فصل

۳

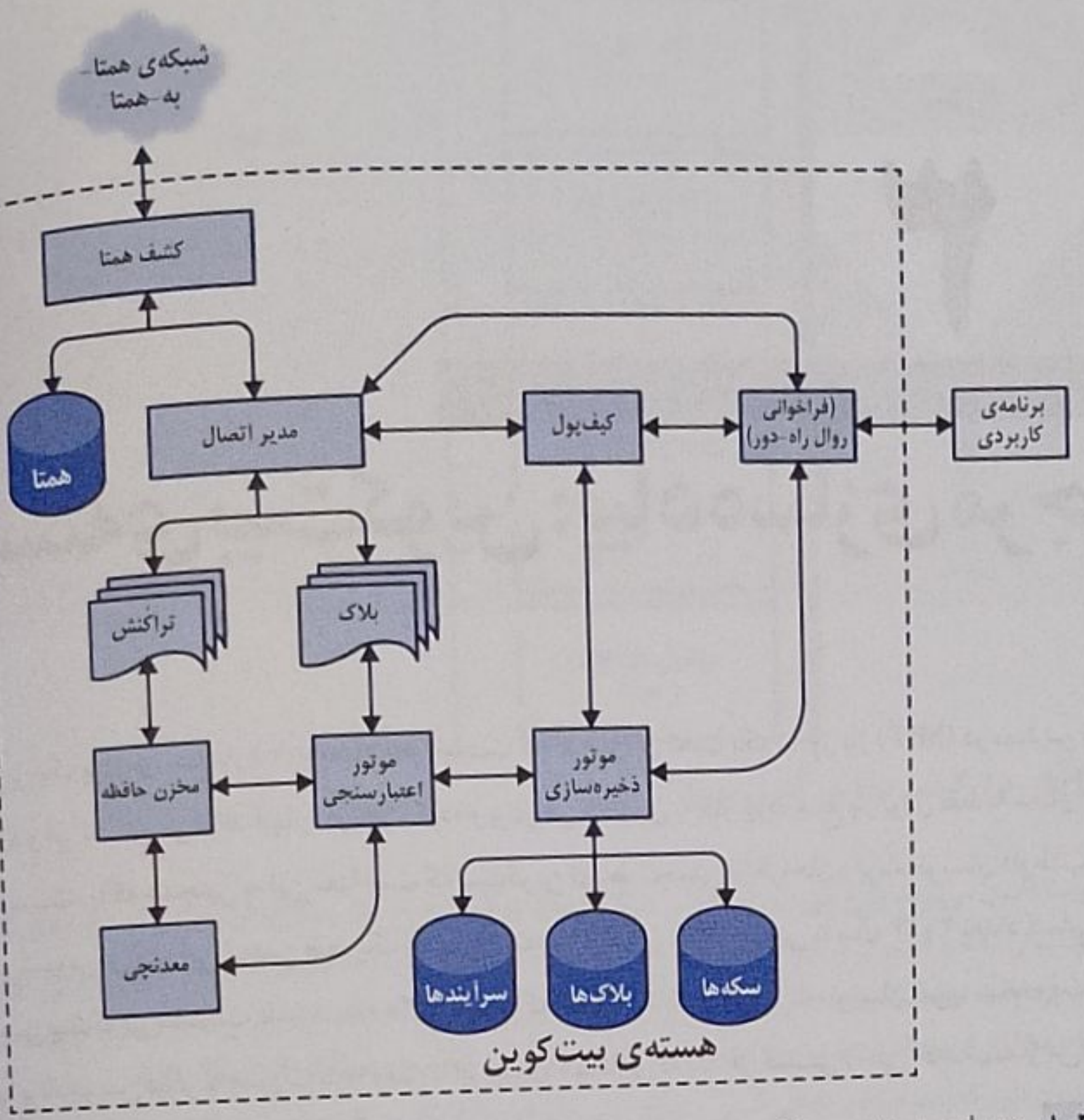
هسته‌ی بیت‌کوین: پیاده‌سازی مرجع

بیت‌کوین یک پروژه‌ی منبع باز (open source) اســت که کُدهای آن تحت یک مجوز باز (MIT) در دسترس همگان قرار دارد و هر کســی می‌تواند آنها را دریافت کرده و برای هر منظوری به کار ببرد. منبع باز بودن فقط به معنای مجانی بودن نیســت، بلکه همچنین به این معنا است که بیت‌کوین توسط انجمنی از طراحان و برنامه‌نویسان داوطلب توسعه داده می‌شــود. در ابتدا این انجمن فقط یک عضو داشت: ساتوشــی ناکاموتو. ولی تا سال ۲۰۱۶ تعداد کسانی که در کُدنویسی بیت‌کوین مشارکت داشتند به ۴۰۰ نفر رسید که حدود ۱۲ نفر از آنها برنامه‌نویسان تقریباً تمام-وقت هستند و دهها برنامه‌نویس دیگر به صورت پاره-وقت با این پروژه همکاری دارند. هر کســی (حتی خود شــما) می‌تواند در کُدنویسی بیت‌کوین مشارکت داشته باشد.

وقتی ساتوشــی ناکاموتو مقاله‌ی معروف اعلام موجودیت بیت‌کوین را منتشر کرد، در واقع نرم‌افزار بیت‌کوین را نوشته بود. ناکاموتو می‌خواست قبل از اعلام عمومی بیت‌کوین از عملی بودن آن مطمئن شود. آن پیاده‌سازی اولیه، که در ابتدا به ســادگی «بیت‌کوین» یا «مشتری ساتوشی» خوانده می‌شد، تا به امروز تغییرات و اصلاحات زیادی را پشت ســر گذاشته و امروزه به چیزی تبدیل شده است که آن را با نام هســته‌ی بیت‌کوین (Bitcoin Core) می‌شناسیم، نامی که برای متمایز کردن این پیاده‌سازی از سایر پیاده‌سازی‌های مشابه به آن داده شده است. هسته‌ی بیت‌کوین *پیاده‌سازی مرجع* سیســتم بیت‌کوین است، یعنی مرجعی رســمی است که چگونگی پیاده‌ســازی تک تک اجزای این فناوری را مشــخص می‌کند. هســته‌ی بیت‌کوین تمامی جنبه‌های بیت‌کوین را پیاده‌ســازی می‌کند، از جملــه کیف‌پول، موتور اعتبارســنجی تراکُنش و بلاک، و یک گره-کامل در شبکه‌ی همتا-به-همتای بیت‌کوین. معماری هسته‌ی بیت‌کوین را در شکل ۳-۱ مشاهده می‌کنید.

هشدار

هر چند در هســته‌ی بیت‌کوین یک پیاده‌سازی مرجع برای کیف‌پول هم وجود دارد، ولی این پیاده‌سازی برای اســتفاده‌ی واقعی کاربران یا برنامه‌های کاربردی نیست. به برنامه‌نویسانی که می‌خواهند برنامه‌ی کیف‌پول بنویسند، توصیه می‌شــود استانداردهای جدیدتر مثل BIP-39 یا BIP-32 را پیاده‌سازی کنند. «بیپ» مخفف پیشنهاد بهسازی بیت‌کوین (Bitcoin Improvement Proposal) است.



شکل ۳-۱ معماری هسته‌ی بیت‌کوین.

محیط برنامه‌نویسی بیت‌کوین

اگر برنامه‌نویس هستید، شاید بخواهید برای نوشتن برنامه‌های کاربردی بیت‌کوین برای خود یک محیط برنامه‌نویسی با تمام ابزارها، کتابخانه‌ها و نرم‌افزارهای پشتیبانی داشته باشید. در این فصل (که برای افراد فنی در نظر گرفته شده) مراحل این کار را قدم به قدم توضیح خواهیم داد. اگر [حداقل فعلاً] علاقه‌ی چندانی به مسائل فنی ندارید، می‌توانید از این فصل بگذرید و به فصل بعد بروید.

کامپایل کردن هسته‌ی بیت‌کوین از کُد منبع

کُد منبع بیت‌کوین را می‌توانید از محل‌های مختلفی، از جمله سایت گیت‌هاب، دریافت کنید. برای این کار، به صفحه‌ی بیت‌کوین در سایت گیت‌هاب (https://github.com/bitcoin/bitcoin) رفته و از منوی clone or download (سمت راست صفحه) گزینه‌ی Download ZIP را انتخاب کنید تا فایل آرشیو *bitcoin-master.zip* در سیستم شما بارگذاری شود (این فایل را در دیسک پیوست کتاب هم می‌توانید بیابید). [با گزینه‌ی Open in Desktop هم می‌توانید یک کپی از این کُد روی میز کار سیستم خود ایجاد کنید.]

توجه

در بسیاری از مثال‌های این فصل از رابط کاربری خط-فرمان (که به آن «پوسته» هم گفته می‌شود) استفاده خواهیم کرد. در این پوسته یک «اعلان خط-فرمان» می‌بینید که یک خط کوچک در مقابل آن چشمک می‌زند؛ وقتی فرمانی را وارد کرده و کلید Enter را می‌زنید، پوسته آن فرمان را اجرا کرده و بعد از نمایش خروجی فرمان (البته در صورتی که خروجی داشته باشد)، به اعلان خط-فرمان برمی‌گردد. شکل اعلان خط-فرمان در سیستم‌های مختلف متفاوت است، ولی در مثال‌های این فصل آن را با کاراکتر $ نشان داده‌ایم. به بیان دیگر، وقتی در مثال‌های این فصل پیشوند $ را در ابتدای یک خط می‌بینید، نباید این کاراکتر را وارد کنید: فقط فرمان بعد از آن را تایپ کرده و کلید Enter را بزنید. در این مثال‌ها، خطوطی که زیر یک فرمان می‌بینید، پاسخ سیستم‌عامل به آن فرمان هستند.

روش دیگر بارگیری فایل کُد منبع بیت‌کوین استفاده از فرمان `git clone` در پنجره‌ی خط-فرمان است، که آن را در مثال زیر می‌بینید:

```
$ git clone https://github.com/bitcoin/bitcoin.git
Cloning into 'bitcoin'...
remote: Counting objects: 66193, done.
remote: Total 66193 (delta 0), reused 0 (delta 0), pack-reused 66193
Receiving objects: 100% (66193/66193), 63.39 MiB | 574.00 KiB/s, done.
Resolving deltas: 100% (48395/48395), done.
Checking connectivity... done.
$
```

توجه

`git` یک «سیستم کنترل ویرایش» توزیع‌شده است که هر برنامه‌نویس حرفه‌ای باید آن را داشته باشد. این فرمان در دو نوع خط-فرمان و گرافیکی عرضه می‌شود که می‌توانید بسته به نیاز و سلیقه‌ی خود از آنها استفاده کنید. (ویرایش‌های مختلف فایل نصبی `git` برای ویندوز را می‌توانید در دیسک پیوست کتاب بیابید.)

وقتی بارگیری فایل‌های بیت‌کوین و ایجاد پوشه‌ی نصب آن (به نام *bitcoin*) به اتمام رسید، با تایپ کردن فرمان زیر در اعلان خط-فرمان سیستم خود و زدن Enter وارد این دایرکتوری شوید:

```
$ cd bitcoin
```

انتخاب ویرایش مناسب هسته‌ی بیت‌کوین

وقتی نرم‌افزار بیت‌کوین (یا هر نرم‌افزار دیگر) را با فرمان `git clone` بارگیری می‌کنید، آخرین ویرایش موجود آن را در اختیار شما قرار می‌دهد، اما ممکن است این یک ویرایش در دست توسعه و در نتیجه ناپایدار باشد، پس قبل از کامپایل کردن آن، یک ویرایش پایدار را با استفاده از برچسب *نرم‌افزار* (که چیزی نیست جز شماره‌ی ویرایش‌های مختلف یک نرم‌افزار) انتخاب کنید. این کار باعث می‌شود تا نرم‌افزار بارگیری‌شده با ویرایشی که شما انتخاب می‌کنید، هماهنگ و سازگار شود. برای اطلاع از تمامی ویرایش‌های موجود نرم‌افزار بیت‌کوین می‌توانید فرمان `git tag` را به صورت زیر به کار ببرید:

```
$ git tag
v0.1.5
```

```
v0.1.6test1
v0.10.0
...
v0.11.2
v0.11.2rc1
v0.12.0rc1
v0.12.0rc2
...
```

خروجی فرمان `git tag` فهرست تمام ویرایش‌های منتشرشده‌ی بیت‌کوین از آغاز تا امروز را نشان می‌دهد. بنا بر قرارداد، ویرایش‌های آزمایشی با پسوند `rc` مشخص می‌شوند، و ویرایش‌های پایدار که می‌توان از آنها روی سیستم‌های تولیدی استفاده کرد، این پسوند را ندارند. از فهرست بالا می‌توانید ببینید که آخرین ویرایش پایدار بیت‌کوین در زمان اجرای این فرمان `v0.11.2` بوده است (شاید وقتی شما از این فرمان استفاده می‌کنید، ویرایش جدیدی منتشر شده باشد که می‌توانید آن را در سیستم خود نصب کنید). برای هماهنگ‌سازی نرم‌افزار بارگیری‌شده با فرمان `git clone` با این ویرایش پایدار، فرمان `git checkout` را به صورت زیر به کار ببرید:

```
$ git checkout v0.11.2
HEAD is now at 7e27892... Merge pull request #6975
```

برای اطمینان از موفقیت نصب ویرایش مورد نظر، از فرمان `git status` به صورت زیر استفاده کنید:

```
$ git status
HEAD detached at v0.11.2
nothing to commit, working directory clean
```

تولید و پیکربندی اسکریپت‌های نصب هسته‌ی بیت‌کوین

در کُد منبع بیت‌کوین تعداد زیادی فایل مستندات آن نیز گنجانده شده است. فایل راهنمای اصلی بیت‌کوین که نام آن *README.md* است، در پوشه‌ی *bitcoin* قرار دارد که می‌توانید آنها را با فرمان `more README.md` (و زدن کلید space برای رفتن به صفحه‌ی بعد) مطالعه کنید [یک نسخه از این فایل در پوشه‌ی *bitcoin/doc* نیز قرار داده شده است]. در این فصل مشتری خط-فرمان بیت‌کوین، که در لینوکس به آن `bitcoind` گفته می‌شود، را ساخته و اجرا می‌کنیم. برای دیدن چگونگی کامپایل و ساخت مشتری خط-فرمان `bitcoind` در سیستم‌عامل خود می‌توانید فرمان `more doc/build-unix.md` را در اعلان خط-فرمان اجرا کنید. دستورات مشابه برای کامپایل و ساخت مشتری بیت‌کوین در سیستم‌عامل‌های ویندوز و Mac OS X را نیز می‌توانید در فایل‌های *build-windows.md* و *build-osx.md* در همان دایرکتوری *bitcoin/doc* بیابید.

پیش‌نیازهای کامپایل و ساخت بیت‌کوین (که در بخش اول این مستندات ذکر شده‌اند) را به دقت فراهم و اجرا کنید. قبل از آن که بتوانید کامپایل کردن بیت‌کوین را شروع کنید، باید تعدادی کتابخانه را در سیستم خود نصب کرده باشید. اگر هر یک از این پیش‌نیازها وجود نداشته باشد، فرآیند کامپایل و ساخت هسته‌ی بیت‌کوین با خطا متوقف خواهد شد. اگر این اتفاق افتاد، می‌توانید قطعه‌ی گمشده را نصب کرده و فرآیند کامپایل را از جایی که قطع شده، ادامه دهید. با فرض آن که تمامی پیش‌نیازهای کامپایل و ساخت بیت‌کوین از قبل نصب و آماده شده‌اند، فرآیند نصب بیت‌کوین با تولید چند *اسکریپت ساخت* با استفاده از اسکریپت `autogen.sh` شروع می‌شود:

```
$ ./autogen.sh
...
glibtoolize: copying file 'build-aux/m4/libtool.m4'
glibtoolize: copying file 'build-aux/m4/ltoptions.m4'
glibtoolize: copying file 'build-aux/m4/ltsugar.m4'
glibtoolize: copying file 'build-aux/m4/ltversion.m4'
...
configure.ac:10: installing 'build-aux/compile'
configure.ac:5: installing 'build-aux/config.guess'
configure.ac:5: installing 'build-aux/config.sub'
configure.ac:9: installing 'build-aux/install-sh'
configure.ac:9: installing 'build-aux/missing'
Makefile.am: installing 'build-aux/depcomp'
...
```

نکته

فرآیند کامپایل/ساخت هسته‌ی بیت‌کوین از ویرایش 0.9 به بعد به گونه‌ای تغییر یافت که از سیستم autogen/configure/make استفاده کند (و به احتمال زیاد استفاده از آن در آینده نیز ادامه خواهد یافت). روش کار در ویرایش‌های قدیمی‌تر هسته‌ی بیت‌کوین قدری متفاوت است و از یک Makefile ساده استفاده می‌کند. فرآیند کامپایل/ساخت هسته‌ی بیت‌کوین را بر اساس ویرایشی که برای نصب انتخاب کرده‌اید، دنبال کنید.

اسکریپت autogen.sh یک مجموعه اسکریپت پیکربندی خودکار ایجاد می‌کند که سیستم شما را پویش کرده و بهترین پیکربندی برای آن، و اطمینان از این که تمامی کتابخانه‌های لازم برای کامپایل شدن کُد هسته را در سیستم خود دارید، انتخاب خواهد کرد. مهمترین اسکریپت این مجموعه اسکریپتی به نام configure است که گزینه‌های مختلف برای سفارشی‌سازی فرآیند کامپایل/ساخت را ارائه می‌کند. برای دیدن این گزینه‌ها می‌توانید فرمان ./configure –help. را اجرا کنید:

```
$ ./configure --help
'configure' configures Bitcoin Core 0.11.2 to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...


...
Optional Features:
--disable-option-checking ignore unrecognized --enable/--with options
--disable-FEATURE do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG] include FEATURE [ARG=yes]

--enable-wallet enable wallet (default is yes)

--with-gui[=no|qt4|qt5|auto]
...
```

اسکریپت configure به شما اجازه می‌دهد از طریق پرچم‌های --enable-FEATURE و --disable-FEATURE، که در آنها FEATURE نام گزینه‌ی دلخواه است (خروجی مثال بالا را ببینید)، ویژگی‌های مختلف bitcoind را فعال یا غیرفعال کنید. در این فصل ما bitcoind را با ویژگی‌های پیش‌فرض کامپایل و نصب می‌کنیم و از پرچم‌های اسکریپت configure استفاده نخواهیم کرد (ولی برای آشنایی با این گزینه‌ها خوب است نگاهی به مستندات آن بیندازید). اگر کامپیوتری که می‌خواهید bitcoind را نصب کنید، به طور انحصاری متعلق به شما نیست یا چند نفر دیگر در آن شریک هستید (مثل محیط‌های شرکتی یا دانشگاهی)، به احتمال زیاد یک *دایرکتوری خانه* به شما اختصاص داده شده که فقط می‌توانید برنامه‌های خود را در آن (یا زیردایرکتوری‌های آن) نصب کنید. در این حالت باید دایرکتوری نصب bitcoind را با استفاده از پرچم --perfix=$HOME به دایرکتوری خانه‌ی خود تغییر دهید.

در اینجا چند گزینه‌ی مفید اسکریپت configure را که می‌توانید از آنها برای نصب bitcoind در حالت غیرپیش‌فرض استفاده کنید، توضیح می‌دهیم:

--perfix=$HOME

با این گزینه می‌توان مکان پیش‌فرض نصب فایل‌های اجرایی هسته‌ی بیت‌کوین (دایرکتوری */usr/local/*) را تغییر داد.

--disable-wallet

از این گزینه می‌توان برای غیرفعال کردن پیاده‌سازی مرجع کیف‌پول استفاده کرد.

--with-incompatible-bdb

این گزینه اجازه می‌دهد تا از یک ویرایش ناسازگار «کتابخانه‌ی پایگاه داده‌ی برکلی» برای ساخت کیف‌پول استفاده کنید.

--with-gui=no

این گزینه کامپایل/ساخت/نصب «رابط کاربر گرافیکی» (GUI) را که به کتابخانه‌ی Qt (کوئیک‌تایم) نیاز دارد، غیرفعال می‌کند. این گزینه فقط رابط سرویس‌دهنده‌ی بیت‌کوین را نصب خواهد کرد.

بعد از آشنا شدن با گزینه‌های اسکریپت configure، اکنون می‌توانید آن را اجرا کنید تا تمامی کتابخانه‌های مورد نیاز را شناسایی کرده و یک اسکریپت کامپایل/ساخت مناسب برای سیستم شما تولید کند:

```
$ ./configure
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
...
[many pages of configuration tests follow]
...
$
```

اگر همه چیز به خوبی پیش برود، فرمان configure با تولید اسکریپت‌های سفارشی‌شده‌ی کامپایل/ساخت bitcoind برای سیستم شما به پایان خواهد رسید. اگر این فرمان به هر اِشکال غیرمترقبه‌ای برخورد کند یا کتابخانه‌های

مورد نیاز را در سیستم شما نیابد، با خطا متوقف خواهد شد. محتمل‌ترین منشأ بروز خطا در اجرای فرمان configure عدم وجود یک کتابخانه یا یک کتابخانه‌ی ناسازگار است. برای رفع این اشکالات و آماده‌سازی همه‌ی پیش‌نیازهای لازم، مستندات کامپایل/ساخت بیت‌کوین را به دقت مطالعه کرده، و سپس یک بار دیگر اسکریپت configure را اجرا کنید.

کامپایل /ساخت فایل‌های اجرایی هسته‌ی بیت‌کوین

اکنون آماده‌اید تا کُد منبع هسته‌ی بیت‌کوین را کامپایل کنید، فرآیندی که (بسته به سرعت CPU و مقدار حافظه‌ی آزاد کامپیوتر) می‌تواند تا یک ساعت طول بکشد. در حین کامپایل، کامپیوتر را به حال خود رها نکنید: هر چند ثانیه یا چند دقیقه یک بار آن را چک کرده و پیغام‌های آن را بررسی کنید تا خطایی رُخ نداده باشد. اگر خطایی رُخ داده، یا فرآیند کامپایل قطع شده است، می‌توانید با وارد کردن فرمان make آن را از سر بگیرید:

```
$ make
Making all in src
  CXX      crypto/libbitcoinconsensus_la-hmac_sha512.lo
  CXX      crypto/libbitcoinconsensus_la-ripemd160.lo
  CXX      crypto/libbitcoinconsensus_la-sha1.lo
  CXX      crypto/libbitcoinconsensus_la-sha256.lo
  CXX      crypto/libbitcoinconsensus_la-sha512.lo
  CXX      libbitcoinconsensus_la-hash.lo
  CXX      primitives/libbitcoinconsensus_la-transaction.lo
  CXX      libbitcoinconsensus_la-pubkey.lo
  CXX      script/libbitcoinconsensus_la-bitcoinconsensus.lo
  CXX      script/libbitcoinconsensus_la-interpreter.lo

[... many more compilation messages follow ...]

$
```

اگر همه چیز خوب و بدون خطا پیش رفته باشد، هسته‌ی بیت‌کوین کامپایل شده است، و آخرین گام نصب فایل‌های اجرایی با فرمان sudo make install خواهد بود. از آنجا که برای این مرحله به دسترسی سرپرست سیستم نیاز دارید، احتمالاً برنامه‌ی نصب گذرواژه‌ی کاربر super (که در ویندوز به آن administrator گفته می‌شود) را از شما طلب خواهد کرد:

```
$ sudo make install
Password:
Making install in src
  ../build-aux/install-sh -c -d '/usr/local/lib'
libtool: install: /usr/bin/install -c bitcoind /usr/local/bin/bitcoind
libtool: install: /usr/bin/install -c bitcoin-cli /usr/local/bin/bitcoin-cli
libtool: install: /usr/bin/install -c bitcoin-tx /usr/local/bin/bitcoin-tx
...
$
```

برنامه‌ی نصب فایل‌های اجرایی bitcoind را به طور پیش‌فرض در دایرکتوری */usr/local/bin/* قرار می‌دهد. برای اطمینان از این که آیا واقعاً این فایل‌ها در محل درست نصب شده‌اند، می‌توانید از فرمان which به صورت زیر استفاده کنید:

```
$ which bitcoind
/usr/local/bin/bitcoind

$ which bitcoind-cli
/usr/local/bin/bitcoind-cli
```

اجرای یک گره هسته‌ی بیت‌کوین

شبکه‌ی همتا-به-همتای بیت‌کوین از هزاران گره (node) تشکیل شده است، که اکثر آنها توسط داوطلبان و شرکت‌های تجاری که برنامه‌های کاربردی بیت‌کوین تولید می‌کنند، اداره می‌شوند. این گره‌های فعال دارای اشراف کامل و رسمی بر بلاک‌چین بیت‌کوین هستند، و یک نسخه از تمامی تراکُنش‌ها (که به طور مستقل توسط سیستم خود آنها اعتبارسنجی شده‌اند) نزد خود نگه می‌دارند. اگر یک گره [کامل] بیت‌کوین داشته باشید، برای اعتبارسنجی تراکُنش‌ها به هیچ کس متکی نخواهید بود؛ علاوه بر آن، گره شما در شبکه‌ی بیت‌کوین مشارکت داشته و به مستحکم‌تر شدن آن کمک می‌کند.

با این حال، داشتن یک گره بیت‌کوین مستلزم داشتن سیستمی است که همیشه به شبکه متصل بوده و دارای منابع سخت‌افزاری کافی برای پردازش تمامی تراکُنش‌های بیت‌کوین باشد. اگر بخواهید یک نسخه‌ی کامل از بلاک‌چین نگه دارید و این نسخه همواره اندیس‌گذاری‌شده (مرتب و به سرعت قابل جستجو) باشد، به مقداری زیاد فضای دیسک و حافظه‌ی RAM نیاز خواهید داشت. امروزه برای داشتن یک گره اندیس-کامل (full-index node) به بیش از ۲۵۰ GB فضای دیسک و ۲ GB حافظه‌ی RAM، و همچنین اینترنت پُرسرعت با سقف ارسال/دریافت تقریباً نامحدود نیاز است. اگر پهنای باند اینترنت شما محدود است یا هزینه‌ی آن بر حسب مقدار داده‌ی مبادله‌شده محاسبه می‌شود، یا باید قید داشتن یک گره کامل بیت‌کوین را بزنید، یا مقدار مصرف آن را به کمک نرم‌افزارهای مناسب محدود کنید (مثال ۳-۲ را ببینید).

توجه

هسته‌ی بیت‌کوین به طور پیش‌فرض یک نسخه‌ی کامل از بلاک‌چین، شامل تمامی تراکُنش‌هایی که از زمان تولد بیت‌کوین در سال ۲۰۰۹ در این شبکه اتفاق افتاده‌اند، نگه می‌دارد. تخمین زده می‌شود که در ابتدای سال ۲۰۱۸ حجم بلاک‌چین بیت‌کوین به بیش از ۲۰۰ GB رسیده باشد، و حجم آن هر روز و هر ساعت بیشتر هم می‌شود. هسته‌ی بیت‌کوین تا بارگیری کامل مجموعه داده‌های بلاک‌چین (موسوم به *دیتاست*) قادر به پردازش تراکُنش‌ها یا به‌روزرسانی تراز حساب‌ها نخواهد بود. وقتی یک گره کامل برای اولین بار شروع به کار می‌کند، برای همگام‌سازی آن به فضای دیسک، پهنای باند و زمان کافی نیاز است. برای کاهش حجم بلاک‌چین می‌توان هسته‌ی بیت‌کوین را طوری پیکربندی کرد که بلاک‌های قدیمی را دور بیندازد (مثال ۳-۲ را ببینید)، ولی قبل از آن که بتوان بلاک‌های قدیمی را پاکسازی کرد، همچنان کل دیتاست بلاک‌چین باید بارگیری شود.

با وجود این حجم از منابع مورد نیاز، هزاران داوطلب در سرتاسر جهان کامپیوترهای خود را به گره کامل بیت‌کوین تبدیل کرده‌اند. برخی از این گره‌ها روی سیستم‌های کوچک و ساده‌ای مثل رَزبری پای (کامپیوترهایی به اندازه‌ی یک بسته سیگار با قیمتی زیر ۶۰۰ هزار تومان) اجرا می‌شوند. اجرای گره‌های بیت‌کوین روی سرویس‌دهنده‌های اجاره‌ای (که معمولاً سیستم‌عامل لینوکس دارند) نیز رایج است؛ یک *سرویس‌دهنده‌ی خصوصی مجازی* (VPS) یا *سرویس‌دهنده‌ی پردازش ابری* (CCS) [با اجاره‌ی کمتر از ۱۰ دلار در ماه] را به سادگی می‌توان به یک گره کامل بیت‌کوین تبدیل کرد. اما اصلاً چرا یک نفر باید بخواهد یک گره کامل بیت‌کوین راه بیندازد؟ از مهمترین دلایل آن می‌توان به موارد زیر اشاره کرد:

- برای نوشتن نرم‌افزار برای بیت‌کوین به دسترسی API به شبکه‌ی بیت‌کوین و بلاک‌چین نیاز دارید که فقط از طریق یک گره کامل امکان‌پذیر است.
- تولید برنامه‌های کاربردی که تراکُنش‌ها را بر اساس قواعد اجماع بیت‌کوین اعتبارسنجی کنند. شرکت‌هایی که برنامه‌های کاربردی بیت‌کوین تولید می‌کنند، معمولاً تعداد زیادی گره کامل دارند.
- پشتیبانی از بیت‌کوین به عنوان یک پول جهانی. هر چه تعداد گره‌های کامل بیت‌کوین بیشتر باشند، شبکه‌ی بیت‌کوین قویتر شده و بهتر می‌تواند به کاربران سرویس بدهد.
- عدم اعتماد به برنامه‌های ثالث برای اعتبارسنجی تراکُنش‌ها.

اگر بعد از خواندن این کتاب علاقمند شدید خودتان برای بیت‌کوین برنامه بنویسید، باید یک گره کامل برای خود دست و پا کنید.

اولین اجرای هسته‌ی بیت‌کوین

وقتی برای اولین بار bitcoind را اجرا می‌کنید، به شما یادآوری می‌کند که باید برای رابط برنامه‌نویسی JSON-RPC یک *فایل پیکربندی* با گذرواژه‌ی قوی ایجاد کنید. این گذرواژه دسترسی به *رابط برنامه‌نویسی* (API) هسته‌ی بیت‌کوین را کنترل می‌کند. برای اجرای bitcoind فقط کافی است نام این برنامه را پای اعلان خط-فرمان بنویسید و Enter را بزنید:

```
$ bitcoind
Error: To use the "-server" option, you must set a rpcpassword in the
configuration file:
/home/ubuntu/.bitcoin/bitcoin.conf
It is recommended you use the following random password:
rpcuser=bitcoinrpc
rpcpassword=2XA4DuKNCbtZXsBQRRNDEwEY2nM6M4H9Tx5dFjoAVVbK
(you do not need to remember this password)
The username and password MUST NOT be the same.
If the file does not exist, create it with owner-readable-only file permissions.
It is also recommended to set alertnotify so you are notified of problems;
for example: alertnotify=echo %s | mail -s "Bitcoin Alert" admin@foo.com
```

همان طور که می‌بینید، وقتی bitcoind را برای اولین بار اجرا می‌کنید، به شما هشدار می‌دهد که به یک فایل پیکربندی با حداقل دو مدخل rpcuser و rpcpassword نیاز دارید. علاوه بر آن توصیه می‌کند که ساز و کار اخطاردهی را نیز فعال کنید. در قسمت بعد گزینه‌های مختلف پیکربندی bitcoind و چگونگی ایجاد یک فایل پیکربندی را بررسی خواهیم کرد.

پیکربندی گره هسته‌ی بیت‌کوین

فایل پیکربندی bitcoind را که نام آن باید *bitcoin.conf* باشد، در برنامه‌ی ویرایشگر متنی دلخواه خود (ایجاد و) باز کرده و مدخل‌های نام‌کاربری (rpcuser) و گذرواژه (rpcpassword) را عوض کنید. *از گذرواژه‌ای که در این کتاب می‌بینید، استفاده نکنید.* این فایل باید در دایرکتوری *bitcoin*. (که در داخل دایرکتوری خانه‌ی کاربری که با آن وارد سیستم می‌شوید، قرار دارد) ذخیره شود، بنابراین نام کامل آن *bitcoin/bitcoin.conf*. است:

```
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

علاوه بر گزینه‌های rpcuser و rpcpassword، هسته‌ی بیت‌کوین بیش از ۱۰۰ گزینه‌ی پیکربندی یا پارامتر خط-فرمان دیگر نیز دارد که با آنها می‌توانید رفتار گره شبکه، چگونگی ذخیره‌سازی بلاک‌چین، و بسیاری دیگر از جنبه‌های عملیاتی آن را کنترل کنید. برای دیدن این گزینه‌ها فقط کافی است فرمان bitcoind -help را اجرا کنید:

```
$ bitcoind --help
Bitcoin Core Daemon version v0.11.2

Usage:
  bitcoind [options]                     Start Bitcoin Core Daemon

Options:
  -?
       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)
...
[many more options]
...

  -rpcsslciphers=<ciphers>
       Acceptable ciphers (default:
       TLSv1.2+HIGH:TLSv1+HIGH:!SSLv2:!aNULL:!eNULL:!3DES:@STRENGTH)
```

اجازه دهید تعدادی از مهمترین گزینه‌های پیکربندی یا پارامترهای خط-فرمان bitcoind را کمی بیشتر توضیح دهیم:

alertnotify

با اجرای فرمان یا اسکریپت مشخص‌شده، (معمولاً از طریق ایمیل) یک پیغام اضطراری به صاحب گره ارسال می‌کند.

conf

ارجاع به یک مکان ثانویه برای فایل پیکربندی. این یکی از پارامترهای خط-فرمان bitcoind است، و نمی‌تواند در داخل فایل پیکربندی باشد.

datadir

با این گزینه می‌توان سیستم فایل و دایرکتوری محل ذخیره‌سازی داده‌های بلاک‌چین را انتخاب کرد. مکان پیش‌فرض همان دایرکتوری *bitcoin*. (در دایرکتوری *HOME$* کاربر) است. توجه داشته باشید که، قبل از شروع به کار bitcoind، این سیستم فایل بایستی صدها گیگابایت ظرفیت خالی داشته باشد.

`prune`

این گزینه می‌تواند، با حذف بلاک‌های قدیمی، فضای دیسک مورد نیاز را به چند ده مگابایت کاهش دهد. اگر فضای کافی برای ذخیره‌سازی کامل بلاک‌چین ندارید، این گزینه را فعال کنید.

`txindex`

اندیس تمام تراکُنش‌ها را نگه می‌دارد. فعال کردن این گزینه به معنای ذخیره کردن یک نسخه‌ی کامل از بلاک‌چین است که به شما اجازه می‌دهد فقط با داشتن ID یک تراکُنش آن را از طریق برنامه بازیابی کنید.

`maxconnections`

حداکثر تعداد گره‌هایی که گره شما اتصال از آنها را می‌پذیرد. اگر این گزینه را از مقدار پیش‌فرض پایین‌تر بیاورید، مصرف پهنای باند کاهش خواهد یافت. اگر پهنای باند اینترنت شما محدودیت سرعت یا سقف حجمی دارد، از این گزینه استفاده کنید.

`maxmempool`

حداکثر مقدار مخزن حافظه‌ی تراکُنش را (بر حسب مگابایت) مشخص می‌کند. این گزینه اجازه می‌دهد مصرف حافظه‌ی گره خود را کاهش دهید.

`maxreceivebuffer/maxsendbuffer`

مقدار بافر ارسال/دریافت هر اتصال را (بر حسب مضارب ۱۰۰۰ بایت) مشخص می‌کند. این گزینه اجازه می‌دهد مصرف حافظه را در گره خود کاهش دهید.

`minrelaytxfee`

حداقل مبلغ تراکُنش برای دریافت کارمزد. اگر مبلغ یک تراکُنش کمتر از این مقدار باشد، کارمزدی بابت آن دریافت نمی‌شود. اگر با کمبود حافظه روبرو هستید، به کمک این گزینه می‌توانید اندازه‌ی مخزن تراکُنش درون-حافظه را کاهش دهید.

اندیس پایگاه داده‌ی تراکُنش و گزینه‌ی txindex

هسته‌ی بیت‌کوین، به طور پیش‌فرض، پایگاه داده‌ای ایجاد می‌کند که فقط حاوی تراکُنش‌های مرتبط با کیف‌پول کاربر است. اگر می‌خواهید با فرمان‌هایی مانند `getrawtransaction` (که در ادامه توضیح خواهیم داد) به هر تراکُنشی دسترسی داشته باشید، باید هسته‌ی بیت‌کوین گره خود را طوری پیکربندی کنید که یک اندیس تراکُنش کامل بسازد؛ و این کار را می‌توانید با استفاده از گزینه‌ی `txindex=1` در فایل پیکربندی هسته‌ی بیت‌کوین انجام دهید. اگر مدتی بعد از اجرای `bitcoind` تصمیم به ایجاد یک اندیس کامل گرفته و این گزینه را فعال کرده‌اید، باید اجرای `bitcoind` را قطع کرده و یک بار دیگر آن را با پارامتر خط-فرمان `reindex`- اجرا کنید. فرآیند ایجاد اندیس کامل وقت‌گیر است، پس کمی صبور باشید.

در مثال ۳-۱ یک فایل پیکربندی می‌بینید که نشان می‌دهد چگونه می‌توان با استفاده از گزینه‌های بالا یک گره بیت‌کوین با اندیس کامل ایجاد کرد، گرهی که امکان دسترسی به API بیت‌کوین را (که برای نوشتن نرم‌افزارهای بیت‌کوین لازم است) نیز داشته باشد.

مثال ۳-۱ فایل پیکربندی برای یک گره کامل

```
alertnotify=myemailscript.sh "Alert: %s"
datadir=/lotsofspace/bitcoin
txindex=1
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

مثال ۳- ۲ فایل پیکربندی مورد نیاز برای اجرای هسته‌ی بیت‌کوین روی سرویس‌دهنده‌ی کوچک و با منابع محدود را نشان می‌دهد.

مثال ۳-۲ فایل پیکربندی برای سیستمی با منابع محدود

```
alertnotify=myemailscript.sh "Alert: %s"
maxconnections=15
prune=5000
minrelaytxfee=0.0001
maxmempool=200
maxreceivebuffer=2500
maxsendbuffer=500
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

همین که فایل پیکربندی را با گزینه‌های مناسب برای سیستم خود [ایجاد و] ویرایش کردید، می‌توانید bitcoind را با این پیکربندی آزمایش کنید. در زیر یک نمونه از اجرای هسته‌ی بیت‌کوین با پارامتر خط-فرمان printtoconsole (که باعث می‌شود خروجی‌های برنامه روی صفحه‌ی نمایشگر نمایش داده شوند) را مشاهده می‌کنید:

```
$ bitcoind -printtoconsole
Bitcoin version v0.11.20.0
Using OpenSSL version OpenSSL 1.0.2e 3 Dec 2015
Startup time: 2015-01-02 19:56:17
Using data directory /tmp/bitcoin
Using config file /tmp/bitcoin/bitcoin.conf
Using at most 125 connections (275 file descriptors available)
Using 2 threads for script verification
scheduler thread start
HTTP: creating work queue of depth 16
No rpcpassword set - using random cookie authentication
Generated RPC authentication cookie /tmp/bitcoin/.cookie
HTTP: starting 4 worker threads
Bound to [::]:8333
Bound to 0.0.0.0:8333
Cache configuration:
* Using 2.0MiB for block index database
* Using 32.5MiB for chain state database
* Using 65.5MiB for in-memory UTXO set
init message: Loading block index...
```

```
Opening LevelDB in /tmp/bitcoin/blocks/index
Opened LevelDB successfully

[... more startup messages ...]
```

همین که به اندازه‌ی کافی اطلاعات کسب کردید و از پیکربندی و اجرای صحیح bitcoind مطمئن شدید، می‌توانید با زدن Ctrl+C ادامه آن را قطع کنید. اگر می‌خواهید هسته‌ی بیت‌کوین به صورت یک فرآیند در پس‌زمینه‌ی سیستم اجرا شود، آن را پارامتر خط-فرمان daemon به صورت bitcoind -daemon اجرا کنید.

برای نظارت بر پیشـــرفت و وضعیت لحظه‌ای گره بیت‌کوین، از فرمان bitcoin-cli getinfo به صورت زیر استفاده کنید:

```
$ bitcoin-cli getinfo

{
    "version" : 110200,
    "protocolversion" : 70002,
    "blocks" : 396328,
    "timeoffset" : 0,
    "connections" : 15,
    "proxy" : "",
    "difficulty" : 120033340651.23696899,
    "testnet" : false,
    "relayfee" : 0.00010000,
    "errors" : ""
}
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

خروجی فرمان بالا نشـــان می‌دهد این گره در حال اجرای ویرایش 0.11.2 هسته‌ی بیت‌کوین، با بلاک‌چینی به ارتفاع ۳۹۶۳۲۸ بلاک، و ۱۵ اتصال فعال شبکه است.

بعد از رسیدن به بهترین پیکربندی، بایستی (با توجه به سیستم‌عامل خود) کاری کنید که هر بار کامپیوتر شروع به کار می‌کند، هســـته‌ی بیت‌کوین نیز اجرا شـــود تا بتواند همواره در حال فعالیت باشـــد. در دایرکتوری *contrib/init* تعدادی اسکریپت برای این منظور عرضه شده است و فایل *README.md* (در همین دایرکتوری) هم نشان می‌دهد که هر اسکریپت برای کدام سیستم‌عامل مناسب است.

رابط برنامه‌نویسی (API) هسته‌ی بیت‌کوین

مشتری هسته‌ی بیت‌کوین یک رابط JSON-RPC پیاده‌سازی می‌کند که می‌توان از طریق برنامه‌ی کمکی bitcoin-cli به آن دسترسی داشت. این برنامه‌ی خط-فرمان قابلیت‌هایی را به صورت تعاملی در اختیار شما قرار می‌دهد که می‌توانید در برنامه‌های خود از طریق API از آنها استفاده کنید. برای شروع، فرمان help فهرستی از *فراخوانی‌های راه دور* (RPC) موجود در این برنامه را به نمایش می‌گذارد:

```
$ bitcoin-cli help
addmultisigaddress nrequired ["key",...] ( "account" )
```

```
addnode "node" "add|remove|onetry"
backupwallet "destination"
createmultisig nrequired ["key",...]
createrawtransaction [{"txid":"id","vout":n},...] {"address":amount,...}
decoderawtransaction "hexstring"

...
...
verifymessage "bitcoinaddress" "signature" "message"
walletlock
walletpassphrase "passphrase" timeout
walletpassphrasechange "oldpassphrase" "newpassphrase"
```

هر یک از این فرمان‌ها می‌توانند تعدادی پارامتر (آرگومان) بگیرند که از طریق همین فرمان help می‌توانید با ساختار این دستورات و نقش پارامترهای آنها آشنا شوید. برای مثال، فرض کنید می‌خواهید فراخوانی راه دور getblockhash را بیشتر بشناسید؛ با اجرای فرمان زیر اطلاعات کاملی از این فرمان به دست خواهید آورد:

```
$ bitcoin-cli help getblockhash
getblockhash index

Returns hash of block in best-block-chain at index provided.

Arguments:
1. index         (numeric, required) The block index

Result:
"hash"         (string) The block hash

Examples:
> bitcoin-cli getblockhash 1000
> curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest",
"method": "getblockhash", "params": [1000] }' -H 'content-type: text/plain;'
http://127.0.0.1:8332/
```

در انتهای این توضیحات دو مثال از چگونگی فراخوانی این فرمان RPC، با استفاده از برنامه‌ی کمکی bitcoin-cli یا یک مشتری HTTP به نام curl، داده شده است. مثال اول را اجرا کنید و نتیجه‌ی آن را ببینید:

```
$ bitcoin-cli getblockhash 1000
00000000c937983704a73af28acdec37b049d214adbda81d7e2a3dd146f6ed09
```

خروجی این فراخوانی RPC یک دَرهم بلاک است (که در فصل‌های آینده بیشتر درباره‌ی آن توضیح خواهیم داد). در این فرمان از هسته‌ی بیت‌کوین خواسته شده تا دَرهم بلاک ۱۰۰۰ را برگرداند. آنچه در حال حاضر باید بدانید این است که این فراخوانی را روی هر گرهی اجرا کنید، نتیجه‌ی باید همانی باشد که در مثال بالا می‌بینید. در قسمت‌های بعد چند فرمان RPC بسیار سودمند و خروجی موردانتظار آنها را نشان خواهیم داد.

گرفتن اطلاعات درباره‌ی وضعیت مشتری هسته‌ی بیت‌کوین

فرمان: `getinfo`

فراخوانی راه دور `getinfo` اطلاعات پایه درباره‌ی وضعیت گره شبکه‌ی بیت‌کوین، کیف‌پول، و پایگاه داده‌ی بلاک‌چین نمایش می‌دهد. چگونگی اجرای این فرمان و یک نمونه از خروجی آن را در زیر می‌بینید:

```
$ bitcoin-cli getinfo

{
    "version" : 110200,
    "protocolversion" : 70002,
    "blocks" : 396367,
    "timeoffset" : 0,
    "connections" : 15,
    "proxy" : "",
    "difficulty" : 120033340651.23696899,
    "testnet" : false,
    "relayfee" : 0.00010000,
    "errors" : ""
}
rpcuser=bitcoinrpc
rpcpassword=CHANGE_THIS
```

فرمان‌های RPC هسته‌ی بیت‌کوین داده‌ها را با فرمت نمادگذاری شیء جاواسکریپت (JavaScript Object Notation: JSON) برمی‌گردانند، فرمتی که علاوه بر قابل‌استفاده بودن در برنامه‌های کامپیوتری برای ما نیز قابل‌درک است. در این خروجی می‌توان ویرایش نرم‌افزار مشتری بیت‌کوین (110200) و ویرایش پروتکل بیت‌کوین (70002) را تشخیص داد. همچنین می‌توان دید که این مشتری چه تعداد بلاک را می‌شناسد، یا به عبارت دیگر ارتفاع بلاک بلاک‌چین آن چقدر است (۳۹۶۳۶۷). در این خروجی آمار و اطلاعات متنوعی از شبکه‌ی بیت‌کوین و آن دسته از تنظیمات آن که به این مشتری مربوط است، دیده می‌شود.

توجه

بارگیری کل بلاک‌ها از گره‌های دیگر و «همگام شدن» یک مشتری `bitcoind` با ارتفاع فعلی بلاک‌چین مدت زمان زیادی (شاید بیش از یک روز) طول خواهد کشید. برای دیدن پیشرفت کار و اطلاعات از تعداد بلاک‌های شناخته‌شده می‌توانید از فرمان `getinfo` استفاده کنید.

کاوش و رمزگشایی تراکُنش‌ها

فرمان: `getrawtransaction` و `decoderawtransaction`

در فصل قبل دیدیم که آلیس یک فنجان قهوه از قهوه‌خانه‌ی باب خرید. تراکُنش آلیس در این بلاک‌چین با شناسه‌ی تراکُنش (`txid`) `0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2` ثبت شد. اجازه دهید این شناسه‌ی تراکُنش را (به عنوان آرگومان) به API هسته‌ی بیت‌کوین بدهیم و ببینیم چه اطلاعاتی به ما برمی‌گرداند:

```
$ bitcoin-cli getrawtransaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a↵
5fbd8a57286c345c2f2

{
0100000001186f9f998a5aa6f048e51dd8419a14d8a0f1a8a2836dd734d2804fe65fa35779000↵
000008b4830450221008 84d142d86652a3f47ba4746ec719bbfbd040a570b1deccbb6498c75c4↵
ae24cb02204b9f039ff08df09cbe9f6addac960298cad530a863ea8f53982c09db8f6e3813014↵
10484ecc0d46f1918b30928fa0e4ed99f16a0fb4fde0735e7ade8416ab9fe423cc54123363767↵
89d172787ec3457eee41c04f4938de5cc17b4a10fa336a8d752adfffffffff0260e3160000000↵
0001976a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788acd0ef8000000000001976a9↵
147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac00000000
```

توجه

تا زمانی که یک تراکُنش تأیید نشود، شناسه‌ی تراکُنش (txid) آن رسمیت نخواهد یافت. فقدان دَرهم یک تراکُنش در بلاک‌چین به معنای آن نیست که این تراکُنش پردازش نشده است. این پدیده به «چکش‌خواری تراکُنش» معروف است، چون دَرهم یک تراکُنش می‌تواند قبل از تأیید شدن در یک بلاک (و اضافه شدن به آن) تغییر کند. بعد از تأیید شدن، این txid رسمیت می‌یابد و غیرقابل‌تغییر می‌شود.

فرمان getrawtransaction یک تراکُنش سریالی‌شـــده در قالب هگزادسیمال برمی‌گرداند. برای رمزگشایی این تراکُنش از فرمان decoderawtransaction استفاده می‌کنیم و خروجی هگزادسیمال فرمان getrawtransaction را به عنوان آرگومان به آن می‌دهیم:

```
$ bitcoin-cli decoderawtransaction 0100000001186f9f998a5aa6f048e51dd8419a14d8↵
a0f1a8a2836dd734d2804fe65fa35779000000008b4830450221008 84d142d86652a3f47ba474↵
6ec719bbfbd040a570b1deccbb6498c75c4ae24cb02204b9f039ff08df09cbe9f6addac960298↵
cad530a863ea8f53982c09db8f6e381301410484ecc0d46f1918b30928fa0e4ed99f16a0fb4fd↵
e0735e7ade8416ab9fe423cc5412336376789d172787ec3457eee41c04f4938de5cc17b4a10fa↵
336a8d752adfffffffff0260e3160000000001976a914ab68025513c3dbd2f7b92a94e0581f5↵
d50f654e788acd0ef8000000000001976a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a8↵
88ac00000000

{
  "txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
  "size": 258,
  "version": 1,
  "locktime": 0,
  "vin": [
    {
      "txid": "7957a35fe64f80d234d76d83a2...8149a41d81de548f0a65a8a999f6f18",
      "vout": 0,
      "scriptSig": {
        "asm":"3045022100884d142d86652a3f47ba4746ec719bbfbd040a570b1decc...",
        "hex":"4830450221008 84d142d86652a3f47ba4746ec719bbfbd040a570b1de..."
```

```
      },
      "sequence": 4294967295
    }
  ],
  "vout": [
    {
      "value": 0.01500000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 ab68...f654e7 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a914ab68025513c3dbd2f7b92a94e0581f5d50f654e788ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA"
        ]
      }
    },
    {
      "value": 0.08450000,
      "n": 1,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 7f9b1a...025a8 OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a9147f9b1a7fb68d60c536c2fd8aeaa53a8f3cc025a888ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK"
        ]
      }
    }
  ]
}
```


رمزگشایی یک تراکُنش تمامی اجزای تشکیل‌دهنده‌ی آن، از جمله ورودی‌ها و خروجی‌های تراکُنش، را نمایش می‌دهد. در این مثال می‌توان دید تراکُنشی که ۱۵ میلی‌بیت‌کوین به آدرس جدید ما (که در اینجا آدرس بیت‌کوین باب است) واریز کرده، خود یک ورودی داشته و دو خروجی تولید کرده است. ورودی این تراکُنش خروجی یک تراکُنش تأییدشده بوده است: تراکُنشی که `vin txid` (شناسه‌ی ورودی) آن با `7957a35fe` شروع می‌شود. دو خروجی این تراکُنش نیز پرداخت ۱۵ میلی‌بیت‌کوین به ما، و برگرداندن تتمه (۸۴٫۵ میلی‌بیت‌کوین) به صاحب حساب بوده‌اند.

از آنجا که اکنون `txid` تراکُنش‌های قبلی را در اختیار دارید، می‌توانید با همان فرمان‌های `getrawtransaction` و `decoderawtransaction` آنها را بیشتر بررسی کنید و تا جایی که علاقه (یا حوصله) دارید در بلاک‌چین به عقب (پایین) برگردید، چون زنجیره‌ی تراکُنش‌ها (بلاک‌چین) در واقع چیزی نیست جز بیت‌کوین‌هایی که از یک آدرس به آدرس(های) دیگر منتقل شده‌اند.

کاوش بلاک‌ها

فرمان: getblock و getblockhash

کند و کاو در یک بلاک بسیار شبیه کاوش تراکُنش‌ها است، با این تفاوت که برای ارجاع به بلاک‌ها از دو وسیله می‌توان استفاده کرد: *ارتفاع* یک بلاک یا دَرهم آن بلاک. اجازه دهید ابتدا کاوش بلاک را با استفاده از ارتفاع آن شروع کنیم. در فصل قبل دیدیم که تراکُنش آلیس (خرید یک فنجان قهوه از مغازه‌ی باب) در بلاک ۲۷۷۳۱۶ قرار گرفت. اگر این ارتفاع بلاک را به عنوان پارامتر به فرمان getblockhash بدهیم، دَرهم آن بلاک را برمی‌گرداند:

```
$ bitcoin-cli getblockhash 277316
000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4
```

اکنون که فهمیدیم تراکُنش آلیس در کدام بلاک قرار گرفته، می‌توانیم آن بلاک را کاوش کنیم. برای این کار از فرمان getblock کمک می‌گیریم و دَرهم بلاک مورد نظر را به عنوان آرگومان به آن می‌دهیم:

```
$ bitcoin-cli getblock 000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b3↵
1b2cc7bdc4

{
  "hash": "000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4",
  "confirmations": 37371,
  "size": 218629,
  "height": 277316,
  "version": 2,
  "merkleroot":
"c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e",
  "tx": [
    "d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f",
    "b268b45c59b39d7596147577l8b9918caf0ba9d97c56f3b91956ff877c503fbe",
    "04905ff987ddd4cfe603b03cfb7ca50ee81d89d1f8f5f265c38f763eea4a21fd",
    "32467aab5d04f51940075055c2f20bbd1195727c961431bf0aff8443f9710f81",
    "561c5216944e21fa29dd12aaa1a45e3397f9c0d888359cb05e1f79fe73da37bd",
[... hundreds of transactions ...]
    "78b300b2a1d2d9449b58db7bc71c3884d6e0579617e0da4991b9734cef7ab23a",
    "6c87130ec283ab4c2c493b190c20de4b28ff3caf72d16ffa1ce3e96f2069aca9",
    "6f423dbc3636ef193fd8898dfdf7621dcade1bbe509e963ffbff91f696d81a62",
    "802ba8b2adabc5796a9471f25b02ae6aeee2439c679a5c33c4bbcee97e081196",
    "eaaf6a048588d9ad4d1c092539bd571dd8af30635c152a3b0e8b611e67d1a1af",
    "e67abc6bd5e2cac169821afc51b207127f42b92a841e976f9b752157879ba8bd",
    "d38985a6a1bfd35037cb7776b2dc86797abbb7a06630f5d03df2785d50d5a2ac",
    "45ea0a3f6016d2bb90ab92c34a7aac9767671a8a84b9bcce6c019e60197c134b",
    "c098445d748ced5f178ef2ff96f2758cbec9eb32cb0fc65db313bcac1d3bc98f"
  ],
  "time": 1388185914,
  "mediantime": 1388183675,
  "nonce": 924591752,
  "bits": "1903a30c",
```

```
  "difficulty": 1180923195.258026,
  "chainwork":
"0000000000000000000000000000000000000000000934695e92aaf53afa1a",
  "previousblockhash":
"000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569",
  "nextblockhash":
"000000000000000010236c269dd6ed714dd5db39d36b33959079d78dfd431ba7"
}
```

این بلاک حاوی ۴۱۹ تراکُنش است و تراکُنش آلیس (0627052b...) به عنوان شصت و چهارمین تراکُنش آن ثبت شده است. درایه‌ی height به ما می‌گوید که این بلاک ۲۷۷۳۱۶ اُم در بلاک‌چین است.

چگونگی استفاده از رابط برنامه‌نویسی هسته‌ی بیت‌کوین

برنامه‌ی کمکی bitcoin-cli ابزاری بسیار مفید برای کند و کاو در API (رابط برنامه‌نویسی) هسته‌ی بیت‌کوین و آزمایش توابع آن است. ولی API برای آن نیست که با آن بازی و آزمایش کنیم، بلکه هدف اصلی آن فراهم آوردن دسترسی به این توابع از درون برنامه‌های کاربردی است. در این قسمت نشان خواهیم داد که چگونه می‌توانید در یک برنامه‌ی کاربردی توابع API هسته‌ی بیت‌کوین را فراخوانی کنید.

همان طور که قبلاً گفتیم، API هسته‌ی بیت‌کوین یک رابط از نوع JSON-RPC است. رابط برنامه‌نویسی JSON (*نمادگذاری شیء جاواسکریپت*) داده‌ها را با فرمتی برمی‌گرداند که علاوه بر قابل‌استفاده بودن در برنامه‌های کامپیوتری برای ما انسان‌ها نیز قابل‌درک هستند. **RPC** (فراخوانی روال راه دور: Remote Procedure Call) یعنی می‌توان روال‌ها (توابع) را از راه دور (در گره هسته‌ی بیت‌کوین) و از طریق پروتکل‌های شبکه فراخوانی کرده و نتایج آن را دریافت کرد. گره‌های بیت‌کوین از پروتکل‌های HTTP یا HTTPS (برای اتصال اَمن) استفاده می‌کنند.

وقتی از فرمان bitcoin-cli برای گرفتن اطلاعات درباره‌ی فرمان‌های bitcoind استفاده می‌کنید، دو مثال عملی از چگونگی کاربرد آن فراخوانی را نمایش می‌دهد؛ در یکی از این مثال‌ها چگونگی فراخوانی JSON-RPC تابع مورد نظر با استفاده از برنامه‌ی curl، که یک مشتری خط-فرمان HTTP است، نشان داده می‌شود:

```
$ curl --user myusername --data-binary '{"jsonrpc": "1.0", "id":"curltest",
"method": "getinfo", "params": [] }' -H 'content-type: text/plain;' http://
127.0.0.1:8332/
```

این مثال نشان می‌دهد که برنامه‌ی curl یک درخواست HTTP به میزبان محلی (127.0.0.1) ارسال می‌کند، به پورت پیش‌فرض بیت‌کوین (8332) متصل می‌شود، و یک درخواست jsonrpc برای متد getinfo با استفاده از کُدگذاری text/plain به آن می‌فرستد. اگر رابط برنامه‌نویسی JSON-RPC را در کامپیوتر خود نصب و پیاده‌سازی کرده باشید، به جای برنامه‌ی curl از یک کتابخانه‌ی معمولی HTTP هم می‌توانید برای فراخوانی توابع هسته‌ی بیت‌کوین استفاده کنید.

با این حال، امروزه تقریباً تمامی زبان‌های برنامه‌نویسی معروف و رایج شامل کتابخانه‌هایی هستند که API هسته‌ی بیت‌کوین را در خود جای داده‌اند. در این کتاب از کتابخانه‌ی pythonbitcoinlib برای دسترسی ساده‌تر به API هسته‌ی بیت‌کوین استفاده خواهیم کرد. [به یاد داشته باشید که برای اجرای مثال‌های این قسمت باید هسته‌ی بیت‌کوین (که مسئول پاسخگویی به فراخوانی‌های JSON-RPC است) در حال اجرا روی کامپیوتر شما باشد.]

در مثال ۳-۳ یک اسکریپت پایتون به نام rpc_example.py می‌بینید که یک فراخوانی getinfo ساده انجام می‌دهد و پارامترهای بلاک را از داده‌های برگشتی هسته‌ی بیت‌کوین استخراج و چاپ می‌کند.

مثال ۳-۳ اجرای فرمان getinfo از طریق فراخوانی JSON-RPC هسته‌ی بیت‌کوین

```
from bitcoin.rpc import RawProxy

# Create a connection to local Bitcoin Core node
p = RawProxy()

# Run the getinfo command, store the resulting data in info
info = p.getinfo()

# Retrieve the 'blocks' element from the info
print(info['blocks'])
```

نتیجه‌ی اجرای این اسکریپت را در زیر می‌بینید:

```
$ python rpc_example.py
394075
```

این خروجی می‌گوید گره هسته‌ی بیت‌کوین ما ۳۹۴۰۷۵ بلاک در بلاک‌چین خود دارد. قبول داریم که برنامه‌ی چندان درخشانی به نظر نمی‌رسد، ولی نشان می‌دهد چطور می‌توان خیلی ساده‌تر با API هسته‌ی بیت‌کوین ارتباط برقرار کرد.

در مثال بعدی از فراخوانی‌های فرمان‌های getrawtransaction و decoderawtransaction برای بازیابی اطلاعات تراکُنش آلیس (پرداخت پول یک فنجان قهوه به باب) استفاده می‌کنیم. اسکریپت پایتون مثال ۳-۴ (به نام rpc_transaction.py) بعد از استخراج تراکُنش آلیس، خروجی‌های آن را نمایش می‌دهد (یک بار دیگر یادآوری می‌کنیم که تراکُنش آلیس دو خروجی دارد: پرداخت به باب و برگشت تتمه‌ی ورودی به خود آلیس). در اینجا، برای هر خروجی، فقط آدرس گیرنده و مقدار آن را چاپ کرده‌ایم.

مثال ۳-۴ بازیابی یک تراکُنش و نمایش خروجی‌های آن

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# Alice's transaction ID
txid = "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2"

# First, retrieve the raw transaction in hex
raw_tx = p.getrawtransaction(txid)

# Decode the transaction hex into a JSON object
decoded_tx = p.decoderawtransaction(raw_tx)

# Retrieve each of the outputs from the transaction
for output in decoded_tx['vout']:
    print(output['scriptPubKey']['addresses'], output['value'])
```

اگر این اسکریپت را اجرا کنید، خروجی زیر را خواهید دید:

```
$ python rpc_transaction.py
([u'1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA'], Decimal('0.01500000'))
([u'1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK'], Decimal('0.08450000'))
```

مثال‌های قبلی هر دو بسیار ساده بودند؛ در واقع، برای گرفتن این اطلاعات اصلاً نیازی به برنامه‌نویسی نیست! (فرمان `bitcoin-cli` همین اطلاعات را به سادگی در اختیار شما قرار می‌دهد.) اما، مثال بعدی صدها فراخوانی RPC انجام می‌دهد و قدرت واقعی یک رابط برنامه‌نویسی را به نمایش می‌گذارد.

مثال ۳-۵ (اسکریپت پایتون `rpc_block.py`) ابتدا بلاک ۲۷۷۳۱۶ را بازیابی می‌کند، سپس اطلاعات ۴۱۹ تراکُنش داخل این بلاک را با ارجاع به `txid` آنها استخراج کرده، و خروجی تمامی این تراکُنش‌ها را با یکدیگر جمع می‌زند.

مثال ۳-۵ بازیابی یک بلاک و جمع زدن خروجی تراکُنش‌های آن

```
from bitcoin.rpc import RawProxy

p = RawProxy()

# The block height where Alice's transaction was recorded
blockheight = 277316

# Get the block hash of block with height 277316
blockhash = p.getblockhash(blockheight)

# Retrieve the block by its hash
block = p.getblock(blockhash)

# Element tx contains the list of all transaction IDs in the block
transactions = block['tx']

block_value = 0

# Iterate through each transaction ID in the block
for txid in transactions:
    tx_value = 0
    # Retrieve the raw transaction by ID
    raw_tx = p.getrawtransaction(txid)
    # Decode the transaction
    decoded_tx = p.decoderawtransaction(raw_tx)
    # Iterate through each output in the transaction
    for output in decoded_tx['vout']:
        # Add up the value of each output
        tx_value = tx_value + output['value']

    # Add the value of this transaction to the total
    block_value = block_value + tx_value

print("Total value in block: ", block_value)
```

با اجرای این اسکریپت خروجی زیر را خواهید دید:

```
$ python rpc_block.py
('Total value in block: ', Decimal('10322.07722534'))
```

این مثال نشـــان می‌دهد که مجموع کل تراکُنش‌های این بلاک ۱۰۳۲۲٫۰۷۷۲۲۵۳۴ BTC (شامل ۲۵ BTC جایزه و ۰٫۰۹۰۹ BTC کارمزد) اســـت. این اطلاعات را با اطلاعاتی که ســـایت‌های کاوشگر بلاک درباره‌ی بلاک ۲۷۷۳۱۶ به دســـت می‌دهند، مقایســـه کنید. برخی سایت‌های کاوشـــگر بلاک مبالغ جایزه و کارمزد را از مقدار کل تراکُنش‌ها کسر می‌کنند؛ ببینید آیا می‌توانید این تفاوت را تشخیص دهید.

مشتری‌ها، کتابخانه‌ها، و جعبه‌ابزارهای دیگر

در زیســـت‌بوم بیت‌کوین انواع مختلفی از مشـــتری، کتابخانه، جعبه‌ابزار، و حتی پیاده‌ســـازی گره-کامل وجود دارد. این پیاده‌ســـازی‌ها برای زبان‌های برنامه‌نویسی مختلف ارائه شـــده‌اند، تا هر کسی با هر سلیقه و علاقه‌ای بتواند برای بیت‌کوین برنامه بنویسید. در این قســـمت تعدادی از بهترین کتابخانه‌ها، مشتری‌ها، و جعبه‌ابزارهای بیت‌کوین برای زبان‌های برنامه‌نویسی محبوب و رایج را معرفی می‌کنیم.

C/C++

هسته‌ی بیت‌کوین https://github.com/bitcoin/bitcoin

پیاده‌سازی مرجع بیت‌کوین.

libbitcoin https://github.com/libbitcoin/libbitcoin

گره، کتابخانه‌ی اجماع، و جعبه ابزار برنامه‌نویسی چند-پلتفرمی C++.

کاوشگر بیت‌کوین https://github.com/libbitcoin/libbitcoin-explorer

ابزار خط-فرمان libbitcoin.

picocoin https://github.com/jgarzik/picocoin

کتابخانه‌ی مشتری سبک‌وزن زبان C برای بیت‌کوین، که توسط جف گارزیک توسعه داده شده است.

جاواسکریپت

bcoin https://bcoin.io/

پیاده‌سازی ماژولار و مقیاس‌پذیر گره-کامل به همراه API.

Bitcore https://bitcore.io/

گره کامل، API، و کتابخانه محصول Bitpay.

BitcoinJS https://github.com/bitcoinjs/bitcoinjs-lib

کتابخانه‌ی بیت‌کوین به زبان جاواسکریپت خالص برای node.js و مرورگرهای وب.

جاوا

bitcoinj

https://bitcoinj.github.io/

کتابخانه‌ی مشتری گره-کامل به زبان جاوا.

(Bits of Proof) BOP

https://bitsofproof.com/

پیاده‌سازی بیت‌کوین به صورت یک کلاس جاوا.

پایتون

python-bitcoinlib

https://github.com/petertodd/python-bitcoinlib

گره، کتابخانه‌ی اجماع، و کتابخانه‌ی بیت‌کوین به زبان پایتون، که توسط پیتر تاد توسعه داده شده است.

pycoin

https://github.com/richardkiss/pycoin

کتابخانه‌ی بیت‌کوین به زبان پایتون، که توسط ریچارد کیس توسعه داده شده است.

pybitcointools

https://github.com/vbuterin/pybitcointools

کتابخانه‌ی بیت‌کوین به زبان پایتون، که توسط ویتالیک بوترین توسعه داده شده است.

Ruby

bitcoin-client

https://github.com/sinisterchipmunk/bitcoin-client

کتابخانه‌ی پوشش‌دهنده‌ی Ruby برای رابط برنامه‌نویسی JSON-RPC.

Go

btcd

https://github.com/btcsuite/btcd

مشتری گره-کامل بیت‌کوین به زبان Go.

Rust

rust-bitcoin

https://github.com/apoelstra/rust-bitcoin

کتابخانه‌ی Rust برای سریال‌سازی، خوانش، و فراخوانی‌های API.

C#

NBitcoin

https://github.com/MetacoSA/NBitcoin

کتابخانه‌ی جامع بیت‌کوین برای چارچوب دات‌نت.

Objective-C

CoreBitcoin

https://github.com/oleganza/CoreBitcoin

جعبه‌ابزار بیت‌کوین برای زبان‌های برنامه‌نویسی Objective-C و سوئیفت (مخصوص سیستم‌عامل Mac OS X).

کتابخانه‌های بسیار دیگری نیز برای زبان‌های برنامه‌نویسی دیگر موجود هستند، و هر روز هم بر تعداد آنها افزوده می‌شود.