LEARNING bitcoin

#bitcoin

Table of Contents

About

Chapter 1: Getting started with bitcoin

Remarks

This section provides an overview of what bitcoin is, and why a developer might want to use it.

It should also mention any large subjects within bitcoin, and link out to the related topics. Since the Documentation for bitcoin is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

On Debian-based distributions (e.g. Ubuntu), follow the following steps.

Get Bitcoin Package:
`apt-add-repository ppa:bitcoin/bitcoin`

Update:
`apt-get update`

Install:
`apt-get install bitcoind -y`

Reboot:
`reboot`

After rebooting confirm Bitcoin was installed by checking for the directory `~/.bitcoin` or by running bitcoind:
`bitcoind`

Node Configuration

bitcoin.conf

The `bitcoin.conf` file allows customization for your node. Create a new file in a text-editor and save it as `bitcoin.conf` in your `/bitcoin` directory.

Location of your `/bitcoin` directory depends on your operation system.

Windows XP `C:\Documents and Settings\<username>\Application Data\Bitcoin`
Windows Vista, 7, 10 `C:\Users\<username>\AppData\Roaming\Bitcoin`
Linux `/home/<username>/.bitcoin`
Mac OSX `/Users/<username>/Library/Application Support/Bitcoin`

What a typical `bitcoin.conf` file could look like:

```
rpcuser=someusername
rpcpassword=somepassword
rpcallowip=localhost
daemon=1
prune=600
minrelaytxfee=2500
maxconnections=20
maxuploadtarget=250
```

Below is a list of available options and settings available per the Bitcoin source code.

```
-?    This help message
-alerts    Receive and display P2P network alerts (default: 1)
-alertnotify=<cmd>    Execute command when a relevant alert is received or we see a really
long fork (%s in cmd is replaced by message)
-blocknotify=<cmd>    Execute command when the best block changes (%s in cmd is replaced by
block hash)
-checkblocks=<n>    How many blocks to check at startup (default: 288, 0 = all)
-checklevel=<n>    How thorough the block verification of -checkblocks is (0-4, default: 3)

-conf=<file>    Specify configuration file (default: bitcoin.conf)
-datadir=<dir>    Specify data directory
-dbcache=<n>    Set database cache size in megabytes (4 to 16384, default: 100)
-loadblock=<file>    Imports blocks from external blk000??.dat file on startup
-maxorphantx=<n>    Keep at most <n> unconnectable transactions in memory (default: 100)
-maxmempool=<n>    Keep the transaction memory pool below <n> megabytes (default: 300)
-mempoolexpiry=<n>    Do not keep transactions in the mempool longer than <n> hours (default:
72)
-par=<n>    Set the number of script verification threads (-2 to 16, 0 = auto, <0 = leave that
many cores free, default: 0)
-prune=<n>    Reduce storage requirements by pruning (deleting) old blocks. This mode is
incompatible with -txindex and -rescan. Warning: Reverting this setting requires re-
downloading the entire blockchain. (default: 0 = disable pruning blocks, >550 = target size in
MiB to use for block files)
-reindex    Rebuild block chain index from current blk000??.dat files on startup
-txindex    Maintain a full transaction index, used by the getrawtransaction rpc call
(default: 0)

Connection options:
-addnode=<ip>    Add a node to connect to and attempt to keep the connection open
-banscore=<n>    Threshold for disconnecting misbehaving peers (default: 100)
-bantime=<n>    Number of seconds to keep misbehaving peers from reconnecting (default: 86400)

-bind=<addr>    Bind to given address and always listen on it. Use [host]:port notation for
IPv6
-connect=<ip>    Connect only to the specified node(s)
-discover    Discover own IP addresses (default: 1 when listening and no -externalip or -
proxy)
-dns    Allow DNS lookups for -addnode, -seednode and -connect (default: 1)
-dnsseed    Query for peer addresses via DNS lookup, if low on addresses (default: 1 unless -
connect)
-externalip=<ip>    Specify your own public address
-forcednsseed    Always query for peer addresses via DNS lookup (default: 0)
-listen    Accept connections from outside (default: 1 if no -proxy or -connect)
-listenonion    Automatically create Tor hidden service (default: 1)
-maxconnections=<n>    Maintain at most <n> connections to peers (default: 125)
-maxreceivebuffer=<n>    Maximum per-connection receive buffer, <n>*1000 bytes (default: 5000)

-maxsendbuffer=<n>    Maximum per-connection send buffer, <n>*1000 bytes (default: 1000)
-onion=<ip:port>    Use separate SOCKS5 proxy to reach peers via Tor hidden services (default:
```

```
-proxy)
-onlynet=<net>    Only connect to nodes in network <net> (ipv4, ipv6 or onion)
-permitbaremultisig    Relay non-P2SH multisig (default: 1)
-peerbloomfilters    Support filtering of blocks and transaction with bloom filters (default:
1)
-port=<port>    Listen for connections on <port> (default: 8333 or testnet: 18333)
-proxy=<ip:port>    Connect through SOCKS5 proxy
-proxyrandomize    Randomize credentials for every proxy connection. This enables Tor stream
isolation (default: 1)
-seednode=<ip>    Connect to a node to retrieve peer addresses, and disconnect
-timeout=<n>    Specify connection timeout in milliseconds (minimum: 1, default: 5000)
-torcontrol=<ip>:<port>    Tor control port to use if onion listening enabled (default:
127.0.0.1:9051)
-torpassword=<pass>    Tor control port password (default: empty)
-upnp    Use UPnP to map the listening port (default: 0)
-whitebind=<addr>    Bind to given address and whitelist peers connecting to it. Use
[host]:port notation for IPv6
-whitelist=<netmask>    Whitelist peers connecting from the given netmask or IP address. Can
be specified multiple times. Whitelisted peers cannot be DoS banned and their transactions are
always relayed, even if they are already in the mempool, useful e.g. for a gateway
-whitelistalwaysrelay    Always relay transactions received from whitelisted peers (default:
1)
-maxuploadtarget=<n>    Tries to keep outbound traffic under the given target (in MiB per
24h), 0 = no limit (default: 0)

Wallet options:
-disablewallet    Do not load the wallet and disable wallet RPC calls
-keypool=<n>    Set key pool size to <n> (default: 100)
-fallbackfee=<amt>    A fee rate (in BTC/kB) that will be used when fee estimation has
insufficient data (default: 0.0002)
-mintxfee=<amt>    Fees (in BTC/kB) smaller than this are considered zero fee for transaction
creation (default: 0.00001)
-paytxfee=<amt>    Fee (in BTC/kB) to add to transactions you send (default: 0.00)
-rescan    Rescan the block chain for missing wallet transactions on startup
-salvagewallet    Attempt to recover private keys from a corrupt wallet.dat on startup
-sendfreetransactions    Send transactions as zero-fee transactions if possible (default: 0)

-spendzeroconfchange    Spend unconfirmed change when sending transactions (default: 1)
-txconfirmtarget=<n>    If paytxfee is not set, include enough fee so transactions begin
confirmation on average within n blocks (default: 2)
-maxtxfee=<amt>    Maximum total fees (in BTC) to use in a single wallet transaction; setting
this too low may abort large transactions (default: 0.10)
-upgradewallet    Upgrade wallet to latest format on startup
-wallet=<file>    Specify wallet file (within data directory) (default: wallet.dat)
-walletbroadcast    Make the wallet broadcast transactions (default: 1)
-walletnotify=<cmd>    Execute command when a wallet transaction changes (%s in cmd is
replaced by TxID)
-zapwallettxes=<mode>    Delete all wallet transactions and only recover those parts of the
blockchain through -rescan on startup (1 = keep tx meta data e.g. account owner and payment
request information, 2 = drop tx meta data)

Debugging/Testing options:
-debug=<category>    Output debugging information (default: 0, supplying <category> is
optional). If <category> is not supplied or if <category> = 1, output all debugging
information.<category> can be: addrman, alert, bench, coindb, db, lock, rand, rpc,
selectcoins, mempool, mempoolrej, net, proxy, prune, http, libevent, tor, zmq, qt.
-gen    Generate coins (default: 0)
-genproclimit=<n>    Set the number of threads for coin generation if enabled (-1 = all cores,
default: 1)
-help-debug    Show all debugging options (usage: --help -help-debug)
-logips    Include IP addresses in debug output (default: 0)
```

```
-logtimestamps    Prepend debug output with timestamp (default: 1)
-printtoconsole    Send trace/debug info to console instead of debug.log file
-shrinkdebugfile    Shrink debug.log file on client startup (default: 1 when no -debug)

Chain selection options:
-testnet    Use the test chain

Node relay options:
-minrelaytxfee=<amt>    Fees (in BTC/kB) smaller than this are considered zero fee for
relaying, mining and transaction creation (default: 0.00001)
-limitfreerelay=<n>    Rate-limit free transactions to <n>*1000 bytes per minute (default: 15)

-bytespersigop    Minimum bytes per sigop in transactions we relay and mine (default: 20)

-datacarrier    Relay and mine data carrier transactions (default: 1)
-datacarriersize    Maximum size of data in data carrier transactions we relay and mine
(default: 83)

Block creation options:
-blockminsize=<n>    Set minimum block size in bytes (default: 0)
-blockmaxsize=<n>    Set maximum block size in bytes (default: 750000)
-blockprioritysize=<n>    Set maximum size of high-priority/low-fee transactions in bytes
(default: 0)

RPC server options:
-server    Accept command line and JSON-RPC commands
-rest    Accept public REST requests (default: 0)
-rpcbind=<addr>    Bind to given address to listen for JSON-RPC connections. Use [host]:port
notation for IPv6. This option can be specified multiple times (default: bind to all
interfaces)
-rpcuser=<user>    Username for JSON-RPC connections
-rpcpassword=<pw>    Password for JSON-RPC connections
-rpcauth=<userpw>    Username and hashed password for JSON-RPC connections. The field <userpw>
comes in the format: <USERNAME>:<SALT>$<HASH>. A canonical python script is included in
share/rpcuser. This option can be specified multiple times
-rpcport=<port>    Listen for JSON-RPC connections on <port> (default: 8332 or testnet: 18332)

-rpcallowip=<ip>    Allow JSON-RPC connections from specified source. Valid for <ip> are a
single IP (e.g. 1.2.3.4), a network/netmask (e.g. 1.2.3.4/255.255.255.0) or a network/CIDR
(e.g. 1.2.3.4/24). This option can be specified multiple times
-rpcthreads=<n>    Set the number of threads to service RPC calls (default: 4)

UI Options:
-choosedatadir    Choose data directory on startup (default: 0)
-lang=<lang>    Set language, for example "de_DE" (default: system locale)
-min    Start minimized
-rootcertificates=<file>    Set SSL root certificates for payment request (default: -system-)

-splash    Show splash screen on startup (default: 1)
-resetguisettings    Reset all settings changes made over the GUI
```

Many of the boolean options can also be set to off by specifying them with a "no" prefix: e.g. -nodnseed

Source: https://en.bitcoin.it/wiki/Running_Bitcoin

Read Getting started with bitcoin online: https://riptutorial.com/bitcoin/topic/8043/getting-started-with-bitcoin

Chapter 2: Thin Clients

Introduction

This client downloads a complete copy of the headers for all blocks in the entire block chain. This means that the download and storage requirements scale linearly with the amount of time since Bitcoin was invented.

Examples

Request a merkle block with bitcore-p2p

In this example we will ask the bitcoin network for the merkle block number 442603.

In order to do this we need to send a filterload message and then we have to send a getdata message using the inventory type MSG_MERKLEBLOCK.

The peers should reply with a merkleblock message for the requested block and a tx message for any transactions in the requested block that matched the filter.

In bitcore-p2p we need to register an event for each message's type that we want recive.

```
let Pool = require('bitcore-p2p').Pool;
let BloomFilter = require('bitcore-p2p').BloomFilter;
let NetworksData = require('bitcore-lib').Networks;
let Messages = require('bitcore-p2p').Messages;

let network = 'livenet';  // Network can be livenet or testnet
let txs = []; // Here we store the transactions
let filteredBlocks = []; // Here we store the merkleblocks

// Date that we are loocking for
let data = {
  code: '88adcf0215d5fcbca5c6532aaecffb48128cf1a6',  // 1DTh7XPb42PgCFnuMHSitMPWxCfNNFej8n in
hex fromat
  format: 'hex',
};

// Isatnciate and connect a node Pool
let pool = new Pool({network: NetworksData[network]});
pool.connect();

// Create a filter and a bitcoin message with the filter
let filter = BloomFilter.create(1000, 0.1).insert(new Buffer(data.code, data.format));
let filterLoad = new Messages({network: NetworksData[network]}).FilterLoad(filter);

// Create a bitcoin message for require a merkleblock
let blockHashRequested = '0000000000000000004f8325a66388e22c10e6de9f0f6e5809eaf1e0393efe02';
let getDataForFilteredBlock = new Messages({network:
NetworksData[network]}).GetData.forFilteredBlock(blockHashRequested);

// Transactions and merkleblock are sent in different messages
```

```
pool.on('peertx', function(peer, message) {
  txs.push({
    peer: peer,
    message: message,
  });

  console.log('Recived from: ', peer.host);
  console.log('The transaction: ', message.transaction.hash);
});

pool.on('peermerkleblock', function(peer, message) {
  filteredBlocks.push({
    peer: peer,
    message: message,
  });

  console.log('Recived from: ', peer.host);
  console.log('The merkleBlock: ', message.merkleBlock.header.hash);
});

// Wait for pool to connect
setTimeout(function(){
  pool.sendMessage(filterLoad);
  pool.sendMessage(getDataForFilteredBlock);
}, 5000);

//Recived from:  138.68.111.63
//The merkleBlock:  00000000000000000004f8325a66388e22c10e6de9f0f6e5809eaf1e0393efe02
//Recived from:  138.68.111.63
//The transaction:  9aec6cc42ddcf5900d280f3fa598f5cdb101f00614785165f777a6856314f4d9
//Recived from:  103.3.61.48
//The merkleBlock:  00000000000000000004f8325a66388e22c10e6de9f0f6e5809eaf1e0393efe02
//Recived from:  103.3.61.48
//The transaction:  9aec6cc42ddcf5900d280f3fa598f5cdb101f00614785165f777a6856314f4d9
```

Download an header chain with bitcore-p2p

IMPORTANT This is just an example code, do not use in production.

In order to download an header chain we have to send a getHeaders message.

In this example we will require as much as possible headers after the 40000th one.The peer will respond with batch of 2000 headers so, we have to take the last header hash for be able to require the next 2000 headers.

Consider the fact that bitcore-p2p is an event based library the most robust option could be promisify the network interface in order to use async/await for download the header chain. Here for the sake of simplicity we will use a generator to face the asynchronous nature of the network.

```
let Messages = require('bitcore-p2p').Messages;
let Pool = require('bitcore-p2p').Pool;
let NetworksData = require('bitcore-lib').Networks;

let network = 'livenet';
let headers = [];          // do not do that in production!
let validHeaders = [];     // do not do that in production!
let firsHeader = '000000000000000004ec466ce4732fe6f1ed1cddc2ed4b328fff5224276e3f6f';
```

```
// Isatnciate and connect a node Pool
let pool = new Pool({network: NetworksData[network]});
pool.connect();

// we have to reverese the hash becaouse is in the format xxxx0000 instead of 0000xxxx or vice
versa
function reverseHash(hash){
  return hash.match(/.{1,2}/g).reverse().join('');
}

// check if the response is the one associate with the last request becaouse could be a
response associate to an old request
function isValidResponse(firstHeaderRecived, headerHashRequested){
  // the header hash of the block before the first block header that we get on the response
  let headerHashBeforeFirstHeaderRecived =
reverseHash(firstHeaderRecived.prevHash.toString('hex'));
  if (headerHashBeforeFirstHeaderRecived === headerHashRequested){
    return true;
  }
  else{
    return false;
  }
}

pool.on('peerheaders', function(peer, message) {
  let lastHeaderHashRequested;
  if (validHeaders[validHeaders.length -1]) {
    lastHeaderHashRequested = validHeaders[validHeaders.length -1].hash;
  }
  else {
    lastHeaderHashRequested = firsHeader;
  }
  if (isValidResponse(message.headers[0], lastHeaderHashRequested) && headers.length === 0) {
    headers.push({
      peer: peer,
      message: message,
    });

    console.log('Recived from: ', peer.host, message.headers.length, 'headers');
    console.log('The first block hash is', message.headers[0].hash);
    console.log('The last block hash is', message.headers[message.headers.length - 1].hash);

    syncronize.next();
    //console.log(syncronize)
  }
});

function* sync(lastHash) {
  let startHash = new Buffer(lastHash, 'hex');
  let message = new Messages({network: NetworksData[network]}).GetHeaders();
  // require as much as possible headers after startHash
  message.starts.push(startHash);
  pool.sendMessage(message);
  yield;
  validHeaders.push(...headers[0].message.headers);
  headers = [];
  let lastDownloadedHeader = validHeaders[validHeaders.length - 1];
  if (validHeaders.length % 2000 === 0) {
    yield * sync(reverseHash(lastDownloadedHeader.hash));
  }
}

syncronize = sync(reverseHash(firsHeader));

// Wait for pool to connect
setTimeout(function(){
  console.log(pool);
  syncronize.next();
}, 5000);
```

Read Thin Clients online: https://riptutorial.com/bitcoin/topic/8154/thin-clients

Credits

S. No	Chapters	Contributors
1	Getting started with bitcoin	Community, David Ammouial, m1xolyd1an
2	Thin Clients	Fi3