一、BitCoin的Make编译

OS环境：Ubuntu-16.04

从github上下载或者fork源码，解压缩后进入目录bitcoin-master

1、更新apt-get源

sudo apt-get update

2、更新依赖库

sudo apt-get install make
sudo apt-get install gcc
sudo apt-get install g++

sudo apt-get install build-essential
sudo apt-get install libtool
sudo apt-get install autotools-dev
sudo apt-get install autoconf
sudo apt-get install pkg-config
sudo apt-get install libssl-dev

sudo apt-get install libevent-dev

sudo apt-get install libboost-all-dev

sudo apt-get install libminiupnpc-dev

sudo apt-get install libqt4-dev

sudo apt-get install libprotobuf-dev

sudo apt-get install protobuf-compiler

sudo apt-get install libqrencode-dev

sudo apt-get install libdb-dev

sudo apt-get install libdb++-dev

3、开始编译

./autogen.sh

./configure或者./configure -with-incompatible-bdb

make

sudo make install

安装完毕后，会将编译后的二进制程序文件都生成到/usr/local/bin目录下

二、代码修改点

1、创世区块、区块大小、出块间隔、难度调整周期、每区块奖励

文件 /src/chainparams.cpp

这个类是主链的参数配置

```
修改类 class CMainParams : public CChainParams {
```

修改属性：

这是产量减半的区块数

bitcoin是每210000个区块产量减半

```
consensus.nSubsidyHalvingInterval = 210000;
```

初始难度值

这是一个256位的难度值，bitcoin中这个难度值表示难度为1

```
consensus.powLimit = uint256S("00000000ffffffffffffffffffffffffffffffffffffffffffffffffffffffff");
```

难度目标调整，bitcoin中每两个星期调整一次

```
consensus.nPowTargetTimespan = 14 * 24 * 60 * 60; // two weeks
```

两个星期的理论区块数
```
consensus.nMinerConfirmationWindow = 2016
```

区块生产的时间间隔，bitcoin 中是控制在 10 分钟，也就是 600 秒
```
consensus.nPowTargetSpacing = 10 * 60;
```
调整为：1 * 60
也就是 60 秒一个区块

网络协议端口号
```
nDefaultPort = 8333;
```

构建创世区块

其中的50是第一个周期的初始产量
```
genesis = CreateGenesisBlock(1231006505, 2083236893, 0x1d00ffff, 1, 50 * COIN);
```

上述的类是主链的设置，接下来是测试链的设置
```
class CTestNetParams : public CChainParams {
```

参数修改的点与上同

修改区块大小限制
/src/**validation.cpp**
函数名

bool CheckBlock(const CBlock& block, CValidationState& state, const Consensus::Params& consensusParams, bool fCheckPOW, bool fCheckMerkleRoot)

{

在这个方法有一个调用，如下

// Size limits

if (block.vtx.empty() || block.vtx.size() * WITNESS_SCALE_FACTOR > MAX_BLOCK_WEIGHT || ::GetSerializeSize(block, SER_NETWORK, PROTOCOL_VERSION | SERIALIZE_TRANSACTION_NO_WITNESS) * WITNESS_SCALE_FACTOR > MAX_BLOCK_WEIGHT)

return state.DoS(100, false, REJECT_INVALID, "bad-blk-length", false, "size limits failed");

这一段是限制大小的

我们要将现有的 1M 大小限制修改为 8M

src/consensus/**consensus.h**

```
static const unsigned int MAX_BLOCK_WEIGHT = 4000000;
```

将值修改为 8000000

6、地址格式

保持与比特币的不变

6、RPC 端口号（主链与测试链）

RPC 端口定义为 8632　　测试链为 18632

源码路径	修改点	备注
rpc/server.cpp	HelpExampleRpc方法	其中的8332端口号修改

contrib/linearize/example-linearize.cfg	#mainnet default
	port=8332
	
	#testnet default
	#port=18332
		主链和测试链的端口号修改

doc/**REST-interface.md**	文档中的8332，18332，18443	端口号修改
contrib/linearize/README.md	8332	端口号修改
src/chainparamsbase.cpp	/** * Main network */ class CBaseMainParams : public CBaseChainParams { public: CBaseMainParams()	

```
    {
        nRPCPort = 8332;
    }
};


/**
 * Testnet (v3)
 */
class CBaseTestNetParams : public CBaseChainParams
{
public:
    CBaseTestNetParams()
    {
        nRPCPort = 18332;
        strDataDir = "testnet3";
    }
};



/*
 * Regression test
```

	*/
	class CBaseRegTestParams : public CBaseChainParams
	{
	public:
	CBaseRegTestParams()
	{
	nRPCPort = 18443;
	strDataDir = "regtest";
	}
	};	

contrib/linearize/**linearize-hashes.py**	if 'port' not in settings: settings['port'] = 8332	
doc/man/**bitcoin-cli.1**	.IP Connect to JSON\-RPC on <port> (default: 8332 or testnet: 18332)	
contrib/debian/examples/**bitcoin.conf**	# Listen for RPC connections on this TCP port: #rpcport=8332	
contrib/rpm/**bitcoin.spec**	出现8332和18332的地方都替换 18443\18444也一并替换成其他端口号	

doc/release-notes/**release-notes-0.10.0.md**	出现8332和18332的地方	
doc/man/**bitcoin-qt.1**	.IP Listen for JSON\-RPC connections on <port> (default: 8332 or testnet: 18332)	
doc/man/**bitcoind.1**	.IP Listen for JSON\-RPC connections on <port> (default: 8332 or testnet: 18332)	

src/rpc/**server.cpp**	std::string HelpExampleRpc(const std::string& methodname, const std::string& args)
	{
	return "> curl --user myusername --data-binary '{\"jsonrpc\": \"1.0\", \"id\":\"curltest\", "
	"\"method\": \"" + methodname + "\", \"params\": [" + args + "] }' -H 'content-type: text/plain;' http://127.0.0.1:8332/\n";
	}	

doc/release-notes/release-notes-0.12.0.md doc/developer-notes.md doc/release-notes/release-notes-0.14.0.md	文档内容中，出现8332和18332的都替换掉	

doc/release-notes/release-notes-0.15.0.md		

7、bitcoin 协议端口号（主链与测试链）

协议端口定义为 8633　测试链为 18633

源码路径	修改点	备注
contrib/seeds/nodes_main.txt	代码中出现的8333都修改掉	这是种子节点的地址，对我们来说是不需要的，不过先把端口号改掉
src/**chainparamsseeds.h**	这里面是IPV6格式的种子节点地址，	
src/test/**addrman_tests.cpp**	代码中出现的所有8333	
contrib/qos/README.md	文档中出现的8333	
/contrib/qos/**tc.sh**	代码中的8333	

/src/test/**netbase_tests.cpp**	代码中的8333	
doc/tor.md	文档中的8333和18333	
test/functional/proxy_test.py	代码中的8333和18333	
src/test/net_tests.cpp	代码中的8333和18333	
contrib/debian/examples/**bitcoin.conf**	文档中的8333和18333	
contrib/seeds/generate-seeds.py	代码中的8333	

src/rpc/**net.cpp**	UniValue disconnectnode(const JSONRPCRequest& request)
	{
	if (request.fHelp || request.params.size() == 0 || request.params.size() >= 3)
	throw std::runtime_error(
	"disconnectnode \"[address]\" [nodeid]\n"
	"\nImmediately disconnects from the specified peer node.\n"
	"\nStrictly one out of 'address' and 'nodeid' can be provided to identify the node.\n"
	"\nTo disconnect by nodeid, either set 'address' to the empty string, or call using the named 'nodeid' argument only.\n"
	"\nArguments:\n"
		代码中的8333

	"1. \"address\" (string, optional) The IP address/port of the node\n" "2. \"nodeid\" (number, optional) The node ID (see getpeerinfo for node IDs)\n" "\nExamples:\n" + HelpExampleCli("disconnectnode", "\"192.168.0.6:8333\"") + HelpExampleCli("disconnectnode", "\"\" 1") + HelpExampleRpc("disconnectnode", "\"192.168.0.6:8333\"") + HelpExampleRpc("disconnectnode", "\"\", 1"));	
contrib/rpm/bitcoin.spec	8333和18333	
doc/man/**bitcoin-qt.1**	8333和18333	
doc/man/bitcoind.1	8333和18333	
src/**chainparams.cpp**	`nDefaultPort = 8333;`	
doc/developer-notes.md	文档中的8333	

7、Restful 端口（主链与测试链）

18443 改为 18643
18444 改为 18644

8、修改/src/**chainparams.cpp**

```
class CMainParams : public CChainParams {
```
类中的属性
```
nDefaultPort = 8333; 修改为 8633
```

```
class CTestNetParams : public CChainParams {
```
类中的属性
```
nDefaultPort = 18333; 修改为 18633
```

```
class CRegTestParams : public CChainParams {
```
类中的属性
```
nDefaultPort = 18444; 修改为 18644
```

修改 src/**chainparamsbase.cpp**

```
class
CBaseMainParams :
public
CBaseChainParams
                    {
                    public:
                        CBaseMainParams()
                        {
                            nRPCPort = 8332;
                        }
                    };
```

修改：将 8332 改为 8632

```
class
CBaseTestNetParams :
public
CBaseChainParams
                        {
                        public:
                            CBaseTestNetParams()
                            {
                                nRPCPort = 18332;
                                strDataDir = "testnet3";
                            }
                        };
```

修改：将 18332 改为 18632

```
/*
 * Regression test
 */
class CBaseRegTestParams : public CBaseChainParams
{
public:
    CBaseRegTestParams()
    {
        nRPCPort = 18443;
        strDataDir = "regtest";
    }
};
```

修改：18443 改为 18643

三、CPUMiner

使用https://github.com/pooler/cpuminer

1、安装依赖

sudo apt-get install curl libcurl3 libcurl3-dev

到http://www.digip.org/jansson/　下载

然后进入到jansson-2.10目录

执行命令

./configure

make

make check

make install

2、编译安装cpuminer

```
./autogen.sh  # only needed if building from git repo
./nomacro.pl  # in case the assembler doesn't support macros
./configure CFLAGS="-O3" # Make sure -O3 is an O and not a zero!
make
```

3、运行minerd
如果出现错误提示:

minerd: error while loading shared libraries: libjansson.so.4: cannot open shared object file: No such file or directory

按照以下命令执行:

1）、whereis libjansson

显示：`libjansson: /usr/local/lib/libjansson.a /usr/local/lib/libjansson.la /usr/local/lib/libjansson.so`

自己的`libjansson.so`所在位置为:`/usr/local/lib/libjansson.so`

2）、cd /usr/local/lib

查询libjansson文件是否真的存在

3）、创建软连接

```
ln -s /usr/local/lib/libjansson.so /usr/lib/libjansson.so.4
```

4）、重新加载库

```
ldconfig
```

5）、安装后使用

默认安装在/usr/local/bin目录中，可以执行命令测试一下是否成功

minerd --version

minerd --help

bitcoind 启动后
curl -H "Content-Type: application/json" -d '{"username":"nihao","password":"457"}' http://127.0.0.1:8332

执行后没什么错误提示就是连接成功了

三、fork事项目录

1、core 的修改，区块生产时间间隔、一次奖励数额，难度调整周期、区块大小限制等，如上所述
2、网络端口号和 RPC 端口号
3、快照程序，获得到指定高度的所有账户的余额
4、挖矿程序（调试矿池，部署矿池后，连接挖矿程序开挖，挖矿测试测试的时候可以使用 minerd）
5、重放保护
    1、修改交易事务签名部分，加入新的标志和版本号
    2、区块同步时的校验，只能同步接收带有我们重放标志的事务
    3、交易事务入池校验（符合规则的才放入内存池）
    4、矿池程序加入校验（只接收符合规则的交易事务）
    如果时间来得及，可以在矿池程序校验部分，除了判断仅接收符合规则的事务，还可以加入一个分叉区块高度的判断，这样可以兼容之前的交易事务，也就是 bitcoin 的事务，这个看情况再加吧，属于人性化设置）

四、计算脚本

1、比特币的发行总量计算

```
#区块初始的奖励数量
start_block_reward = 50
#奖励减半的区块数间隔
reward_interval = 210000

def max_money():
        # 50 BTC = 50 0000 0000 Satoshis
        #初始奖励的聪数（聪为比特币的最小计量单位）
        current_reward = 50 * 10**8
        total = 0
        while current_reward > 0:
                total += reward_interval * current_reward
                current_reward /= 2
        return total

print "Total BTC to ever be created:", max_money(), "Satoshis"
```