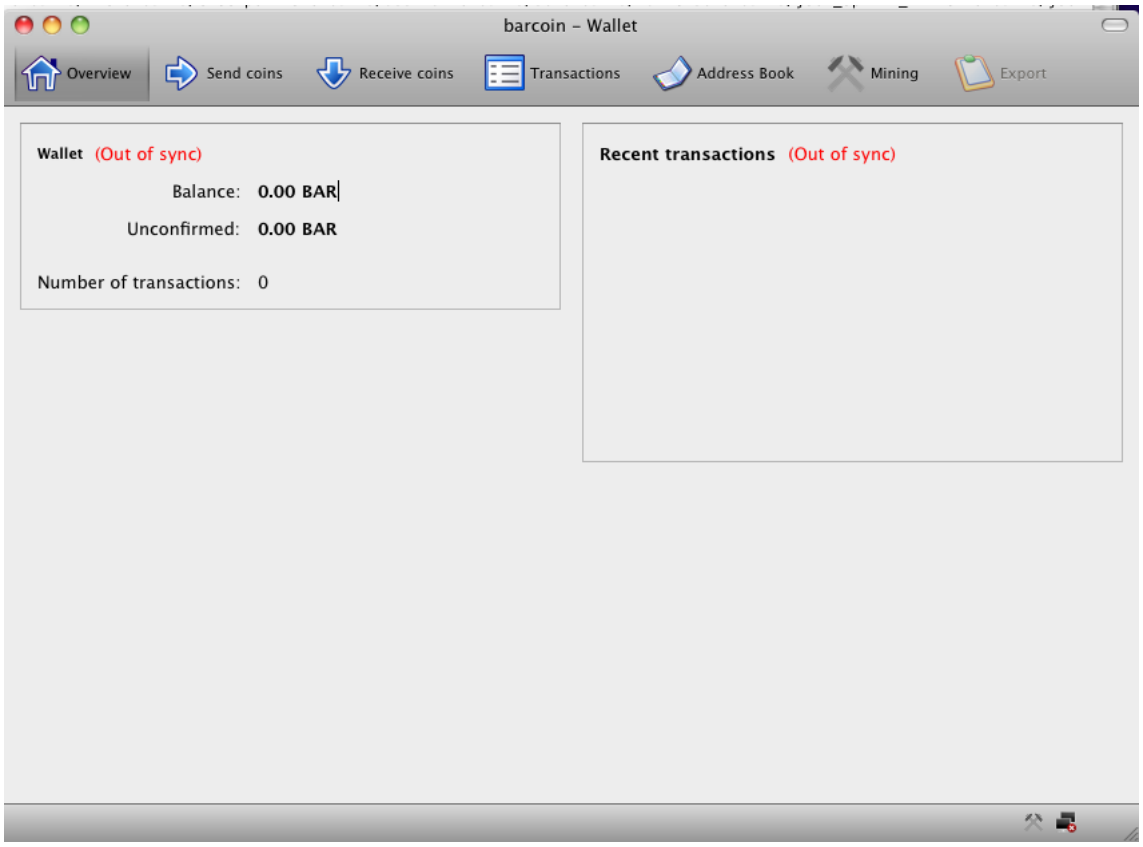
Now, lets build it:

```
barcoin% make -f Makefile
```

Go, go, go, do not look back. After a bit you'll see it finish and an icon should appear in the barcoin folder:



Now launch that and voila! A Mac barcoin wallet:



Just like the Windows and Linux wallets, you may want to add **addnode=x.x.x.x** where the x.x.x.x is the IP of your seed node. This won't be needed after a few clients begin connecting to the network, eventually they will begin talking to each other via IRC.

Linux -Qt

This is by a long shot the easiest wallet to compile, but its hindered by two things for distribution: Linux has very small market share, though for a personal or club coin, what the hell right? and Most Linux users will compile their own software so you'll not likely get far distributing a Linux executable (as well you shouldn't). My example here is based on Debian and should equate to most Debian/Ubuntu flavors.

Now, since we already built a system and installed the dependencies in the first bit–wait, you didn't? You did it all on Windows? Nice. You should write a guide next time! Now, where were we…oh yes, you already have a working coin building system, so lets just stick with it. First things first:

```
cd ~/barcoin
barcoin% qmake "USE_UPNP=-"
```

Thinking, thinking, output:

```
Project MESSAGE: Building without UPNP support
Removed plural forms as the target language has less forms.
If this sounds wrong, possibly the target language is not set or recognized.
```

Now, build it:

```
barcoin$ make
```

Yeah, seriously, that's it. Just 'make.' Ha–Debian is so beautiful, is it not? Ok now after a bit of churning and burning it will finish.

Windows -Qt

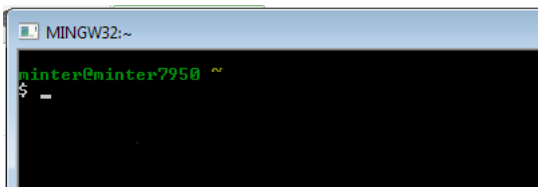
This is the trickiest one to crack of the GUI wallets. I am going to detail how I got this to work and offer you an easy way to get the dependencies in an attempt to make this work for you too. That said, it may not–and I've already said I won't do tech support. So here's the deal. I got this to work and then duplicated it on a second machine to ensure it wasn't a fluke! Most of the information needed to compile the basic coind.exe or GUI wallet is in this thread: https://bitcointalk.org/index.php?topic=149479.0 [https://bitcointalk.org/index.php?topic=149479.0] Unfortunately nothing is as easy as it seems, and although the MinGW and QT installs went fine, I couldn't compile it without a few tweaks to the .pro file.

If you don't want to install these dependencies by hand, clone https://github.com/foocoin/deps.git [https://github.com/foocoin/deps.git] in to C: If not, here's how to do it manually:

Begin by installing MinGW32 from here: https://sourceforge.net/downloads/mingw [https://sourceforge.net/downloads/mingw] Go ahead and install the whole bloody thing if you like, but at least the “C Compiler”, “C++ Compiler” and “MSYS Basic System” components. Everything else leave stock, next, next, next kind of thing.

Next, install ActivePerl 32 or 64 bit from here: http://www.activestate.com/activeperl/downloads [http://www.activestate.com/activeperl/downloads] Again, standard install, next, next, next and so forth.

Now open the “MinGW System Shell” from Start - Programs and you'll basically have a Linux prompt:



Now make a /c/deps folder to keep our files in:

```
$mkdir /c/deps
$cd /c/deps
```

Now download the following files and put them in C:\Deps:

- OpenSSL: http://www.openssl.org/source/openssl-1.0.1e.tar.gz [http://www.openssl.org/source/openssl-1.0.1e.tar.gz]

Install it like so:

```
/c/deps$ tar xvfz openssl-1.0.1e.tar.gz
/c/deps$ cd openssl-1.0.1e
/c/deps$ ./config
/c/deps$ make
```

- Berkeley DB 4.8: http://download.oracle.com/berkeley-db/db-4.8.30.NC.tar.gz [http://download.oracle.com/berkeley-db/db-4.8.30.NC.tar.gz]

Install it like so:

```
/c/deps$ tar xvfz db-4.8.30.NC.tar.gz
/c/deps$ cd db-4.8.30.NC/build_unix
/c/deps$ ../dist/configure --disable-replication --enable-mingw --enable-cxx
```

- Boost: http://sourceforge.net/projects/boost/files/boost/1.53.0/ [http://sourceforge.net/projects/boost/files/boost/1.53.0/]

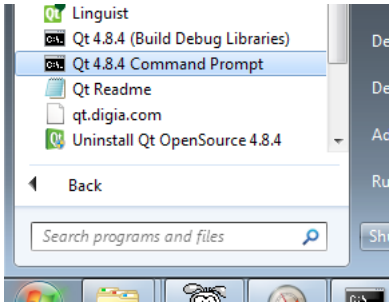
For this one, open a regular command (CMD) window and do the following:

```
cd \deps\boost-1.53.0\
bootstrap.bat mingw
b2 --build-type=complete --with-chrono --with-filesystem --with-program_options --with-system --with-thread toolset=gcc stage
```

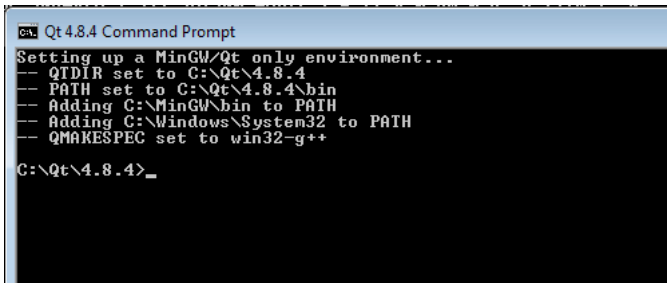
For simplicity's sake, my versions are simply named deps\boost; deps\ssl; etc. If you build your own, either rename the folders in \deps OR change the paths to suit your changes in the coin-qt.pro file. Remember to change the Boost suffix too to match the version you compile with!

At this point you're ready to build normal non-Qt coin wallets on windows. Go ahead and check the thread at the beginning of this section if you'd like to know how. We're making a GUI though:

Next, install the Qt-MiniGW32 4.8.4 Build from here: http://download.qt-project.org/official_releases/qt/4.8/4.8.4/qt-win-opensource-4.8.4-mingw.exe [http://download.qt-project.org/official_releases/qt/4.8/4.8.4/qt-win-opensource-4.8.4-mingw.exe] Again, all normal installation options, next next next…you know the drill. Once QT is installed, you will find a program in Start - All Programs - Qt by Digia - Qt Command Prompt:



Fire it up and it will look pretty much like a DOS box:



Now since we don't have git on this our Windows computer (you can install it if you want, Cygwin is a good way to do that) you must download the barcoin-master.zip file from http://github.com/barcoin [http://github.com/barcoin] and extract it to the PC. For this example, we'll put it in c:\. One last thing we need to do before we compile for Windows. We need to edit the “barcoin-qt.pro” file to enable the Windows libs, includes, and correct ordering for some of the syntax:

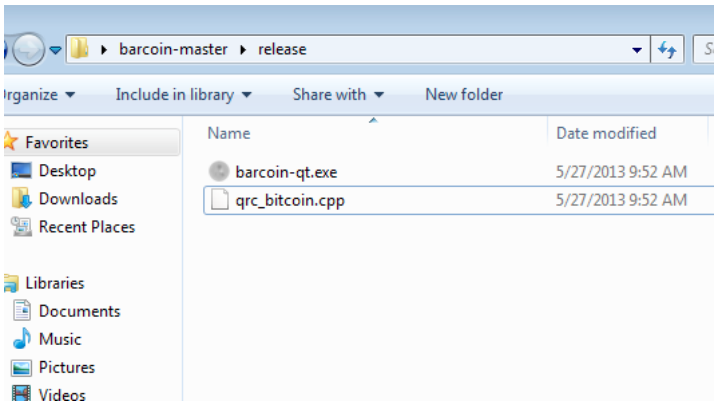
barcoin/barcoin-qt.pro:

```
LINES 11-22, UNCOMMENT ALL OF THESE TO ENABLE WINDOWS BUILDS:
#windows:LIBS += -lshlwapi
#LIBS += $$join(BOOST_LIB_PATH,,-L,) $$join(BDB_LIB_PATH,,-L,) $$join(OPENSSL_LIB_PATH,,-L,) $$join(QRENCODE_LIB_PATH,,-L,)
#LIBS += -lssl -lcrypto -ldb_cxx$$BDB_LIB_SUFFIX
#windows:LIBS += -lws2_32 -lole32 -loleaut32 -luuid -lgdi32
#LIBS += -lboost_system-mgw46-mt-sd-1_53 -lboost_filesystem-mgw46-mt-sd-1_53 -lboost_program_options-mgw46-mt-sd-1_53 -lboost_thread-mgw46
#BOOST_LIB_SUFFIX=-mgw46-mt-sd-1_53
#BOOST_INCLUDE_PATH=C:/deps/boost
#BOOST_LIB_PATH=C:/deps/boost/stage/lib
#BDB_INCLUDE_PATH=c:/deps/db/build_unix
#BDB_LIB_PATH=c:/deps/db/build_unix
#OPENSSL_INCLUDE_PATH=c:/deps/ssl/include
#OPENSSL_LIB_PATH=c:/deps/ssl
```

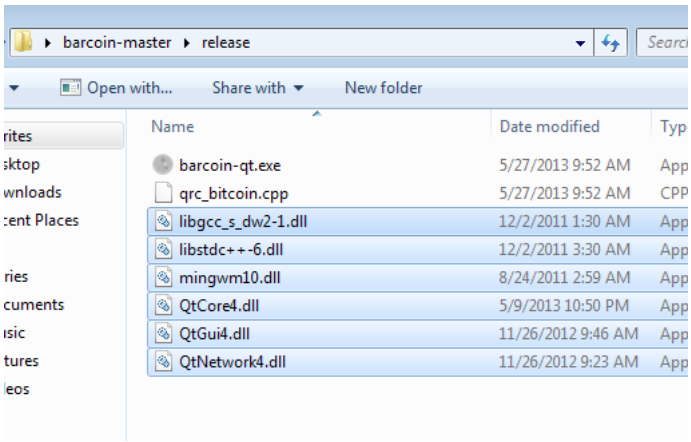
IF YOU BUILT YOUR OWN dependencies, then also change the paths in the file above to suit their locations, use / instead of \, yea–its odd. Now go back to your Qt Command Shell window and build the same way we built on the other platforms:

```
c:\Qt-Builder> cd \barcoin-master\src
c:\barcoin-master\src> qmake "USE_UPNP=- barcoin-qt.pro
c:\barcoin-master\src> make -f Makefile.Release
```

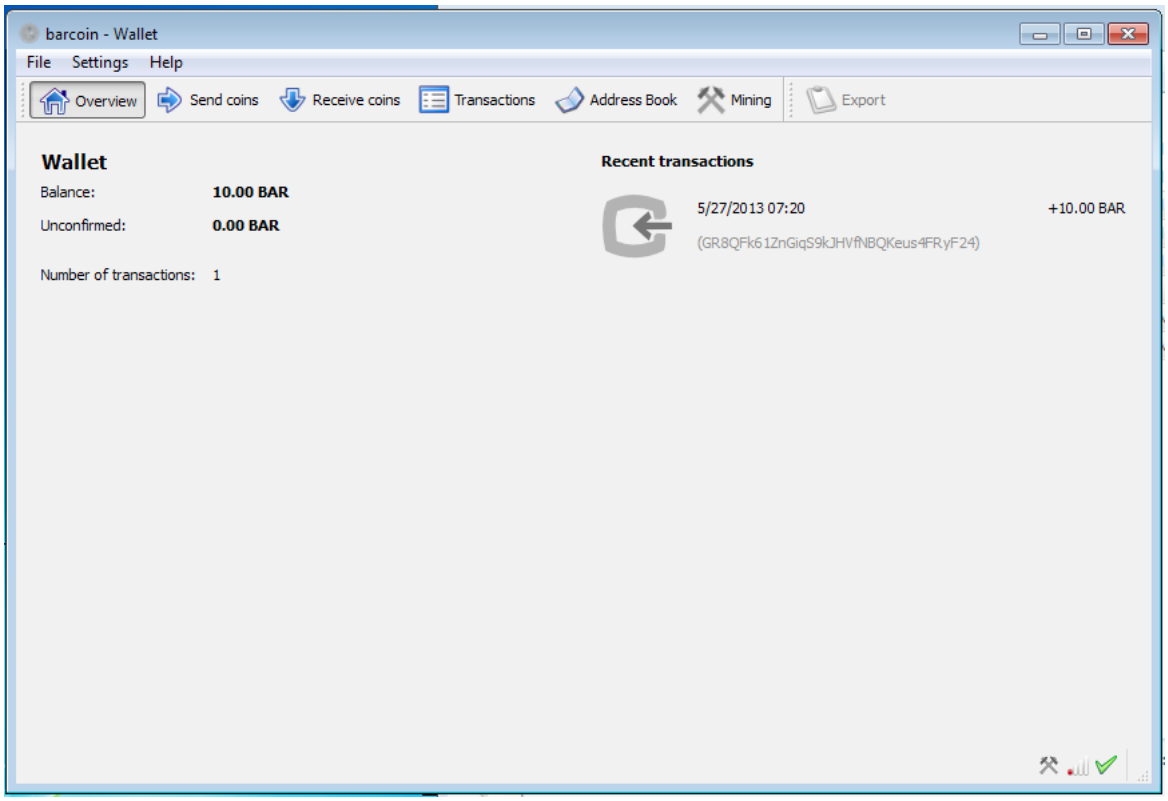
Wait for a bit…and once its done, you'll find a folder called “release” under the main barcoin-master folder containing the .exe and a .cpp file:



This isn't enough to redistribute though, to make the file run you'll need to include the QT and gcc libs along with the file. I've put them on a git repository here: https://github.com/foocoin/windows-qt-addins.git [https://github.com/foocoin/windows-qt-addins.git] Just download the 6 files and insert them in to the “release” folder along with the .exe and .cpp:



To redistribute, simply rename the “release” folder and zip it up! You can now run the .exe file on Windows:



Woah, hey look at that, we already have a balance! Actually, I'd sent the 10 BAR to this computer from the one I left mining all night. If you don't have many connections to the

network, you may need to add a line like so to your %appdata%\barcoin\barcoin.conf:

```
addnode=IP ADDRESS OF YOUR SEED NODE
```

If you created a seed node, it should connect but if not, simply add a node. Once a few clients begin connecting they will use IRC to connect to each other, so the addnode should only be needed for the initial wallet connections.

- scrypt_altcoin_cloning_guide.txt · Last modified: 2013/05/27 12:32 by shakezula
-