How to clone ABCCoin in to another SHA256 coin. By shakezula.

Step 1:

Search and replace the entire source tree for the terms “ABCCoin” and “abccoin” and replace them with your coin’s name. Be sure to use Case Sensitive search/replace if you want to keep the normal naming conventions, though its optional.

Step 2:

Change the following lines of code (in bold) to suit your coin’s chosen parameters:

File: src/chainparams.cpp:

LINES 26-29:

```
pchMessageStart[0] = 0x04;
pchMessageStart[1] = 0x04;
pchMessageStart[2] = 0xb5;
pchMessageStart[3] = 0x04;
```

The bits in bold are the start values in hex. You can simply randomize them, set them all to something different, or set them all the same. Just don’t leave them the same as the coin you’re cloning—it can cause chains to intermingle (or try to).

LINES 30-31:

```
nDefaultPort = 5530;
nRPCPort = 5531;
```

Set these to your preferred ports for RPC/P2P.

LINE 33:

```
nSubsidyHalvingInterval = 100000;
```

If you want your coin to halve, put in the number of blocks you'd like it to halve at here. This is optional and is related to changes in main.cpp which will be lower in the list.

<u>LINE 38:</u>

```
const char* pszTimestamp = "ABCCoin";
```

Change this to something unique to specify the beginning of your coin. This can be any phrase or link you'd like to use, just plain text.

<u>LINE 49:</u>

```
genesis.nTime    = 1300000000;
```

Set this to the current epoch or desired Unix time. You can get Unix time from this site: http://www.epochconverter.com/

<u>LINE 51:</u>

```
genesis.nNonce   = 0;
```

This number will need filled in with information from the debug.log on the second compile.

<u>LINES 55-58:</u>

```
//while (hashGenesisBlock > bnProofOfWorkLimit.getuint256()){
//    if (++genesis.nNonce==0) break;
//    hashGenesisBlock = genesis.GetHash();
//}
```

This is the Genesis block hashing bit. Uncomment this to create a new Genesis Block, leave it commented for normal coin operation. Note that line 54 is NOT commented in normal operation, don't comment it out EVER.

<u>LINES 66-67:</u>

```
        assert(hashGenesisBlock == uint256("0xsomehash”));
        assert(genesis.hashMerkleRoot == uint256("0xsomehash”));
```

This is where you’ll input the merkel/genesis block after creation. For the first go around, leave this blank, it will then require the code to create the GB/Merkel on first run.

LINE 69:

```
        vSeeds.push_back(CDNSSeedData("someaddress.com or IP
addy", "someaddress.com"));
```

Put your IP of a seednode in here, or a FQDN (something.com). Put the same info in both fields.

LINE 72:

```
        base58Prefixes[PUBKEY_ADDRESS] = 36;
```

Change this, its the first letter or digit of the coin’s address. Find an address using this guide: https://en.bitcoin.it/wiki/List_of_address_prefixes

File: src/main.cpp

LINES 54/56:

```
int64 CTransaction::nMinTxFee = 10000;

int64 CTransaction::nMinRelayTxFee = 10000;
```

Set transaction fees if desired here, in single-satoshi format.

Lines 1248-1267:

```
static const int64 nStartSubsidy = 10 * COIN;
static const int64 nMinSubsidy = 1 * COIN;
```

Set the number of coins per block and minimum coins per block (if you're using halving at a specific interval) this will allow the block to halve down to a certain point but continue to produce a reward of whatever you set here.

```
int64 static GetBlockValue(int nHeight, int64 nFees)
{
    int64 nSubsidy = nStartSubsidy;

    // Mining phase: Subsidy is cut in half every SubsidyHalvingInterval
    nSubsidy >>= (nHeight / Params().SubsidyHalvingInterval());
```

If you're not using halving, simply comment this line out, and then these lines too:

```
    // Inflation phase: Subsidy reaches minimum subsidy
    // Network is rewarded for transaction processing with transaction fees
and
    // the inflationary subsidy
    if (nSubsidy < nMinSubsidy)
    {
        nSubsidy = nMinSubsidy;
    }

    return nSubsidy + nFees;
}
```

You can do any kind of tricks in here that the scrypt coins do, ala, random blocks, super blocks, etc. You can even take another coin's complete nSubsidy section and paste it right in here if you'd like to.

<u>LINES 1269-1270:</u>

```
static const int64 nTargetTimespan = 60; // 1 minutes
static const int64 nTargetSpacing = 30; // 30 seconds
```

The top one is how often you want blocks to be found, i.e.: block target. If you want 2 minute blocks, put in 120, 4 minute blocks would be 240, etc. The bottom one is how often the retarget for difficulty will calculate, again this is in seconds. Your coin will retarget (change diff) at the interval ratio of the top number over the bottom number, in this coin, that'd be once every 2 blocks (60/30=2)

LINE 1273:

static const int64 nAveragingInterval = nInterval * **20**; // 40 blocks

This line is how many blocks back the code will consider the hash rate for when deciding what to make the diff adjust by. You can make this longer to make more smooth curves in diff change, or shorter to make the coin react faster.

LINES 1276-1277:

tatic const int64 nMaxAdjustDown = **20**; // 20% adjustment down
static const int64 nMaxAdjustUp = **10**; // 10% adjustment up

This is the percentage that any one diff jump will be. Remember this is all math and averages, so crazy high or low numbers here can have unforeseen consequences.

File: src/main.h:

LINE 47:

static const int64 MAX_MONEY = **1000000000** * COIN;

Set this number to be the maximum number of coins ever to be created.

File: src/qt/bitcoinunits.cpp:

LINES 37-39:

```
case BTC: return QString("ABC");
case mBTC: return QString("mABC");
case uBTC: return QString::fromUtf8("µABC");
```

This is the 3 letter designator (or 2 letter, or 4 letter, etc) for your coin that will show up in the GUI wallet. Don't change it ANYWHERE else in the file, only these lines.

LINES 48-50:

```
    case BTC: return QString("ABCCoin");
    case mBTC: return QString("MilliABCCoin (1 / 1,000)");
    case uBTC: return QString("MicroABCCoin (1 / 1,000,000)");
```

Like above, you can also change the long hand name of your coin here, but don't change it anywhere else in the file.

File: src/version.cpp:

LINE 11:

```
const std::string CLIENT_NAME("ABCCoin");
```

The name of your client that will appear in the "Debug Console." Can be set to anything.

LINE 14:

```
#define CLIENT_VERSION_SUFFIX   "-beta"
```

The suffix of your client's version (0.9.0-beta) in the about dialog. Can be set to anything.

File src/clientversion.h:

LINES 9-12:

```
#define CLIENT_VERSION_MAJOR             0
#define CLIENT_VERSION_MINOR             9
#define CLIENT_VERSION_REVISION          0
#define CLIENT_VERSION_BUILD             0
```

This is the version number for your client. DO NOT set this lower than 0.8.9!! This must be higher than the version of Bitcoin you're customizing! 0.9.0 is a good starting point, as is 1.0.0.

File src/checkpoints.cpp:

LINE 40:

```
( 0,     uint256("0xsomehash”))
```

This will need updated with the hash of your genesis block before final compile.

LINE 45:

```
1300000000, // * UNIX timestamp of last checkpoint block
```

This needs to match the same timestamp you used on line 49 in **chainparams.cpp**. For simplicity sake, I color coded them, make sure they match.

There are a few other items you can change, mostly in /src/qt/bitcoingui.cpp (stuff like “Send Coins” or the names of the menus on the Toolbars) and in /src/qt/forms/about.ui (The names and info that goes in the About Dialog. All completely optional.

Step 3:

Customize the following graphics files:

/src/qt/res/icons/**bitcoin.png**
/src/qt/res/icons/**bitcoin.ico**
/src/qt/res/icons/**bitcoin_testnet.ico**
/src/qt/res/icons/**bitcoin.icns**
/src/qt/res/icons/**toolbar.png**
/src/qt/res/icons/**toolbar_testnet.png**

These are the icons for the coin. I like to use iConvert.com to make these files, do not change their names. Bitcoin.png will be used as the “in-wallet” logo also by default.

/src/qt/res/images/**splash.png**
/src/qt/res/images/**splash_testnet.png**
/src/qt/res/images/**about.png**

These are the splash screens and the image from inside the “About” Dialog.

The other icons in /src/qt/res/icons can be changed to your liking as well, just don’t rename any of them. This can give you custom graphics for the toolbars, transactions, and mining bits in wallet.

Step 4:

This is where you now want to compile your coin for the first time. I do this on my iMac or Debian box, but you could also do it on your Windows machine with MinGW or whatever. Run make -f makefile.unix or whatever and then once the coin’s built, run it for the first time with ./abccoind &. It will error out immediately.

Now check the %appdata%\abccoin\debug.log file and you’ll find this info:

```
00000c4782d13f093d5135f3579b1b9e1bcb14b1763f46c9db269de4e0248c
49
4674a0af6bf5e5cc97108ddb9aefa6513ba6f25474943b0a97caed71078727
96
1e0fffff
CBlock(hash=00000c4782d13f093d5135f3579b1b9e1bcb14b1763f46c9d
b269de4e0248c49, ver=1, hashPrevBlock=0000000000
nTime=1300000000, nBits=1e0fffff, nNonce=2200113, vtx=1)
  CTransaction(hash=4674a0af6b, ver=1, vin.size=1, vout.size=1,
nLockTime=0)
    CTxIn(COutPoint(0000000000, 4294967295), coinbase
04ffff001d01040b46697265466c79436f696e)
    CTxOut(nValue=1.00000000,
scriptPubKey=04678afdb0fe5548271967f1a67130)
```

vMerkleTree:
4674a0af6bf5e5cc97108ddb9aefa6513ba6f25474943b0a97caed7107872796

You need to take the bits and put them in to the **src/chainparams.cpp** and **src/checkpoints.cpp** at the lines specified above. For simplicity's sake, they are color coded as well—and check that the nTime is the SAME as the one you set!

The **GREEN** one is the Genesis Block Hash. Goes in **BOTH** files above.

The **ORANGE** one is the Merkel Root. Goes in **chainparams.cpp**.

The **BLUE** one is the nNonce. Goes in **chainparams.cpp**.

Put these values in their corresponding files and recompile the coin using your preferred method. At this point, the coin should accept the genesis block and you're ready to compile a Windows Qt or distribute the source. ABCCoin's -qt.pro file has been modified to make compiling EASY with the foocoin dependency package and minGW/QT 4.8.5 Command Line for Windows.