在 Linux 中安装、配置和使用比特币节点

José Ulloa Araya
joseeduardo.ulloa24@estudiantes.uva.es ORCID: 0009-0001-2385-6990

Diego R. Llanos
diego.llanos@uva.es ORCID: 0000-0001-6240-9109

巴利亚多利德大学信息工程学院

2025年5月

arXiv:2601.09748v1 [cs.CR] 12 Jan 2026

摘要

本文记录了在 Linux 环境中安装、配置和运行比特币完整节点的过程，从手动编译源代码到与网络完全同步。文中介绍了这一过程的技术阶段，分析了 Bitcoin Core 生成的主要文件，并通过实证研究考察了 txindex、prune、dbcache、maxmempool 和 maxconnections 等参数的影响。本文还记录了区块下载（IBD）机制运行期间的系统资源使用情况，并阐述了各项资源在运行中的重要性。本文为今后研究比特币节点性能，或开发用于查询区块链数据的工具，奠定了坚实基础。

关键词： 比特币 Core、区块链、全节点、Linux、初始区块下载、系统性能、配置参数。

1. 引言

比特币是一种去中心化网络，无需借助中介即可通过基于节点的分布式技术架构进行交易。每个完整的比特币节点都会验证、存储并传播符合协议规则的区块和交易，从而构成系统完整性与安全性的一部分。运行自己的节点不仅可以参与网络，还能直接访问区块链数据，从而在不依赖第三方的情况下验证信息。

本文旨在帮助读者理解 Linux 环境中完整比特币节点的运行方式，并记录 Bitcoin Core 软件的安装、配置和运行过程。

关于本研究的总体目标，我们希望详细记录整个过程，重点关注运行期间生成的文件，以及不同配置参数对节点性能和运行状况的影响。为此，我们将分析 Bitcoin Core 生成的文件系统，评估 txindex、prune、dbcache、maxmempool 和 maxconnections 等重要配置，并编写一份指南，介绍如何在家庭环境中安装和基本运行节点。观察的变量包括系统资源的使用情况、同步时间，以及每种配置所带来的查询和分析能力。

总之，本研究表明，只要正确配置相应设置，并充分了解各种配置选项所带来的影响，就可以在本地环境中使用适量的机器资源安装并运行比特币全节点。

2. 什么是比特币节点´，以及它在网络中的作用是什么

首先需要了解，比特币是一个去中心化网络，旨在实现对等方之间的价值转移，而无需任何类型的中介。节点是这一系统中的关键要素，负责验证协议规则、存储区块链的全部历史记录，并向其他用户传播信息。在 Carlos Callejo 和 V´ıctor Ronco 所著的《傻瓜式加密货币》中，他们指出：“每一台这样的计算机设备都是一个节点”，并且“所有节点彼此连接，形成一个稳固的网络，每个节点都保存着一份部分或完整的交易历史副本”[2, 第 18]页。简单来说，它们就是彼此连接的计算机，共同构成一个网络，其中每台计算机都保存着一份副本。

即使 Andreas M. Antonopoulos 也在《*Mastering Bitcoin*》一书中指出，全节点不´仅存储区块链，还“通过积极验证交易和区块来参与其中，确保网络的完整性和一致性”[1, 第 141] 页。因此，每台计算机（节点）都有能力拒绝不符合规则的区块或区块集合，即使它们是由计算能力强大的参与者构建的。

图1展示了比特币网络运行方式的可视化示意图。其中可以看到用户、节点、矿工、钱包以及交易流。



图1：比特币网络运行方式的可视化示意图。发表于《*Mastering Bitcoin*》的 信息图，Antonopoulos（2019），第24页。

2.1. 一个简单易懂的示例：学校成绩单 .

前面的技术性语言可能听起来相当复杂，因此我们通过这个例子来帮助理解比特币是什么：想象一个学校系统，所有班级都有一位老师，每位老师都保存着一份所有学生成绩单的完全相同副本。如果一位老师想在成绩单上记下某些内容，就必须先与其他老师核实。只要所有老师一致认定学生确实

参加了考试，而且拟定的成绩有效，就会将成绩记入成绩单；反之，如果成绩单上存在分歧，例如其他老师觉得不可信，就会拒绝这项成绩变更。比特币网络中的节点也是如此运作的：它们彼此验证，以共同验证一本公共账本，而无需中央权威机构。

2.2. 节点的基本属性

在《比特币 ˜小书》中，书中指出：“世界各地有数千名验证者，他们会验证比特币区块链，并存储比特币交易的完整历史记录。这些验证者被称为‘全节点’”[3, 第 84], 页。因´此，简单来说，这些全节点保障了去中心化（因为不存在单一的控制点）、安全性（因为数据验证是通过分布式过程进行的）、透明性（所有参与方都可以查看交易的公开历史记录）和抗审查性（如果没有共识，任何节点都无法强行推行更改）。

2.3. 节点与矿工的区别

这可能是一个常见的问题，因为有些人会以为所有节点也都是矿工，但事实恰恰相反：矿工负责提出新区块，并争夺验证奖励；而全节点则负责验证提出的区块是否符合协议规则。在《傻瓜式加密货币》中，对这一差异有很好的说明：“节点只存储信息；矿工则‘工作’以验证交易。”[2, 第 21] 页。

2.4. 为什么要运行全节点

运行比特币全节点是参与数字主权的一种切实方式；对此，《比特币小书》指出：“每位用户都可以通过运行全节点来验证一切，而不必信任其他任何人。”[3, 第 98-99] 页。如今，在家中安装和维护节点的技术门槛很低，即便是经验最少的用户，也能增强网络，并以机械方式验证自己的余额和交易，而无需依赖第三方。这正是本文试图证明的事情。

2.5. 比特币作为分布式基础设施

最后，在《*Mastering Bitcoin* 》一书中指出：“比特币也是协议、点对点网络和分布式计算创新的名称”[1, 第 13] 页。每个节点都是该网络的一部分，而这个网络的存在确保比特币始终保持其原本的目标：成为一种不受中央控制的电子货币系统。

3. 在 Linux 系统上安装和配置 Bitcoin Core

在 GNU/Linux 机器上安装比特币全节点，是理解这一去中心化网络运作方式不可或缺的学习体验。安装并非只是运行软件，而是一个伴随着技术抉择的过程，这些抉择会直接影响最终系统的鲁棒性、安全性和可扩展性。此处记录的是 Bitcoin Core 29.0 版的安装过程：从源代码编译，并配置为系统服务。

3.1. 手动编译的理由

虽然当时已经 有编译好的 Bitcoin Core 版本，但还是决定从源代码编译，而不是从某个版 o源，而不是某个 v 预编译版本有三个根本原因：

[1]在比特币的语境中 ，当我们说“公共账本”时 ，我们指的是区块链

- 可以确切了解并控制启用了哪些配置选项，例如钱包支持（wallet）、完整交易数据库（txindex）[2]，或通过 ZeroMQ 发送通知[3]可以更好地兼容操作系统，避免依赖预编译二进制文件。这符合项目从基础入手研究比特币应用架构的目标。
-
-

3.2. 系统准备与专用环境

为此，我们使用了一台配备英特尔酷睿 i5-8400 处理器的计算机，拥有 6 个核心和 16 GB 内存，并预留了一块 2 TB 磁盘用于存放区块链。我们将部分系统挂载在目录$/opt/BLOCKCHAIN$下，并仅授予 bitcoin 用户访问权限。此隔离的目的如下：

- 便于管理数据量（目前超过 500 GB）。通过将区块链与操作系统分离来提升读写性能。
- 确保将来能够迁移和/或备份节点。
-

3.3. 下载并编译源代码

我们选择了 $bitcoin\ v29.0$，直接从 GitHub 官方代码库下载，且未进行任何修改。我们根据比特币代码库的最新官方文档，使用了基于 CMake 的构建工具：

$ git clonehttps://github.com/bitcoin/bitcoin.git $ cd
bitcoin $ git checkout v29.0

已安装所需的依赖项，优先选择 Ubuntu 官方软件仓库中提供的软件包：

$ sudo apt install build-essential cmake pkgconf python3 libevent-dev \
libboost-dev libsqlite3-dev libzmq3-dev

构建环境的配置通过以下命令完成：

$ cmake -Bbuild -DWITH_SQLITE=ON -DENABLE_WALLET=ON -DWITH_
ZMQ=ON$ cmake --build build -j$(nproc)$ sudo cmake --install build

通过这些选项，启用了对 SQLite 数据库钱包的支持、对 bitcoin-cli 的支持，以及访问 ZMQ 事件的支持。

[2]此选项允许仅使用其交易 ID[3]查询旧交易。ZeroMQ 是 Bitcoin Core 中用于异步发送消息的库，可针对事件发出实时通知；如需了解更多详情，可查阅官方文档或在终端中使用。

3.4. 节点配置与安全性

在 /etc/bitcoin/bitcoin.conf 中创建´了一个配置文件，其中包含图*2*所示的参数´。

```
交易索引=1
数据目录=/opt/BLOCKCHAIN
rpcauth=bitcoin:<生成的哈希值>
```

图2：/etc/bitcoin/bitcoin.conf 配置

txindex 选项$=1$ 很重要，因为它可以查询区块链中的任意交易，而不仅仅是与自有地址[1, p. 56], 相关联的交易。这对于之后开发可视化工具来说非常重要。

rpcauth 字段由脚本 $rpcauth.py$ 生成（该脚本包含在从 GitHub 下载的 Bitcoin 软件包中），可为分析工具提供安全的 RPC 身份验证。

3.5. 作为系统服务运行

为确保节点持久运行并便于管理，我们将其配置为 systemd 服务，因此创建了一个配置文件，如图*3*所示。

```
[单元]
描述=比特币守护进程
位于=network.target之后

[服务]
ExecStart=/usr/local/bin/bitcoind -conf=/etc/bitcoin/bitcoin.conf \
   -pid=/run/bitcoind/bitcoind.pid
User=bitcoin
Group=bitcoin
Type=forking
进程标识符文件=/run/bitcoind/bitcoind.pid
重启=失败时
运行时目录=bitcoind

[安装]
期望的目标=multi-user.target
```

图3：比特币守护进程配置

为了启动时自动重启

在系统启动时自动启动节点。此外，还可以´重新启动它，使其在发生任何´故障，甚至可以通过 systemctl 管理它。

3.6. 初步验证

节点已成功运行，并开始与比特币网络同步。通过以下命令验证其是否正常运行：

```
$ sudo systemctl status bitcoind
```

要检查同步状态，可以使用此命令：

```
$ bitcoin-cli getblockchaininfo
```

区块链的完整同步耗时约 23 小时，下一章将对此进行更深入的介绍，

4. 同步过程说明

上一章中已完成配置，并通过 systemd 将节点作为系统服务运行后，便开始下载比特币网络中的区块（这一过程称为初始区块下载），该过程负责逐一下载并验证区块，使比特币机器能够完整、独立地运行。

该节点从零开始同步。为了尽可能精确地掌握进度，我编写了一个小型 Bash脚本，每隔 30 分钟通过 bitcoin-cli 查询客户端，并将区块高度和 verificationprogress 的值存储在 CSV 文件中。verificationprogress 表示区块链验证进度的百分比。

处理于4月29日05:54开始，起始区块为304263。4月30日05:00评估结果时，达到的高度为894549，因此当时决定停止脚本运行，这意味着在约23小时内下载并验证了590,000个区块。

4.1. 流程的演变与结果

在此过程中，verificationprogress变量持续稳步上升，如图4所示。该变量的取值范围为0到1，用于指示同步进度；其估算值综合考虑了网络近期活动和已验证历史记录。

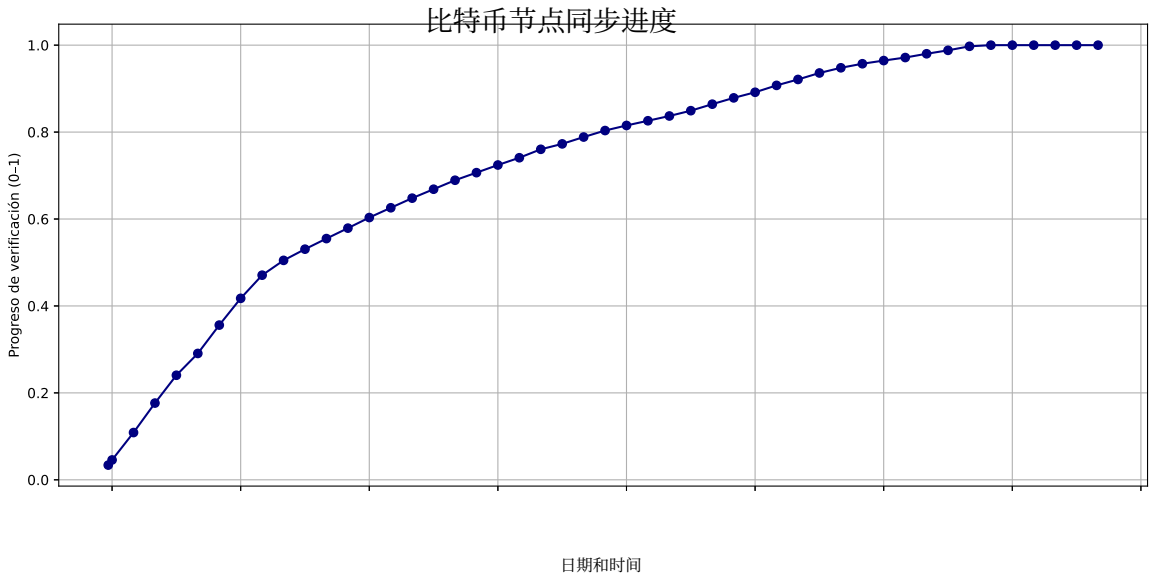


图4：初始下载过程中比特币节点的同步进度。

最初几个小时一切都在持续进行，没有中断；如图 4 所示，区块下载和验证的频率不断增加。到4月30日凌晨，verificationprogress 的值已接近 1，尽管同步尚未完成。直到 05:00，节点才达到 894549 个区块和区块头的总高度，这是节点完成初始区块下载状态的必要条件，符合将初始区块下载（initialblockdownload）设为 false 的初始条件。

这一点通过以下命令得到了验证：

bitcoin-cli getblockchaininfo | grep 初始区块下载

4.2. 相关事件讨论

在执行接近尾声时出现了一个特殊情况：verificationprogress 的验证进度甚至趋于稳定（或轻微波动），尽管区块和区块头实际上已基本持平，这可能会引起一些担忧，让人以为同步过程中出现了错误。但这符合进度估算算法（Bitcoin Core 自身的一部分）的运作方式：它会根据最新区块和网络近期活动，不断重新校准估算状态。

节点通过同时满足两个条件，达到了最终运行状态：一方面，区块与区块头一致；另一方面，对区块链本身进行了充分的纠错验证。因此，系统认为初始化过程已完成，也就是说，初始区块下载为 = false。

4.3. 自动化监控的价值

使用监控脚本，可以通过实证检验节点的性能，并在一段合理的时间内，以足够详尽的程度记录整个过程。这不仅能够验证系统是否正常运行，还提供了数据，证明在硬件和网络条件适宜、配置正确的情况下，全节点可以在不到 24 小时内完成同步。

这种监控方式在研究场景中尤其 有用，因为它可以比较不同的情形、识别可能出现瓶颈的地方，甚至查明启用交易索引=1 或修剪模式等选项的影响。不过，正如第三章所指出的，同步 仅通过不修剪选项完成，以便将全部区块历史记录存储在磁盘上。

5. 比特币核心生成的文件分析

比特币节点完成初始同步后，我们将分析配置中指定的数据目录（在我们的案例中为 数据目录=/*opt/BLOCKCHAIN*）。该目录中逐渐生成了许多文件和子目录，它们代表了节点的状态以及使节点正常运行的各种内部进程。

为了有序地检查文件，选择执行命令 *ls -lhR/opt/BLOCKCHAIN*，以获取 Bitcoin Core 已生成的所有文件的完整递归列表。该列表被作为起点，用于识别、分类和研究可能有意义的文件与文件夹。

所有结构，无论是用于存储区块和交易状态的结构，还是调试日志和网络参数，都具有某种特定功能。本章旨在对这些结构进行检查，重点关注每个结构的技术用途。

在*/opt/BLOCKCHAIN* 路径中检测到的主要文件和目录如图*5*所示。

```
/opt/BLOCKCHAIN
├── banlist.json
├── 区块目录/
│   └── 索引目录/
├── chainstate/
├── debug.log
├── 费用_estimates.dat
├── indexes/
│   └── 交易索引/
├── peers.dat
└── settings.json
```

图5：/opt/BLOCKCHAIN 中的文件和子目录结构。

5.1. ´banlist.json 文件分析

文件 *banlist.json* 由 Bitcoin Core 自动生成和管理，用于记录节点手动或自动封锁的 IP 地址或网络范围（具体取决于是否存在恶意或可疑行为）。此机制可保护节点免受攻击或不受欢迎的连接。

如图 *6*所示，被禁止的网络 字段为空，说明截至目前，该节点认为没有必要屏蔽任何地址。此外，警告指出，在节点运行期间不得手动编辑此文件，因为所做的更改可能会被覆盖。 - -

```
{
  "_warning_": "This file is automatically generated and updated by Bitcoin
      Core. Please do not edit this file while the node is running, as any
      changes might be ignored or overwritten.",
  "banned_nets": []
}
```

图 6：banlist.json 的内容

执行*stat /opt/BLOCKCHAIN/banlist.json*命令后，确认文件大小为 221 字节，最后修改时间为 ´ 2025年4月29日，这表明此后文件未被修改。这些信息证实，其当前状态与 Bitcoin Core 在节点首次运行时创建的基础文件一致。

5.2. 区块目录的分析及其索引

对*/opt/BLOCKCHAIN/blocks*目录进行了分析，该目录包含区块的二进制文件及其相应的索引。为此，使用以下命令列出了主文件夹的内容：

$ ls -lh /opt/BLOCKCHAIN/区块目录

随后，使用以下命令查看该文件夹占用的总空间：

$ du -sh /opt/BLOCKCHAIN/区块目录

结果显示其大小约为*471 GiB*。因此可以确认，该节点应已完整存储区块链，且未使用任何类型的修剪（修剪*=0*）。

此外，还通过以下命令分析了所有 *.dat* 文件的元数据：

$ stat /opt/BLOCKCHAIN/区块目录/*.dat

这使我们能够确认，文件是由用户 比特币创建和修改的，且权限仅限该用户（模式 *0600*）。日期显示文件是逐步创建的，这与同步的时间顺序进展一致。

随后，为了确定该文件夹中的文件确切数量，执行了：

$ ls /opt/BLOCKCHAIN/区块目录 | wc -l

确认其中存在超过 9,000 个文件（包括 blk 和 rev 文件）。最后，开始分析子目录 index/，其中包含 LevelDB 数据库（负责为区块建立索引，以便高效访问）：

$ ls -lh/opt/BLOCKCHAIN/blocks/index $ du -
sh/opt/BLOCKCHAIN/blocks/index $ file /
opt/BLOCKCHAIN/blocks/index/*

结果显示其中有一组二进制文件，例如 MANIFEST、LOG 和多个 *.ldb 文件，这清楚地表明该节点已完整构建，同时还可通过区块哈希进行快速查询。

5.3. ´chainstate/ 目录与 UTXO 集分析

chainstate 目录用于存储未花费的交易输出[4]„ 也称为 UTXO，可快速验证新交易，确保输入来自合法且未花费的输出。[1,第 121] 页。

chainstate/ 文件夹的结构经过优化，采用 LevelDB 数据库，显著提升了 UTXO 集的读写操作效率。以 000003.log、CURRENT、MANIFEST-000002 等为前缀的文件均在 LevelDB 中进行管理，因此不得进行任何修改´，因为任何更改´都可能导致数据损坏´，最终迫使用户重建 UTXO´索引。

为分析当前 UTXO 集的大小，使用了：

$ bitcoin-cli gettxoutsetinfo

如图 7所示，该图反映了第 847,254 个区块中的 UTXO 集状态：数据库在磁盘上的大小约为 447 MB，包含超过 8,200 万个未花费输出。在所显示的数值中，变量 bogosize 用作理论值，用于估算未经压缩的 UTXO 集逻辑大小；另一方面，total amount 字段表示未花费输出所代表的 BTC 数量。简单来说，这就像对比特币网络中当前可用资金状态生成一份快照：每当添加新区块时，数据库就会更新新的输出；如果这些输出被花费，则会被删除，从而使节点能够高效运行，避免必须遍历整个区块链来验证交易。

[4]UTXO 集 nto 未花费交易输出（UTXO），代表可用余额在的 red.

```
{
"height": 847254,
"bestblock": "00000000000000000020
    de7d6b77d65d3a8370cce7be65f6c123b4c26950140",
"transactions": 89805073,
"txouts": 82454865,
"bogosize": 6169726781,
"hash_serialized_2": "
    f0e849efca8c9f01d0e3969e0de21b4525a9d85aa8aaee297e1a21e452d4498d",
"disk_size": 447277290,
"total_amount": 19975766.22532526
}
```


图 7：查询到的 UTXO 集状态

5.4. 分析文件*debug.log*

日志文件包含与软件初始化、配置读取、与对等节点的连接、区块接收以及节点整体维护相关的全部事件。因此，它被视为持续检查节点状态的关键资源，以便及时发现潜在故障。

对于当前正在进行测试的节点，该文件大小约为 210 MB，共有 909.217 行，创建于2025年4月29日，最后修改时间为撰写本节的当天（2025年5月23日）。这意味着该节点自启动以来一直在持续运行。

图图*8* 展示了文件开头几行的摘录，可以清楚地看到节点已正确初始化（同时从配置文件加载了参数）。

```
2025-04-29T03:37:20Z Bitcoin Core version v29.0 (release build)
2025-04-29T03:37:20Z Using the 'sse4(1way),sse41(4way),avx2(8way)'
SHA256 implementation
2025-04-29T03:37:20Z Using RdSeed as an additional entropy source
2025-04-29T03:37:20Z Using RdRand as an additional entropy source
2025-04-29T03:37:20Z Using data directory /opt/BLOCKCHAIN
2025-04-29T03:37:20Z Config file: /etc/bitcoin/bitcoin.conf
2025-04-29T03:37:20Z Config file arg: daemon="1"
2025-04-29T03:37:20Z Config file arg: txindex="1"
2025-04-29T03:37:20Z Command-line arg: conf="/etc/bitcoin/bitcoin.conf"
2025-04-29T03:37:20Z Using at most 125 automatic connections
```

图8：debug.log 文件的初始片段。

在文件的最后几行中，如图 *9*所示，可以确认该节点已完全同步，正在接收新区块，并主动连接到网络中的其他节点；此外，自运行以来，没有出现表明该节点运行异常的错误或警告。

```
2025-05-23T01:58:06Z UpdateTip: new best=000000000000000000017259...
2025-05-23T02:01:55Z New block-relay-only v1 peer connected
2025-05-23T02:02:33Z Saw new header hash=00000000000000000076f0...
2025-05-23T02:22:42Z UpdateTip: new best=000000000000000000001cc8...
```

图9：debug.log 文件的最新片段。

5.5. 分析文件 *fee estimates.dat*

文件*feeestimates.dat* 保存了节点用于估算在指定数量的区块内确认交易所需手续费的统计数据。虽然该文件以二进制形式存储，无法直接读取，但可以通过*bitcoin-cli*间接管理。此外，为了大致了解该文件的最近修改和创建情况，使用了以下命令：

$ stat 命令 /opt/BLOCKCHAIN/fee_estimates.dat

这表明该文件大小约为 250 KB。文件创建于节点启动当天，此后其结构未曾改变，但这并不意味着其内容没有随时间推移而被 Bitcoin Core 定期更新；只是期间没有发生重启，也没有重新生成该文件。通过以下命令检查日志*debug.log* ，证实了这一更新活动：

$ grep fee_estimates.dat /opt/BLOCKCHAIN/debug.log

命令执行后得到的信息表明，节点大约每小时会将内容写入此文件一次，其中包括符合“将费用估算写入 ***fee estimates.dat***”这一模式的条目。这意味着系统会根据网络近期的变化来存储费用估算。

为了验证该文件的用途，使用了 bitcoin-cli estimatesmartfee 命令。该命令会访问已存储的统计数据，并根据希望交易在多少个区块内得到确认，提供手续费建议。例如，执行：

$ bitcoin-cli estimatesmartfee 1

节点建议采用适合快速确认的费率，使交易在接下来的两个区块内得到确认（相当于执行了 *bitcoin-cli estimatesmartfee 2*），结果如图 ***10***所示。


```
{
  "费率":      0.00003145 ,
  "区块":     2
}
```


图 10：前 2 个区块的手续费。

此外，还要验证系统在执行以下命令时是否会限制超出允许范围的值：

$ bitcoin-cli estimatesmartfee 0

这证明系统能够正确限制超出允许参数范围的值，因为像 estimatesmartfee 0 这样的越界调用会返回错误消息，指出该值必须介于 1 和 1008 之间，如图 ***11***所示。


```
error code: -8error message: 无效的确认目标，必须
介于 1 和 1008 之间
```


图 11：限制值的使用

因此，这一切都证实了*fee estimates.dat*是一个重要文件，节点依靠它来维护与当前网络状况相符的最新费率建议。

5.6. 分析 ´索引目录 *indexes/* 和 ´索引 *txindex/*

文件夹 *indexes/* 包含辅助数据，可让人快速访问区块链中的特定数据。该文件夹中尤为重要的是 *txindex*索引目录，它允许根据交易 ID 检索任何交易，而无需知道包含该交易的区块。要实现这一点，需要通过 bitcoin.conf 文件启用*txindex=1* 选项，这样节点便会开始创建一个数据库，额外存储区块链上所有现有的交易；该索引目录位于 *indexes/txindex/*，可使用以下命令查询任何交易：

```
$ bitcoin-cli getrawtransaction <交易 ID> 1
```

启用该 ´索引会增加磁盘占用，同时也会大幅延长首次同步的启动时间。尽管如此，对于需要更高效地访问任意交易的工具、区块浏览器和脚本而言，使用它至关重要。

所分析节点的 ´索引目录 *indexes/* 占用约 58 GB，其中几乎全部集中在 *txindex/*，此外还存储着数千个 .ldb 文件，每个文件平均大小为 34MB。此索引功能还使开发者和研究人员能够直接访问比特币网络的历史数据，从而开展统计研究、交易重建，并与其他来源进行交叉验证。

5.7. peers.dat 文件分析

peers.dat 文件存储了比特币节点所知对等节点（peers）的持久信息。它很有用，因为可以保存一份节点列表，客户端启动时会尝试重新连接这些节点；这也有助于建立稳定的比特币 P2P 网络连接。

再次使用 stat 命令分析该文件，其大小约为 2.6 MB。该文件也在节点首次启动时创建，自那以后一直保持更新；分析 debug.log 中 peers.dat 的相关记录，未发现外部修改的迹象。

```
$ stat 命令 /opt/BLOCKCHAIN/peers.dat $ grep
peers.dat /opt/BLOCKCHAIN/debug.log
```

为了分析其内容，使用了以下命令（该命令会返回一个 JSON 对象列表，其中包含每个已连接对等方的详细信息）：

```
$ bitcoin-cli getpeerinfo | jq .
```

在图*12* 中可以看到部分获取的信息，例如，结果显示该节点与 10 个对等节点保持连接。每个条目都包含以下信息：id（节点为每个连接分配的本地标识符）、addr（远程对等节点的 IP 地址和端口）、network（对等节点使用的网络类型）、servicesnames（对等节点的能力）、relaytxes（指示对等节点是否中继交易）、pingtime 和 minping（´有助于评估网络延迟）、subver（对等节点运行的软件版本）、bytessent 和 bytesrecv（已发送和接收数据的累计量，以字节为单位），以及 connection type（已建立的连接类型）。

```
[
  {
    "id": 126,
    "地址": "66.37.25.18:8333",
    "绑定地址": "157.88.123.154:55228",
    "addrlocal": "157.88.123.154:55228",
    "网络": "ipv4",
    "服务": "0000000000000c09",
    "服务名称":[
      "网络",
      "见证者",
      "网络有限",
      "P2P V2"
    ],
    ...,
    "每条消息发送的字节数": {
      ...
    },
    "bytesrecvpermsg": {
      ...
    },
    ...
  },
  ...
```


图12：使用**getpeerinfo**获得的第一个案例摘要示例

这种结构组织方式适用于每个处于活动连接状态的对等节点，共计十个；同时也使我们能够全面了解该节点所连接的网络，并有助于发现异常或瓶颈。

因此，通过 getpeerinfo 进行这项评估后，可以确认该节点已良好接入比特币 P2P 网络，与使用较新协议版本且已启用转发功能的对等方保持连接。此命令还可用于检测连接问题、高延迟或过时的对等节点。因此，它是监控和维护比特币节点的关键工具。

5.8. *settings.json* 文件分析

settings.json 文件充当动态配置文件，会在运行过程中不断更新。但与静态配置文件（例如 bitcoin.conf）不同，此文件用于存储通过 bitcoin-qt 等界面或 RPC 调用所更改的设置，并确保这些设置在不同会话之间持续保留。

$ stat /opt/BLOCKCHAIN/settings.json

再次使用 stat 命令后，可以发现文件大小接近*F190* 字节。自创建以来，该文件从未被修改过，其权限模式*F0600* 仅允许所有者（比特币）访问，从而增强了安全性。

执行以下命令：

$ cat /opt/BLOCKCHAIN/settings.json

[5]远程过程调用：允许从其他程序或脚本控制比特币节点

结果如图*13*所示，其中可以看到该文件仅包含一条警告，并没有可配置的持久化设置。不过，其存在本身表明，系统未来可用于动态存储参数，无需手动编辑默认配置文件。

```
{
"warning": "This file is automatically generated and updated by Bitcoin Core.
     Please do not edit this file while the node is running, as any changes
    might be ignored or overwritten."
}
```

图13：数据目录中的 settings.json 内容

6. 配置调整的影响

为了通过实证检验某些配置参数对比特币节点行为的影响，我们决定开展一系列受控且循序渐进的测试，后文将逐一介绍。这些测试均在同一个运行中的节点上进行，每次测试依次修改配置，并在每次修改前后立即进行测量。

由于某些参数之间存在技术上的不兼容（例如，启用 txindex=1 时不能使用 prune¿0），因此我们选择按一定顺序进行测试，以便在一致的环境下评估每个参数，而无需重新安装节点或对其进行完整重建索引。因此，首先记录的更改是停用交易索引（txindex=0），同时仍将区块历史保留在磁盘上。此后，我们将在前一状态的基础上启用修剪模式，然后再启用其他参数。

为了验证所考虑的配置所产生的具体影响，尤其是对访问旧交易的影响，我们通过 blockstream.info 手动选取了一个旧交易的交易 ID，所选交易如下：

8c14f0db3df150123e6f3dbbf30f8b955a8249b62ac1d1ff16284aefa3d06d87

对应区块 #100000：

000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506

同时还选取了一笔当前交易：

0000000000000000000002e856a0e776fa10c2c2dd061947d72d50b1b30d833f3

对应区块 #898576：

0000000000000000000002e856a0e776fa10c2c2dd061947d72d50b1b30d833f3

由于这些交易确实存有用，其中一笔在区块链中距离当前足够远，可以将其从修剪数据中删除；另一笔则足够近，不会受到影响，从而可以比较不同配置下的信息可用性。

6.1. 禁用交易索引： `txindex=0`

进行的第一个测试是禁用交易索引：修改节点的配置文件，将 txindex=1 改为 txindex=0，因为这样一来，节点将不再维护区块链中所有交易的全局索引，从而影响通过交易 ID 查询交易信息的能力。

修改 bitcoin.conf 文件并重启节点后，通过执行以下命令确认索引已被禁用：

$ bitcoin-cli getindexinfo

得到的结果是一个空对象，确认没有启用任何索引。随后，尝试查询一笔属于区块 #100000 的旧交易：

8c14f0db3df150123e6f3dbbf30f8b955a8249b62ac1d1ff16284aefa3d06d87

以及一笔近期交易，属于区块 #898576：

cf0d196a663995487447e67cc1794314c3fba433ba751cc0fee5dcc6c96a0910

使用命令：

$ bitcoin-cli getrawtransaction <交易 ID> 1

在两种情况下均出现以下错误消息：

错误代码： −5错误消息：没有这样的内存池交易。请使用 −txindex 或提供区块哈希

.....

这一行为确认了节点无法仅凭交易 ′ 的 ′txid 找到该交易。但在重复查询并为每笔交易添加相应的区块哈希 ˜ 后，节点已正确返回结果：

bitcoin −cli getrawtransaction 8c14f0.....1" 000000000003 ba27aa200b1c....." bitcoin −cli getrawtransaction cf0d19.....1" 00000000000000000002 e8....."

由此可以得出，尽管全局索引 ′处于停用状态，仍然可以访问保存在磁盘上的区块交易（前提是知道特定区块的标识符 ′）。因此，从这个角度来看，停用交易索引 txindex ′可以减少磁盘占用，并使节点 ′初始化速度更快，不过显然也限制了更轻松地浏览区块链 ′上交易的可能性。

6.2. 启用修剪模式 ′： prune=550

修剪方法（pruning）[6] 可以减少磁盘空间占用。为此，修改了配置文件 bitcoin.conf ′，并添加以下 ˜ 行：

修剪=550

此参数 ′指示节点仅保留最新的区块，并逐步删除较早的区块，同时将区块文件的存储上限设为 550 MiB。链状态的其余部分（无论是区块头、chainstate 还是未花费交易输出（UTXO））都会保持 ′完整，因此无需在磁盘上保存完整历史记录，也能验证新交易。

保存更改后，停止节点并重新启动节点；随后立即（节点重启后）运行 getblockchaininfo 命令，返回了一个错误：

错 误 代 码： −28错 误 消
息：正在修剪区 块 存 储.....

这意味着初始修剪已经开始。此操作耗时 ′几分钟，在此期间节点无法执行其他操作。完成后，使用以下命令核对结果：

$ bitcoin-cli getblockchaininfo

[6]修剪会从磁盘中删除旧区块以节省空间，但不会影响节点继续保持有效。

它返回的结果如图 *14*所示，这证实该节点确实处于修剪状态，因为它已删除修剪高度所指高度之前的区块。


```
{
  "chain": "main",
  "blocks": 898625,
  "headers": 898625,
  "bestblockhash": "0000000000000000001
      ff3e2ddb6e47b00d062e166351f3940992b829ec9881",
  "bits": "17025049",
  "target":
      "0000000000000000000250490000000000000000000000000000000000000000",
  "difficulty": 121658450774825,
  "time": 1748362081,
  "mediantime": 1748359457,
  "verificationprogress": 0.9999985627622666,
  "initialblockdownload": false,
  "chainwork": "0000000000000000000000000000000000000000000
      c6593f90ef747549b89aaa16",
  "size_on_disk": 553096761,
  "pruned": true,
  "pruneheight": 898283,
  "automatic_pruning": true,
  "prune_target_size": 576716800,
  "warnings": [
  ]
}
```


图 14：修剪模式下 getblockchaininfo 的结果

同样，还观察到磁盘空间占用明显减少：

$ du -sh/opt/BLOCKCHAIN/blocks # 结果：
796M

chainstate/ 目录也占用了约 4.2 GB，这对于一个没有历史记录但状态正确的已同步节点来说，是预期之中的。

为了查看修剪的实际效果，我们尝试通过标识符访问区块 #100000 及其 Coinbase 交易，但这两个命令都返回了以下错误消息：

错误：区块不可用（已修剪数据）

相反，当在最新区块及其关联交易中进行相同测试时，它们仍然可以访问，因为它们位于未修剪范围内。这证实节点会保留区块残留数据，以便在仅记录模式的全节点模式下用于当前验证，但不会保留历史部分。

6.3. 数据库缓存大小： dbcache

dbcache[7]，参数允许设置为执行读取和写入 LevelDB 所需的内部操作而预留的随机存取存储器（RAM）容量；该参数会影响节点启动时的效率、通过区块验证来验证奖励的效率，以及对 UTXO 集的查询效率。

为了评估其影响，我们进行了两次测试：一次使用默认值（约 450–500 MiB），另一次将其设为 64 MiB。为此，我们编辑了 bitcoin.conf 文件，添加了这个新选项：

[7]控制数量　　节点使用的内存容量。该值越大，在硬件性能强劲时，性能就越好

dbcache=64

初始的 bitcoind 进程使用了约 7.6 GiB 的常驻内存（RES），如下方 top 命令的输出所示：

常驻内存：7,6g 内
存百分比：48,9

此外，还执´ 行了 time 命令 bitcoin-cli gettxoutsetinfo（查询并计算当前 l UTXO 集的状态） ，耗时 1 分 58 秒：

实际用时 1 分 58.665 秒

在 bitcoin.conf 中修改数值并重启节点后，内存使用量发生了显著变化：bitcoind 进程的常驻内存增至 4.4 GiB：

常驻内存：4.4g 内
存百分比：28.4

当然，重启服务后显示的消息是：

错误代码：-28 错误消
息：正在验证区块目录...

这是预料之中的，因为正在进行相应的更改，以及前几章中提到的测试。尤其是考虑到现在缓存更小了（这会迫使节点在验证区块时更频繁地从磁盘读取数据），不过进程完成后，就可以进行相应的比较了。

在这种情况下，gettxoutsetinfo 的差异很小，总耗时为 2 分钟 1 秒：

实际耗时 2分1.423秒

尽管如此，内存消耗的减少确实很显著，这一点可以从表1中看出。当在资源受限的环境中运行，或内存受到其他也需要在同一台机器上运行的进程限制时，这显然十分重要。即便如此，对于内存较大的节点，还是建议使用较高的值，以优化短期响应时间。

参数´	**之前** （dbcache 默认值）	**之后** (dbcache=64)
已用内存（常驻内存）	7.6 GiB	4.4 GiB
gettxoutsetinfo 耗时	1分58.665秒	2分1.423秒

表 1：不同 dbcache 值下节点行为的比较

6.4. 内存池最大容量：maxmempool

maxmempool 参数用于定义分配给存储未确认交易（内存池）的随机存取存储器（RAM）容量（以 MiB 为单位）；也就是说，这些交易尚未进入区块，正在等待矿工处理。为评估该参数的影响，首先观察了节点在 maxmempool 默认值下的行为。

为此，通过执行以下命令，获取了内存池的相关数值，例如已存储的交易数量、当前内存使用量以及允许的最大上限：

$ bitcoin-cli getmempoolinfo

此外，还想知道内存池中的活跃交易数量，因此使用了以下命令：

```
$ bitcoin-cli getrawmempool | wc -l
```

随后，使用以下命令分析系统的内存分布，包括已用内存、空闲内存和缓存内存：

```
$ free -h:
```

最后，实时监控 bitcoind 进程的内存使用情况：

```
$ top -p \$(pidof bitcoind)
```

分析时获得的一些重要数值包括：

```
    "usage": 48377424, "maxmempool": 300000000 内
存池中的交易：15.890 bitcoind：常驻内存 11,5G，内存
百分比 73,9
```

随后更新了节点的配置文件，将参数 maxmempool 的值修改为 50 MiB，旨在检验这一限制对待处理交易管理的影响：

```
maxmempool=50
```

保存更改后，使用以下命令重启服务：

```
$ sudo systemctl restart bitcoind
```

接下来，重复之前的步骤以获取更改后的数据。结果显示，交易数量被限制为 10.252，因为节点只保留手续费最高的交易。另一方面，bitcoind 进程自身的内存使用量从 11.5 GiB 降至 1.4 GiB，显示出 maxmempool 减小所产生的影响，如下列相关数值所示：

```
    "usage": 44393264, "maxmempool": 50000000 内
存池中的交易：10.252 bitcoind：常驻内存 1,4G，\%
MEM 内存百分比 9,3
```

这些结果表明，maxmempool 参数值会显著影响节点的行为：限制该参数值一方面会降低节点积累交易的能力，另一方面会减少内存的使用。这对于在资源有限的硬件上运行节点，或出于被动观察的目的运行节点，可能会有所帮助。

6.5. 最大连接数： `maxconnections`

为研究节点在配置 maxconnections 变量时的行为（该变量用于设置节点接受的同时网络连接的最大数量），进行了一些测试。首先，执行以下命令，以查看当时有多少活动连接：

```
$ bitcoin-cli getnetworkinfo | grep 连接
```

此外，获取更详细的网络状态行也很重要，其中包括连接总数（连接）、入站连接（入站连接）和出站连接（出站连接），为此执行了以下命令：

```
$ bitcoin-cli getpeerinfo | jq '. | length'
```

得到以下结果：

```
"连接": 10, "连接_in":
0, "连接_out": 10
```

它们确认 总共有 10 个活动出站连接，但由于重要的是能够比较 CPU 和内存的使用情况 运行了这条命令：

```
$ top -p $(pidof bitcoind)
```

获得以下数值：

```
PID：403251 %CPU：1,0 内存百分比：
7,5 常驻内存：1,2 g（常驻内存）
```

随后尝试´ 将连接数强制设为 20，因此添加了此参数（到 bitcoin.conf 文件中）：

```
maxconnections=20
```

接下来重启服务：

```
$ sudo systemctl restart bitcoind
```

然而，节点仍然保持着 10 个连接，因此可以清楚地看出，当前连接数并未受 maxconnections 的值限制。但由于尝试确认是否有任何影响，便明确减少了连接数，将 maxconnections 从 20 改为 5，然后重启，并检查得到的数值是否发生变化：

```
"连接": 5, "连接_in":
0, "连接_out": 5
```

再次运行 top，以检查对资源使用情况的影响，得到以下数值：

```
进程标识符：403367
CPU：38,7% 内存百分
比：28,9% 常驻内存：
4,5g
```

尽管人们可能会认为，减少连接会降低资源使用量，但在这个具体案例中，情况恰恰相反：将 maxconnections 从 10 降至 5 后，bitcoind 进程开始占用大量 CPU 和内存。这可能是因为它进入了一个非常活跃的阶段，试图维持节点运行，或者寻找其他活跃的对等节点，也可能是在重启后同步某些内部数据结构。

7. 结论

已成功在 Linux 环境中建立并运行比特币全节点，涵盖从源代码编译´到节点与网络同步的全过程。文中详细记录了节点的安装、配置和使用步骤，并分析了 Bitcoin Core 生成的文件，特别关注其结构、用途和运行意义。

此外，本文还通过实证评估了不同参数（交易索引、修剪、区块过滤器索引、dbcache、maxmempool、maxconnections）的配置，并分析了它们对节点行为、资源消耗及可用功能的影响。结果表明，即使在较为普通的家庭环境中，也可以维持节点运行；同时也清楚地证明，只要根据用户目标适当调整参数，节点就能够持续运行。

本文可为未来与区块链研究以及开发区块链数据查询和分析工具相关的工作奠定重要基础。作为未来工作，建议继续推进节点的安装和配置流程，并根据不同用户角色（验证者、分析师、开发者、闪电网络运营者、教育工作者）实现自动化，从而提供满足其需求的解决方案；此外，还可进一步细致考察各种配置对节点性能的直接影响。

参考文献

[1] Andreas M. Antonopoulos. *Mastering Bitcoin*：开放区块链编程 ´开放区块链编程。第 1ª版。独立出版，2019 年。 ISBN：978-1097424108。[2] Carlos Callejo 和 V´ıctor Ronco。《傻瓜式加密货币》。Grupo Planeta，2020 年。 ISBN：978-84-329-0592-6。[3] The Bitcoin Collective。比特币小 ˜书。加利福尼亚州雷德伍德城：21 Million Books，2019 年。 ISBN：978-1700678874。