СОЗДАНИЕ СМАРТ-КОНТРАКТОВ
SOLIDITY ДЛЯ БЛОКЧЕЙНА

ETHEREUM

ПРАКТИЧЕСКОЕ РУКОВОДСТВО

Александр Вячеславович Фролов

Создание смарт-контрактов Solidity для блокчейна Ethereum. Практическое руководство

http://www.litres.ru/pages/biblio_book/?art=45279075
SelfPub; 2019

Аннотация

Эта книга поможет быстро приступить к созданию смарт-контактов Solidity и распределенных приложений DApp для блокчейна Ethereum. Она состоит из 12 уроков с практическими заданиями. Выполнив их, читатель сможет создавать собственные локальные узлы Ethereum, публиковать смарт-контракты и вызывать их методы, обмениваться данными между реальным миром и смарт-контрактами с помощью оракулов, работать с сетью Rinkeby. Книга адресована всем, кто интересуется передовыми технологиями в области блокчейнов и хочет быстро получить знания, позволяющие заниматься интересной и перспективной работой.

Содержание

Введение 6
Обратная связь 10
Урок 1. Кратко о блокчейне и сети Ethereum 11
Что такое блокчейн 13
Распределенная база данных 13
Распределенный реестр данных 14
Транзакции 14
Публичные и приватные блокчейны 15
Практические применения блокчейна 15
Проблемы с блокчейном 17
Как формируется цепочка блокчейна 20
Блокчейн Ethereum 24
Майнинг, или Как создаются блоки 25
Итоги урока 27
Урок 2. Подготовка рабочей среды в ОС Ubuntu и Debian 29
Выбор операционной системы 31
Установка необходимых утилит 34
Установка Geth и Swarm в Ubuntu 36
Установка Geth и Swarm в Debian 39
Предварительная подготовка 39
Загрузка дистрибутива Go 40
Установка переменных окружения 40

Проверка версии Go 41
Установка Geth и Swarm 42
Создаем приватный блокчейн 45
Готовим файл genesis.json 45
Создаем каталог для работы 48
Создаем аккаунт 48
Запускаем инициализацию узла 49
Параметры запуска узла 58
Подключаемся к нашему узлу 60
Управление майнингом и проверка баланса 64
Завершение работы консоли Geth 67
Итоги урока 68
Урок 3. Подготовка рабочей среды на Raspberry Pi 3 69
Подготовка Raspberry Pi 3 к работе 71
Установка Rasberian 71
Установка обновлений 72
Включение доступа SSH 73
Установка статического адреса IP 73
Установка необходимых утилит 75
Установка Go 76
Загрузка дистрибутива Go 76
Установка переменных окружения 77
Проверка версии Go 78
Установка Geth и Swarm 79
Создаем приватный блокчейн 81

Проверка учетной записи и баланса 83
Итоги урока 85
Урок 4. Учетные записи и перевод средств между аккаунтами 86
Просмотр и добавление аккаунтов 88
Просмотр списка аккаунтов 89
Добавление аккаунта 90
Параметры команды geth account 92
Пароли аккаунтов 93
Криптовалюта в Ethereum 95
Перевод средств с одного аккаунта на другой 99
Метод eth.sendTransaction 99
Просмотр состояния транзакции 104
Квитанция транзакции 107
Итоги урока 111
Урок 5. Публикация первого контракта 112
Смарт-контракты в Ethereum 114
Конец ознакомительного фрагмента. 115

Введение

Наша книга предназначена для тех, кто хочет не только понять принципы работы блокчейна Ethereum, но и получить практические навыки в создании распределенных приложений DApp на языке программирования Solidity для этой сети.

Эту книгу лучше не просто читать, а работать с ней, выполняя практические задания, описанные в уроках. Для работы вам потребуются локальный компьютер, виртуальный или облачный сервер с установленной ОС Debian или Ubuntu. Также для выполнения многих заданий можно использовать Raspberry Pi.

На первом уроке мы рассмотрим принципы работы блокчейна Ethereum и основную терминологию, а также расскажем о том, где можно использовать этот блокчейн.

Цель **второго урока** – создать узел приватного блокчейна Ethereum для дальнейшей работы в рамках этого курса на сервере Ubuntu и Debian. Мы рассмотрим особенности установки основных утилит, таких как geth, обеспечивающего работу узла нашего блокчейна, а также демона децентрализованного хранилища данных swarm.

Третий урок научит вас проводить эксперименты с Ethereum на недорогом микрокомпьютере Raspberry Pi. Вы установите операционную систему (ОС) Rasberian на

Raspberry Pi, утилиту Geth, обеспечивающую работу узла блокчейна, а также демон децентрализованного хранилища данных Swarm.

Четвертый урок посвящен аккаунтам и криптовалютным единицам в сети Ethereum, а также способам перевода средств с одного аккаунта на другой из консоли Geth. Вы научитесь создавать аккаунты, инициировать транзакции перевода средств, получать состояние транзакции и ее квитанцию.

На пятом уроке вы познакомитесь со смарт-контрактами в сети Ethereum, узнаете об их выполнении виртуальной машиной Ethereum.

Вы создадите и опубликуете в приватной сети Ethereum свой первый смарт-контракт и научитесь вызывать его функции. Для этого вы будете использовать среду разработки Remix Solidity IDE. Кроме того, вы научитесь устанавливать и использовать пакетный компилятор solc.

Также мы расскажем о так называемом бинарном интерфейсе приложения Application Binary Interface (ABI) и научим его использовать.

Шестой урок посвящен созданию скриптов JavaScript, работающих под управлением Node.js и выполняющих операции со смарт-контрактами Solidity.

Вы установите Node.js в ОС Ubuntu, Debian и Rasberian, напишете скрипты для публикации смарт-контракта в локальной сети Ethereum и вызова его функций.

Кроме того, вы научитесь переводить с помощью скриптов средства между обычными аккаунтами, а также зачислять их на аккаунты смарт-контрактов.

На седьмом уроке вы научитесь устанавливать и использовать популярную среди разработчиков смарт-контрактов Solidity интегрированную среду Truffle. Вы научитесь создавать скрипты JavaScript, вызывающие функции контрактов с помощью модуля truffle-contract, а также выполните тестирование своего смарт-контракта средствами Truffle.

Восьмой урок посвящен типам данных Solidity. Вы напишете смарт-контракты, работающие с такими типами данных, как знаковые и беззнаковые целые числа, числа со знаком, строки, адреса, переменные сложных типов, массивы, перечисления, структуры и словари.

На девятом уроке вы приблизитесь еще на шаг к созданию смарт-контрактов для основной сети Ethereum. Вы научитесь публиковать контракты при помощи Truffle в приватной сети Geth, а также в тестовой сети Rinkeby. Отладка смарт-контракта в сети Rinkeby очень полезна перед его публикацией в основной сети – там практически все по-настоящему, но бесплатно.

В рамках урока вы создадите узел тестовой сети Rinkeby, пополните его средствами и опубликуете смарт-контракт.

Урок 10 посвящен распределенным хранилищам данных Ethereum Swarm. Используя распределенные хранилища, вы экономите на хранении данных большого объема в блокчей-

не Ethereum.

В рамках этого урока вы создадите локальное хранилище Swarm, выполните операции записи и чтения файлов, а также каталогов с файлами. Далее вы научитесь работать с публичным шлюзом Swarm, напишите скрипты для обращения к Swarm из Node.js, а также с помощью модуля Perl Net::Ethereum::Swarm.

Цель урока 11 – освоить работу со смарт-контрактами Solidity с применением популярного языка программирования Python и фреймворка Web3.py. Вы установите этот фреймворк, напишите скрипты для компиляции и публикации смарт-контракта, а также для вызова его функций. При этом Web3.py будет использован как сам по себе, так и совместно с интегрированной средой разработки Truffle.

На 12 уроке вы научитесь передавать данные между смарт-контрактами и реальным миром при помощи оракулов. Это пригодится вам для получения данных с Web-сайтов, устройств интернета вещей IoT, различных приборов и датчиков, и отправки данных из смарт-контрактов на эти устройства. В практической части урока вы создадите оракул и смарт-контракт, получающий актуальный курс обмена USD на рубли с сайта ЦБ РФ.

Обратная связь

Вы можете связаться с автором этой книги по адресам sbook@frolov-lib.ru или https://www.facebook.com/frolov.shop2you.

Пожалуйста, отправляйте ваши замечания и рекомендации, автор постарается их учесть при переиздании книги.

Урок 1. Кратко о блокчейне и сети Ethereum

Цель урока: познакомиться с принципами работы блокчейна Ethereum, областями его применения и основной терминологией.

Практические задания: в этом уроке не предусмотрены.

Едва ли сегодня есть разработчик программного обеспечения (ПО), который бы ничего не слышал о технологии блокчейн (Blockchain), криптовалютах (Cryptocurrency или Crypto Currency), биткоинах (Bitcoin), первичном размещении монет (ICO, Initial coin offering), смарт-контрактах (Smart Contract), а также других понятиях и терминах, имеющих отношение к блокчейну.

Технология блокчейна открывает новые рынки и создает рабочие места для программистов. Если вы постигнете все тонкости криптовалютных технологий и технологий смарт-контрактов, то у вас не должно быть проблем с применением этих знаний на практике.

Надо сказать, что вокруг криптовалют и блокчейнов возникает много спекуляций. Мы оставим в стороне рассуждения об изменениях курсов криптовалют, о создании пирамид, о тонкостях криптовалютного законодательства и т.п.

В нашем учебном курсе мы сосредоточимся главным образом на технических аспектах применения смарт-контрактов блокчейна Ethereum (эфириум, эфир) и разработки так называемых децентрализованных приложений (Distributed Application, DApp).

Что такое блокчейн

Блокчейн (Blockchain, Block Chain) представляет собой цепочку блоков данных, связанных друг с другом определенным образом. В начале цепочки находится первый блок, который называется первичным блоком (genesis block) или блоком генезиса. За ним следует второй, потом третий и так далее.

Все эти блоки данных автоматически дублируются на многочисленных узлах сети блокчейна. Таким образом обеспечивается децентрализованное хранение данных блокчейна.

Вы можете представить себе систему блокчейна как большое количество узлов (физических или виртуальных серверов), объединенных в сеть и реплицирующих все изменения в цепочке блоков данных. Это как бы гигантский мультисерверный компьютер, причем узлы такого компьютера (серверы) могут быть разбросаны по всему миру. И вы тоже можете добавить свой компьютер в сеть блокчейна.

Распределенная база данных

Блокчейн можно представить себе как распределенную базу данных, реплицируемую на все узлы сети блокчейна. В теории блокчейн будет работоспособен до тех пор, пока ра-

ботает хотя бы один узел, хранящий все блоки блокчейна.

Распределенный реестр данных

Блокчейн можно представить себе как распределенный реестр данных и операций (транзакций). Еще одно название такого реестра – гроссбух.

В распределенный реестр можно добавлять данные, но невозможно их изменять или удалять. Такая невозможность достигается, в частности, применением криптографических алгоритмов, специальных алгоритмов добавления блоков в цепочку и децентрализованным хранением данных.

При добавлении блоков и выполнении операций (транзакций) используются приватные и публичные ключи. Они ограничивают пользователей блокчейна, предоставляя им доступ только к своим блокам данных.

Транзакции

Блокчейн хранит информацию об операциях (транзакциях) в блоках. При этом старые, уже выполненные транзакции невозможно откатить или изменить. Новые транзакции хранятся в новых, добавленных блоках.

Таким образом в блокчейне может быть записана в неизменном виде вся история транзакций. Поэтому блокчейн может быть использован, например, для надежного хранения

банковских операций, сведений об авторском праве, истории изменений владельцев объектов недвижимости и т.п.

Блокчейн Ethereum содержит так называемые состояния системы. По мере выполнения транзакций состояние изменяется от начального до текущего. Транзакции записываются в блоки.

Публичные и приватные блокчейны

Тут нужно отметить, что все сказанное верно только для так называемых *публичных* сетей блокчейн, которые не могут контролироваться никакими отдельными физическими или юридическими лицами, государственными органами или правительствами.

Так называемые *приватные* сети блокчейн находятся под полным контролем их создателей, и там возможно все, например, полная замена всех блоков цепочки.

Практические применения блокчейна

Для чего может пригодиться блокчейн?

Если кратко, блокчейн позволяет безопасным образом проводить операции (сделки) между не доверяющими друг другу персонами или компаниями. Записанные в блокчейн данные (транзакции, персональные данные, документы, свидетельства, договоры, накладные и т.п.) невозможно подде-

лать или заменить после записи. Поэтому на базе блокчейна можно создавать, например, доверенные распределенные реестры различного рода документов.

Конечно, вы знаете, что на базе блокчейнов создаются криптовалютные системы, призванные заменить обычные бумажные деньги. Бумажные деньги еще называют фиатными (от Fiat Money).

Блокчейн обеспечивает хранение и неизменность транзакций, записанных в блоки, поэтому его и можно использовать для создания криптовалютных систем. Он содержит всю историю передачи крипто-средств между различными пользователями (аккаунтами), причем любая операция может быть отслежена.

Хотя операции внутри криптовалютных систем могут быть анонимными, вывод криптовалюты и обмен ее на фиатные деньги обычно приводит к раскрытию личности владельца криптовалютного актива.

Так называемые смарт-контракты, представляющие собой программное обеспечение, работающее в сети Ethereum, позволяют автоматизировать процесс заключения сделок и контроль их выполнения. Особенно это эффективно, если оплата по сделке проводится криптовалютой Ether (эфир).

Блокчейн Ethereum и смарт-контракты Ethereum, написанные на языке программирования Solidity, могут использоваться, например, в таких областях:

• альтернатива нотариальному заверению документов;

• хранение реестра объектов недвижимости и сведений об операциях с объектами недвижимости;

• хранение сведений об авторских правах на объекты интеллектуальной собственности (книги, изображения, музыкальные произведения и т.п.);

• создание систем независимого голосования;

• сфера финансов и банковская деятельность;

• логистика в международных масштабах, отслеживание перемещения грузов;

• хранение персональных данных как аналог системе удостоверений личности;

• безопасные сделки в коммерческой области;

• хранение результатов медицинских обследований, а также истории назначенных процедур.

Проблемы с блокчейном

Но, разумеется, не все так просто, как могло бы показаться!

Есть проблемы с верификацией данных перед их добавлением в блокчейн (например, не поддельные ли они?), проблемы в безопасности системного и прикладного ПО, применяемого для работы с блокчейном, проблемы с возможностью использования методов социальной инженерии для похищения доступа к кошелькам с криптовалютой и т.п.

Опять же, если речь идет не о публичном блокчейне, уз-

лы которого разбросаны по всему миру, а о приватном блокчейне, принадлежащем персоне или организации, то уровень доверия здесь будет не выше, чем уровень доверия к этой персоне или этой организации.

Также следует учитывать, что данные, записанные в блокчейн, становятся *доступны для всех*. В этом смысле блокчейн (особенно публичный) не подходит для хранения конфиденциальной информации. Однако тот факт, что информацию в блокчейне невозможно изменить, может помочь предотвратить или расследовать различного рода мошеннические действия.

Децентрализованные приложения Ethereum будут удобны, если платить за их использование криптовалютой. Чем больше людей, владеющих криптовалютой или готовых ее приобрести, тем большую популярность получат приложения DApp и смарт-контракты.

Среди общих проблем блокчейна, затрудняющих его практическое применение, можно упомянуть ограниченную скорость добавления новых блоков и относительно высокую стоимость транзакций. Но технологии в этой области активно развиваются, и есть надежды, что технические проблемы со временем будут решены.

Еще одна проблема заключается в том, что смарт-контракты блокчейна Ethereum работают в изолированной среде виртуальных машин и не имеют доступа к данным реального мира. В частности, программа смарт-контракта не может

сама прочитать данные с сайтов или каких-либо физических устройств (датчики, контакты и т.п.), а также не может вывести данные на какие-либо внешние устройства. Эту проблему и способы ее решения мы будем обсуждать на уроке, посвященном так называемым Оракулам – информационным посредникам смарт-контрактов.

Также есть ограничения в области законодательства. В некоторых странах, например, запрещается использовать криптовалюту как платежное средство, но можно владеть ей как неким цифровым активом, наподобие ценных бумаг. Такие активы можно покупать и продавать на бирже. В любом случае, при создании проекта, работающего с криптовалютами, необходимо ознакомиться с законодательством той страны, под юрисдикцию которой попадает ваш проект.

Как формируется цепочка блокчейна

Как мы уже говорили, блокчейн представляет собой простую цепочку блоков данных. Сначала формируется первый блок этой цепочки, потом к нему добавляется второй и так далее. Предполагается, что данные транзакций хранятся в блоках и добавляются в самый последний блок.

На рис. 1.1. мы показали простейший вариант последовательности блоков, где первый блок ссылается на следующий.

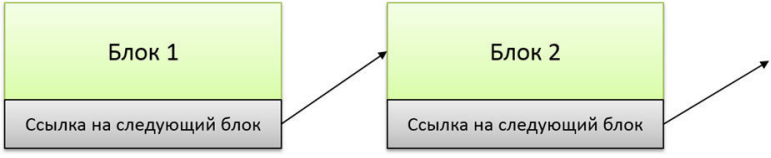

Рис. 1.1. Простая последовательность блоков

В таком варианте, однако, можно очень легко подделать содержимое любого блока цепочки, так как блоки не содержат никакой информации для защиты от изменений. Учитывая, что блокчейн предназначен для использований людей и компаний, между которыми нет доверия, можно сделать вывод, что такой способ хранения данных для блокчейна не подходит.

Давайте займемся защитой блоков от подделки. На пер-

вом этапе мы попробуем защитить каждый блок контрольной суммой (рис. 1.2.).

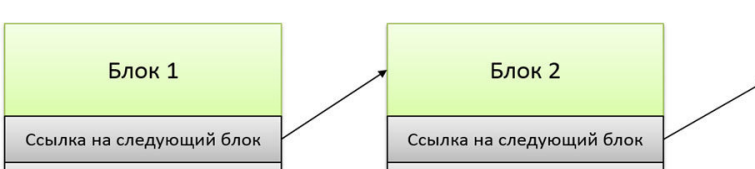

Рис. 1.2. Добавляем защиту данных блоков контрольной суммой

Теперь злоумышленник не может просто так изменить блок, так как в нем находится контрольная сумма данных блока. Проверка контрольной суммы покажет, что данные были изменены.

Для вычисления контрольной суммы можно использовать одну из функций хеширования, такую как MD-5, SHA-1, SHA-256 и т.п. Хеш-функции вычисляют некоторое значение (например, в виде текстовой строки постоянной длины) в результате выполнения необратимых операций над блоком данных. Операции зависят от вида хеш-функции.

Даже при небольшом изменении содержимого блока данных значение хеш-функции также изменится. Анализируя значение хеш-функции, невозможно восстановить блок данных, для которого она была вычислена.

Будет ли достаточна такая защита? К сожалению, нет.

В этой схеме контрольная сумма (хеш-функция) защищает только отдельные блоки, но не всю цепочку блоков. Зная алгоритм вычисления хеш-функции, злоумышленник может легко подменить содержимое блока. Также ничто не помешает ему удалить блоки из цепочки или добавить новые.

Чтобы защитить всю цепочку в целом, можно хранить в каждом блоке вместе с данными еще и хеш данных предыдущего блока (рис. 1.3.).

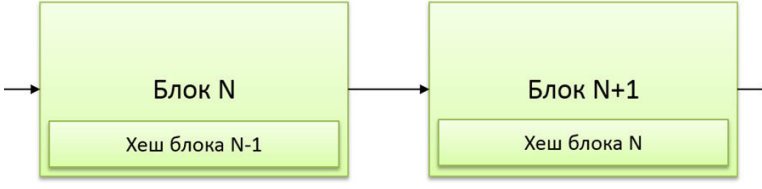

Рис. 1.3. Добавляем в блок данных хеш предыдущего блока

В этой схеме, чтобы изменить какой-нибудь блок, нужно пересчитать хеш-функции всех последующих блоков. Казалось бы, в чем проблема?

В реальных блокчейнах дополнительно созданы искусственные трудности для добавления новых блоков – используются алгоритмы, требующие много вычислительных ресурсов. С учетом того, что для внесения изменений в блок пересчитать нужно не один этот блок, а все последующие, сделать это будет крайне сложно.

Вспомним также, что данные блокчейна хранятся (дублируются) на многочисленных узлах сети, т.е. используется децентрализованное хранение. А это многократно усложняет подделку блока, т.к. изменения требуется внести на все узлы сети.

Так как блоки хранят информацию о предыдущем блоке, то можно проверить содержимое всех блоков цепочки.

Блокчейн Ethereum

Блокчейн Ethereum представляет собой платформу, на базе которой можно создавать распределенные приложения DApp. В отличие от других платформ, Ethereum позволяет использовать так называемые умные контракты (*смарт-контракты*, smart contracts), написанные на языке программирования Solidity.

Эта платформа была создана в 2013 году Виталиком Бутериным, основателем журнала Bitcoin Magazine, и запущена в 2015 году. Все, что мы будем изучать или делать в нашем учебном курсе, имеет отношение именно к блокчейну Ethereum и смарт-контрактам Solidity.

Майнинг, или Как создаются блоки

Майнинг (mining) представляет собой довольно сложный и ресурсоемкий процесс добавления новых блоков в цепочку блокчейна, а вовсе не «добычу криптовалют». Майнинг обеспечивает работоспособность блокчейна, т.к. именно этот процесс отвечает за добавление транзакций в блокчейн Ethereum.

Люди и организации, занимающиеся добавлением блоков, называются майнерами (miner).

Программное обеспечение (ПО), работающее на узлах майнеров, пытается подобрать для последнего блока параметр хеширования с названием Nonce, чтобы получилось определенное значение хеш-функции, заданной сетью. Алгоритм хеширования Ethash, применяемый в Ethereum, позволяет получить значение Nonce только путем последовательного перебора.

Если узел майнера нашел правильное значение Nonce, то это является так называемым доказательством работы (PoW, Proof-of-work). В этом случае, если блок будет добавлен в сеть Ethereum, майнер получает определенное вознаграждение в валюте сети – Ether. На момент подготовки нашей книги вознаграждение составляет 5 Ether, однако со временем оно будет снижено.

Таким образом, майнеры Ethereum обеспечивают работу

сети, добавляя блоки, и получают за это криптовалютные деньги. В интернете вы найдете массу информации о майнерах и майнинге, а мы сосредоточимся на создании контрактов Solidity и децентрализованных приложений DApp в сети Ethereum.

Итоги урока

На первом уроке вы познакомились с блокчейном и узнали, что он представляет собой особым образом составленную последовательностью блоков. Содержимое ранее записанных блоков невозможно изменить, так как для этого придется пересчитать все последующие блоки на многих узлах сети, на что нужно очень много ресурсов и времени.

Блокчейн может применяться для сохранения результатов выполнения транзакций. Его главное назначение – организация безопасного выполнения транзакций между сторонами (персонами и организациями), между которыми нет доверия. Вы узнали, в каких конкретно областях бизнеса и в каких сферах можно использовать блокчейн Ethereum и смарт-контракты Solidity. Это банковская сфера, регистрация прав собственности, документов и т.п.

Вы также узнали, что при использовании блокчейна могут возникать различные проблемы. Это проблемы верификации информации, добавляемой в блокчейн, скорость работы блокчейна, стоимость транзакций, проблема обмена данными между смарт-контрактами и реальным миром, а также потенциально возможные атаки злоумышленников, направленные на похищение криптовалютных средств с аккаунтов пользователей.

Также мы кратко рассказали о майнинге как о процессе

добавления новых блоков в блокчейн. Майнинг необходим для выполнения транзакций. Те, кто занимается майнингом, обеспечивают работоспособность блокчейна и получают за это вознаграждение в криптовалюте.

Урок 2. Подготовка рабочей среды в ОС Ubuntu и Debian

Цель урока: создать узел собственного приватного блокчейна Ethereum для дальнейшей работы в рамках этого курса на сервере Ubuntu и Debian.

Практические задания: установка операционной системы Ubuntu и Debian, установка ПО geth, обеспечивающего работу узла нашего блокчейна, а также демона децентрализованного хранилища данных swarm. Вам будет нужно создать собственный приватный блокчейн Ethereum, выполнить его инициализацию. Подключившись к узлу блокчейна, вы научитесь выдавать простейшие команды в приглашении geth.

Прежде чем мы займемся изучением смарт-контрактов Ethereum, нам необходимо подготовить рабочую среду – установить операционную систему (ОС) и необходимое программное обеспечение (ПО).

Мы могли бы приступить к работе сразу в какой-либо интегрированной среде разработки (IDE, Integrated Development Environment), например, Remix или Truffle. Возможно, это был бы самый быстрый путь к изучению смарт-контрактов Solidity. Однако для того, чтобы глубже разобраться в том, как работает Ethereum, мы начнем с ба-

зовых инструментов.

Выбор операционной системы

Свой первый узел сети Ethereum мы будем делать на базе клиента Go Ethereum (https://geth.ethereum.org/). Это ПО представляет собой реализацию протокола Ethereum и реализовано на языке программирования Go и доступно в виде программы Geth. На базе Geth можно создать полнофункциональный узел сети Ethereum.

Для работы с узлом Geth можно использовать интерфейс командной строки, а также программный интерфейс (API, Application Programming Interface) JSON RPC. Используя этот интерфейс и различные фреймворки, вы сможете создавать ПО, работающее с узлами Ethereum, практически на всех современных языках программирования.

Клиент Geth может работать на платформах, где имеется Go (это, например, Linux, Mac OSX, Windows, Raspberry Pi, Android OS, iOS). На странице загрузки https://geth.ethereum.org/downloads/ доступны реализации для Linux, macOS и Windows. Также вы можете загрузить исходные коды Geth.

При работе над книгой мы использовали ОС Ubuntu Live Server 18.04.2, Ubuntu 18.10 cosmic, Debian версий 9 и 10 Alfa 5, хотя Geth можно установить и на другие сборки Linux. На следующем уроке мы выполним установку Geth на ОС Rasberian для микрокомпьютера Raspberry Pi 3.

Для изучения Ethereum можно арендовать виртуальный или облачный сервер у одного из провайдеров. Также вы можете установить эти ОС на свой настольный компьютер, непосредственно на его диск или на виртуальную машину, например, VMware Workstation, или воспользоваться другой системой виртуализации.

Описание процесса установки Ubuntu и Debian выходит за рамки нашей книги, но в интернете есть немало достаточно подробных руководств, посвященных этому вопросу. Кроме того, при аренде виртуального или облачного сервера провайдер обязательно поможет установить на него ОС. Учтите, что вам нужна 64-разрядная версия ОС Ubuntu и Debian.

Для работы над книгой мы использовали облачный виртуальный сервер в следующей конфигурации:

- 4 ядра CPU с тактовой частотой 2 GHz;
- 2 GB RAM;
- 20 GB Disk SSD.

Мы также установили ОС Ubuntu 18.10 cosmic в виртуальную машину VMware Workstation с такой конфигурацией:

- 8 ядер CPU с тактовой частотой 2 GHz;
- 8 GB RAM;
- 100 GB Disk SATA.

Конфигурация виртуального сервера влияет на скорость майнинга новых блоков. Чтобы нам было удобно работать, эта скорость не должна быть слишком низкой. Не рекомендуем, в частности, использовать менее 2 Гбайт оперативной

памяти.

Для микрокомпьютеров Raspberri Pi, где памяти меньше, есть решение, о котором мы расскажем на следующем уроке.

Сразу после установки ОС Ubuntu или Debian обновите пакеты при помощи команды apt-get:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Если в Ubuntu Server обновилось ядро, может потребоваться перезагрузка ОС. Сообщение об этом вы увидите при подключении к консоли:

```
*** System restart required ***
```

В этом случае перед продолжением работы выполните перезагрузку:

```
$ sudo shutdown -r now
```

Пакеты Ubuntu можно также обновлять также через менеджер обновлений в графическом интерфейсе.

Обновление пакетов нужно делать с правами пользователя root.

Установка необходимых утилит

В ОС Ubuntu установите сервис ssh, если вы планируете подключаться к консоли удаленно (по умолчанию в десктопной версии сервис ssh не устанавливается, при установке Ubuntu Live Server нужно отметить соответствующий флажок).

В качестве имени пользователя при начальной установке ОС укажите book, чтобы у этого пользователя сразу была возможность работать с командой sudo.

Вы также можете создать пользователя book уже после установки Ubuntu. В этом случае при помощи команды visudo добавьте этому пользователю возможность работать с командой sudo. Для этого запустите с правами пользователя root такую команду:

```
# visudo
```

Откроется редактор файла /etc/sudoers. Вам нужно добавить в конец этого файла следующую строку:

```
book ALL=(ALL) ALL
```

Для установки сервиса ssh введите следующую команду:

```
$ sudo apt-get install ssh
```

Далее в Ubuntu и Debian установите редактор vim (если вам удобно в нем работать), утилиты sudo (если она не установлена), git, curl, gcc и mc (mc устанавливать не обязательно, пригодится, если только вы привыкли работать с Midnight Commander):

```
$ sudo apt-get install vim sudo git curl
gcc mc
```

Для того чтобы обезопасить ваш сервер от атак типа брутфорса (перебор паролей) на порт SSH, установите fail2ban:

```
$ sudo apt-get install fail2ban
```

Мы настоятельно рекомендуем использовать эту утилиту в рабочем окружении вместе с брандмауэром.

Установка Geth и Swarm в Ubuntu

Далее мы перейдем к установке Geth, а также ПО узла распределенного хранилища данных Swarm (потребуется позже, на 10 уроке).

Проще всего установить Geth в ОС Ubuntu. Процедура описана здесь: https://github.com/ethereum/go-ethereum/wiki/Installation-Instructions-for-Ubuntu.

Для установки выполните следующие команды:

```
$ sudo apt-get install software-properties-common
$ apt-get install build-essential
$ sudo add-apt-repository -y ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install ethereum
```

Вы также можете установить девелоперскую (нестабильную версию Geth), для чего выполните такую команду:

```
$ sudo apt-get install ethereum-unstable
```

После установки проверьте версию Geth:

```
$ geth version
Geth
Version: 1.8.23-stable
Git                                    Commit:
c942700427557e3ff6de3aaf6b916e2f056c1ec2
Architecture: amd64
Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.10.4
Operating System: linux
GOPATH=
GOROOT=/usr/lib/go-1.10
```

Как видите, здесь мы установили Geth стабильной версии 1.8.23 и Go версии 1.10.4.

Для установки распределенного хранилища данных Swarm на локальный тестовый узел используйте следующую команду:

```
$ sudo apt-get install ethereum-swarm
```

После установки проверьте версию Swarm:

```
$ swarm version
Swarm
Version: 0.3.11-stable
```

```
  Git                                   Commit:
c942700427557e3ff6de3aaf6b916e2f056c1ec2
  Go Version: go1.10.4
  OS: linux
```

Если установка прошла успешно, переходите к разделу урока, посвященного созданию приватного блокчейна.

В том случае, когда при установке произошли ошибки, попробуйте найти решение в поисковой системе Google. Заметим, что ошибки часто связаны с обновлением версий устанавливаемого ПО.

Установка Geth и Swarm в Debian

Установку Geth и Swarm в ОС Debian нужно выполнять из исходных текстов. При этом вначале нужно будет установить Go, а затем уже собственно Geth и Swarm.

На момент создания нашей книги была доступна версия Go 1.12.1. Заметим, что Geth и Swarm находятся в состоянии постоянного совершенствования. Не исключено, что к моменту, когда вы начнете работу над этой книгой, для них придется устанавливать новую версию Go.

Предварительная подготовка

Прежде всего обновите пакеты и установите необходимые утилиты:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ apt-get install vim sudo git curl gcc mc
```

Если вы при установке ОС не добавили пользователю book возможность работы с командой sudo, сделайте это аналогично тому, как это было описано ранее для Ubuntu.

После этого можно переходить к установке Go и Geth.

Загрузка дистрибутива Go

Дистрибутивы Go различных версий и для различных платформ можно найти здесь: https://golang.org/dl/.

Прежде всего подключимся к нашему серверу (физическому или виртуальному) пользователем book и загрузим архив Go нужной версии:

```
$ curl -O https://dl.google.com/go/go1.12.1.linux-amd64.tar.gz
```

Теперь, находясь в консоли с правами пользователя book, распаковываем загруженный архив в каталог /usr/local:

```
$ sudo tar -C /usr/local -xzf go1.12.1.linux-amd64.tar.gz
```

У вас будет запрошен пароль пользователя book. Команда sudo необходима, так как обычному пользователю запрещена запись файлов в каталог /usr/local.

Установка переменных окружения

Далее мы создаем в домашнем каталоге пользователя book каталог go и устанавливаем переменные окружения:

```
  $ mkdir -p ~/go; echo "export GOPATH=
$HOME/go" >> ~/.bashrc
  $ echo "export PATH=$PATH:$HOME/go/bin:/
usr/local/go/bin" >> ~/.bashrc
  $ source ~/.bashrc
```

Проверяем, что переменные окружения установлены:

```
  $ printenv | grep go

  GOPATH=/root/go
  PATH=/usr/local/sbin:/usr/local/bin:/
usr/sbin:/usr/bin:/sbin:/bin:/root/go/
bin:/usr/local/go/bin
```

Проверка версии Go

Прежде чем перейти собственно к установке Geth и Swarm, нужно проверить версию go:

```
  $ go version
  go version go1.12.1 linux/amd64
```

Если у вас версия 1.12.1, то все нормально. Но если ранее по каким-то причинам на вашем сервере была установлена

старая версия go из репозитория, удаляем ее так:

```
sudo apt-get remove golang-go
sudo apt-get remove -auto-remove golang-go
```

Установка Geth и Swarm

Первым шагом загрузите исходный код Geth из репозитория на GitHub:

```
$ mkdir -p $GOPATH/src/github.com/ethereum
$ cd $GOPATH/src/github.com/ethereum
$ git clone https://github.com/ethereum/go-ethereum
$ cd go-ethereum
$ git checkout master

$ go get github.com/ethereum/go-ethereum
```

Далее запустите компиляцию клиента Geth и Swarm:

```
go install -v ./cmd/geth
go install -v ./cmd/swarm
```

Если при компиляции появились ошибки, попробуйте установить Go другой версии. Перед этим удалите все каталоги, созданные в процессе предыдущей установки.

Если же все хорошо, то осталось только проверить версию установленной Geth и Swarm:

```
$ geth version
Geth
Version: 1.9.0-unstable
Architecture: amd64
Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.12.1
Operating System: linux
GOPATH=/home/book/go
GOROOT=/usr/local/go

$ swarm version
Swarm
Version: 0.3.12-unstable
Go Version: go1.12.1
OS: linux
```

Как видите, были установлены нестабильные версии Geth и Swarm. С помощью whereis вы можете определить, в какой каталог была выполнена установка:

```
$ whereis geth
geth: /home/book/go/bin/geth
```

Чтобы установить стабильную версию, загрузите ее бинарный код с сайта https://geth.ethereum.org/downloads/. Затем извлеките из архива программу geth и скопируйте в отдельный каталог.

Актуальную инструкцию по установке Geth и Swarm можно найти по адресу https://media.readthedocs.org/pdf/swarm-guide/latest/swarm-guide.pdf.

Создаем приватный блокчейн

Для того чтобы быстро освоить процесс создания смарт-контрактов, нам нужен блокчейн. Вначале создадим приватный блокчейн на своем сервере, так как с ним проще всего работать. Далее мы будем использовать тестовую сеть Rinkeby, работающую в точности как Ethereum и пригодную для отладки «настоящих» контрактов перед публикацией в Ethereum.

Готовим файл genesis.json

Прежде всего создайте в домашнем каталоге пользователя book файл genesis.json (листинг 2.1.).

Листинг 2.1. Файл genesis.json

```
{
 "config": {
    "chainId": 1999,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
 },
 "difficulty": "10",
 "gasLimit": "5100000",
```

```
  "alloc": {}
}
```

Этот файл описывает первичный блок (genesis block) цепочки блоков – самый первый блок блокчейна. Так как мы создаем наш приватный блокчейн, то и о первом блоке мы должны позаботиться сами.

Хорошее описание первичного блока сети Ethereum можно найти, например, здесь: https://arvanaghi.com/blog/explaining-the-genesis-block-in-ethereum/.

Блок **config** файла genesis.json содержит переменные конфигурации сети Ethereum.

Поле **chainId** содержит идентификатор блокчейна. Он используется для защиты от репликации, т.е. от неавторизованных действий пользователей, пытающихся действовать от имени настоящего отправителя данных.

Если речь идет о главной сети Ethereum, то ее идентификатор равен 1. Идентификатор тестовой сети Rinkeby равен 4. При создании нашей приватной сети мы можем указать любое значение, отличное от известных. Мы выбрали значение 1999.

Значение 0 в поле **homesteadBlock** указывает на то, что мы будем использовать релиз сети Ethereum под названием Homestead. Homestead представляет собой второй релиз сети Ethereum (первый релиз назывался Frontier). Нулевое значение этого параметра используется и в основной сети

Ethereum.

Немного о параметрах с префиксом EIP. При реализации Homestead был обновлен протокол, причем без обратной совместимости с предыдущим релизом Ethereum. Соответствующие изменения отражены в документах «Предложения по улучшению Ethereum» (Ethereum Improvement Proposals, EIPs), опубликованных на сайте https://eips.ethereum.org/.

Реализация изменений может потребовать выполнения так называемой процедуры хардфорка (hard-forking), или обновления ПО узлов. В результате этой процедуры может меняться структура блока, появляется возможность использовать ранее недоступные блоки и вносить различные изменения в протокол.

Так как мы не будем делать хардфорк, то установим значения соответствующих параметров **eip155Block** и **eip158Block** равными нулю.

Параметр **difficulty** важен для нас в практическом смысле. Этот параметр имеет прямое влияние на время генерации новых блоков блокчейна. Для нашего «учебного» блокчейна мы установим очень маленькое значение этого параметра, равное 10, чтобы скорость добавления новых блоков была приемлемой даже на виртуальных серверах небольшой производительности.

С помощью параметра **gasLimit** мы задаем в рамках блокчейна предел расхода так называемого газа (Gas). Газ Ethereum представляет собой расходный ресурс, который

тратится на выполнение таких операций, как отправка транзакций, публикация и выполнение контрактов и т.п. Далее мы расскажем об этом подробнее. В нашей приватной тестовой сети мы устанавливаем достаточно большое значение, чтобы не возникали ограничения при запуске тестовых программ.

Параметр **alloc** позволяет при инициализации блокчейна создать кошельки и заранее наполнить их тестовым эфиром. В первых примерах мы не будем использовать эту возможность.

Создаем каталог для работы

Создайте в домашнем каталоге подкаталог node1:

```
$ mkdir node1
```

В этом каталоге будут располагаться данные блокчейна.

Создаем аккаунт

Теперь перейдем к созданию нашего приватного блокчейна. Прежде всего войдите в домашний каталог пользователя book и создайте новый аккаунт:

```
$ geth –datadir node1 account new
```

При создании аккаунта будет запрошен пароль, который нужно сохранить в безопасном месте:

```
  INFO [02-13|08:42:28.798] Maximum peer
count                              ETH=25
LES=0 total=25
  Your new account is locked with a
password. Please give a password. Do not
forget this password.
  Passphrase:
  Repeat passphrase:
  Address:
{ 4f744742ac711fd111c7a983176db1d48d29f413
```

Команда **account new** выведет на консоль в фигурных скобках так называемый адрес узла. В нашем случае это адрес 4f744742ac711fd111c7a983176db1d48d29f413. Мы будем указывать его в различных командах.

Параметр **datadir** команды geth указывает путь к рабочему каталогу. Мы используем каталог /home/book/node1.

Запускаем инициализацию узла

После создания аккаунта нам нужно выполнить инициализацию узла, выполняем ее из домашнего каталога пользо-

вателя book:

```
$ geth –datadir node1 init genesis.json
```

Здесь мы с помощью параметра **datadir** должны указать путь к рабочему каталогу, а в параметре **init** – путь к файлу первичного блока genesis.json.

Команда выполнит инициализацию и выведет на консоль результаты своей работы:

```
INFO [02-13|08:43:53.934] Maximum peer
count                                ETH=25
LES=0 total=25
INFO [02-13|08:43:53.936] Allocated
cache and file handles         database=/
home/book/node1/geth/chaindata   cache=16
handles=16
INFO [02-13|08:43:53.950] Writing custom
genesis block
INFO [02-13|08:43:53.950] Persisted trie
from memory database     nodes=0 size=0.00B
time=28.058µs    gcnodes=0    gcsize=0.00B
gctime=0s livenodes=1 livesize=0.00B
INFO [02-13|08:43:53.951] Successfully
wrote                                genesis
state  database=chaindata     hash=a5e5bc…
```

```
3f490e
  INFO [02-13|08:43:53.951] Allocated
cache and file handles database=/
home/book/node1/geth/lightchaindata
cache=16 handles=16
  INFO [02-13|08:43:53.955] Writing custom
genesis block
  INFO [02-13|08:43:53.955] Persisted trie
from memory database nodes=0 size=0.00B
time=1.778µs gcnodes=0 gcsize=0.00B
gctime=0s livenodes=1 livesize=0.00B
  INFO [02-13|08:43:53.956] Successfully
wrote genesis
state database=lightchaindata hash=a5e5bc…
3f490e
```

Для работы с узлом вам нужно будет открыть две консоли, подключившись в каждой консоли пользователем book.

Чтобы запустить узел, выполните в первой консоли следующую команду:

```
  $ geth -etherbase
"0x4f744742ac711fd111c7a983176db1d48d29f41
datadir node1 -nodiscover -mine -
minerthreads 1 -maxpeers 0 -
verbosity 3 -networkid 98760 -rpc -
```

```
rpcapi="db,eth,net,web3,personal,web3"
console
```

В качестве параметра –etherbase нужно ввести адрес узла, полученный при первоначальном создании аккаунта.

На экране появится множество сообщений о ходе инициализации. В ходе этого процесса будет запущена генерация файла DAG. Вам нужно будет дождаться завершения процесса генерации:

```
  INFO [02-13|08:51:16.647] Maximum peer
count                      ETH=0 LES=0 total=0
  INFO [02-13|08:51:16.649] Starting peer-
to-peer node               instance=Geth/
v1.8.22-stable-7fa3509e/linux-amd64/
go1.10.4
  INFO [02-13|08:51:16.649] Allocated
cache and file handles     database=/
home/book/node1/geth/chaindata  cache=512
handles=524288
  INFO [02-13|08:51:16.662] Initialised
chain
configuration              config="{ChainID:
1999 Homestead: 0 DAO: <nil> DAOSupport:
false EIP150: <nil> EIP155: 0 EIP158:
0 Byzantium: <nil> Constantinople:
```

```
<nil>  ConstantinopleFix: <nil> Engine:
unknown}"
  INFO [02-13|08:51:16.663] Disk storage
enabled for ethash caches  dir=/home/book/
node1/geth/ethash count=3
  INFO [02-13|08:51:16.663] Disk storage
enabled for ethash DAGs       dir=/home/
book/.ethash           count=2
  INFO [02-13|08:51:16.663] Initialising
Ethereum protocol           versions="[63
62]" network=98760
  INFO [02-13|08:51:16.724] Loaded most
recent local header           number=0
hash=a5e5bc…3f490e td=10 age=49y10mo16h
  INFO [02-13|08:51:16.724] Loaded most
recent local full block       number=0
hash=a5e5bc…3f490e td=10 age=49y10mo16h
  INFO [02-13|08:51:16.724] Loaded most
recent local fast block       number=0
hash=a5e5bc…3f490e td=10 age=49y10mo16h
  INFO [02-13|08:51:16.724] Loaded local
transaction journal        transactions=0
dropped=0
  INFO [02-13|08:51:16.725] Regenerated
local transaction journal   transactions=0
accounts=0
```

```
  INFO [02-13|08:51:16.732] New local node
record                                    seq=3
id=eae4aa1f2059eed4   ip=127.0.0.1   udp=0
tcp=30303
  INFO [02-13|08:51:16.732] Started P2P
networking                                self=
enode://6ef6903c1e19600cf7f6c8b5@127.0.0.1:30303?
discport=0"
  INFO [02-13|08:51:16.732] IPC endpoint
opened                                    url=/home/
book/node1/geth.ipc
  INFO [02-13|08:51:16.733] HTTP endpoint
opened    url=http://127.0.0.1:8545 cors=
vhosts=localhost
  INFO [02-13|08:51:16.733] Transaction
pool      price      threshold      updated
price=1000000000
  INFO [02-13|08:51:16.733] Updated mining
threads                           threads=1
  INFO [02-13|08:51:16.733] Transaction
pool      price      threshold      updated
price=1000000000
  INFO [02-13|08:51:16.734] Commit new
mining work                               number=1
sealhash=5c4116…c8c1bf   uncles=0   txs=0
gas=0 fees=0 elapsed=562.141µs
  INFO [02-13|08:51:16.779] Mapped network
```

```
port                                proto=tcp
extport=30303 intport=30303 interface=NAT-
PMP(192.168.0.1)
  Welcome to the Geth JavaScript console!

  instance: Geth/v1.8.22-stable-7fa3509e/
linux-amd64/go1.10.4
  coinbase:
0x4f744742ac711fd111c7a983176db1d48d29f413
  at block: 0 (Wed, 31 Dec 1969 16:00:00
PST)
  datadir: /home/book/node1
  modules: admin:1.0 debug:1.0 eth:1.0
ethash:1.0 miner:1.0 net:1.0 personal:1.0
rpc:1.0 txpool:1.0 web3:1.0

  > INFO [02-13|08:51:18.459] Generating
DAG in progress                     epoch=0
percentage=0 elapsed=983.779ms
  INFO [02-13|08:51:19.420] Generating DAG
in progress              epoch=0 percentage=1
elapsed=1.944s
  INFO [02-13|08:51:20.395] Generating DAG
in progress              epoch=0 percentage=2
elapsed=2.919s
  INFO [02-13|08:51:21.440] Generating DAG
```

```
in progress              epoch=0 percentage=3
elapsed=3.963s
  …

  INFO [02-13|08:55:56.193] Successfully
sealed new block                  number=46
sealhash=fccbc1…5cc27f hash=776967…3700c9
elapsed=7.306s
  INFO [02-13|08:55:56.193] 🔗 block
reached canonical chain           number=39
hash=9c6ba7…c95452
  INFO [02-13|08:55:56.193] ⛏ mined
potential block                   number=46
hash=776967…3700c9
  INFO [02-13|08:55:56.193] Commit new
mining work                       number=47
sealhash=41558f…3ef931   uncles=0   txs=0
gas=0 fees=0 elapsed=97.707µs
  >     [1;5FINFO     [02-13|08:56:01.030]
Successfully          sealed          new
block          number=47 sealhash=41558f…
3ef931 hash=f67a9b…773bfd elapsed=4.837s
  INFO [02-13|08:56:01.030] 🔗 block
reached canonical chain           number=40
hash=3a6600…bde7e0
```

```
  INFO [02-13|08:56:01.030]  mined
potential block                  number=47
hash=f67a9b…773bfd
  INFO [02-13|08:56:01.030] Commit new
mining work                      number=48
sealhash=d4ab02…7151c7  uncles=0  txs=0
gas=0 fees=0 elapsed=97.374µs
```

В зависимости от производительности вашего виртуального или физического сервера генерация файла DAG может занять несколько минут или дольше.

Файл DAG содержит направленный ациклический граф (Directed Acyclic Graph). Он используется для добавления блоков в Ethereum с помощью алгоритма с названием Ethash. Его размер может составлять более 1 Гбайта. Размер этого блока увеличивается по мере роста сети Ethereum. Текущий размер блока можно узнать, например, на сайте https://investoon.com/tools/dag_size. На момент создания этого учебного курса файл DAG для основной сети Ethereum был размером 2.95 Гбайт.

Чем больше файл DAG, тем труднее выполнить майнинг. Если для майнинга используются видеокарты, то данные DAG должны полностью поместиться в память видеокарты, иначе применение видеокарты для майнинга будет бесполезным.

Так как запуск узла вы будете выполнять часто, рекомендуем подготовить пакетный файл для запуска с именем, например, start_node.sh (листинг 2.2.).

Листинг 2.2. Файл start_node.sh

```
geth -etherbase
"0x4f744742ac711fd111c7a983176db1d48d29f411
datadir node1 -nodiscover -mine -
minerthreads 1 -maxpeers 0 -
verbosity 3 -networkid 98760 -rpc -
rpcapi="db,eth,net,web3,personal,web3"
console
```

Параметры запуска узла

Расскажем о параметрах geth, которые мы использовали при запуске узла. Эти параметры были выбраны исходя из назначения нашего узла – мы создаем узел для учебной приватной сети Ethereum. Когда вы будете создавать узел для работы с тестовой сетью Rinkeby или с основной сетью Ethereum, нужно будет указывать другие параметры.

Чтобы получить краткую справку по всем командам и параметрам geth, запустите ее следующим образом:

```
$ geth -h
```

С параметром **datadir**, который указывает путь к каталогу блокчейна, вы уже знакомы. При запуске узла укажите тот же каталог, что мы использовали при инициализации приватного блокчейна.

Параметр **etherbase** задает публичный адрес, на который будет отправлено вознаграждение за майнинг.

Параметр **nodiscover** отключает поиск других узлов сети. Мы указали его, так как пока будем работать только с одним узлом блокчейна.

Мы также указали значение параметра **maxpeers**, равное нулю. Таким способом мы фактически отключили обмен по сети между узлами нашего блокчейна.

С помощью параметра **mine** мы запускаем так называемый майнинг – процесс создания новых блоков в нашем блокчейне. Это необходимо, так как без появления новых блоков выполнение транзакций и публикация смарт-контрактов будут невозможны.

Параметр **minerthreads** указывает количество потоков, используемых для майнинга. Если ресурсы вашего сервера позволяют и там установлен многоядерный процессор, то для ускорения майнинга можно увеличить значение этого параметра.

Очень важный параметр – **networkid**. Это идентификатор сети. Здесь мы должны указать уникальный идентификатор 98760 нашего приватного блокчейна.

Параметр **verbosity** задает детализацию журнала:

• 0 – не записывать данные в журнал;
• 1 – записывать сообщения об ошибках;
• 2 – записывать предупреждающие сообщения;
• 3 – записывать информационные сообщения;
• 4 – записывать отладочную информацию;
• 5 – записывать детальную информацию.

По умолчанию используется значение 3.

Так как мы будем работать с узлом с помощью протокола JSON RPC, нам необходимо включить такую возможность, указав параметр **rpc**. Дополнительно с помощью параметра **rpcapi** мы перечисляем, какие программные интерфейсы должен предоставить нам узел. Здесь мы указали такой набор: db, eth, net, web3, personal, web3. Подробнее об этом мы расскажем позже.

При запуске geth мы указываем команду **console**. Эта команда запускает интерактивную консоль JavaScript, где мы сможем выдавать команды.

Подключаемся к нашему узлу

Теперь откройте вторую консоль и введите в ней следующую команду:

```
$ geth -datadir node1 -networkid 98760
attach ipc://home/book/node1/geth.ipc
```

Эта команда откроет консоль geth и подключится к вашему приватному узлу.

Здесь необходимо указать те же значения параметров **datadir** и **networked**, что и при запуске узла. Команда **attach** подключается к узлу, адрес которого задан после нее, и запускает интерактивную консоль JavaScript. В адресе нам нужно указать полный путь к рабочему каталогу нашего приватного блокчейна.

Запишите команду подключения в файл attach_node.sh для удобства (листинг 2.3.).

Листинг 2.3. Файл attach_node.sh

```
geth -datadir node1 -networkid 98760
attach ipc://home/book/node1/geth.ipc
```

Теперь запустите этот файл, и вы увидите приглашение консоли geth:

```
$ sh attach_node.sh
Welcome to the Geth JavaScript console!

instance: Geth/v1.8.22-stable-7fa3509e/
linux-amd64/go1.10.4
coinbase:
0x3cd46aab0631305437842cf639218e41ce946baa
```

```
  at block: 379 (Wed, 13 Feb 2019 09:12:48
PST)
  datadir: /home/book/node1
  modules:  admin:1.0  debug:1.0  eth:1.0
ethash:1.0 miner:1.0 net:1.0 personal:1.0
rpc:1.0 txpool:1.0 web3:1.0

  >
```

Введите в этом приглашении команду web3.eth.accounts:

```
  > web3.eth.accounts
  ["0x4f744742ac711fd111c7a983176db1d48d29
```

Вы увидите идентификатор (адрес) аккаунта, который мы создали ранее, указав для него пароль. У вас этот идентификатор будет другой.

Попробуйте также ввести команду web3.version. Эта команда позволяет посмотреть версию фреймворка Web3, с помощью которого мы будем работать с контрактами, версию geth, а также номер сети. Мы задали номер нашей приветной сети, равный 98760.

Для стабильного релиза Geth версии 1.8.22 на консоль будет выведено сообщение:

```
  > web3.version
```

```
{
 api: "0.20.1",
 ethereum: "0x3f",
 network: "98760",
  node:  "Geth/v1.8.22-stable-7fa3509e/
linux-amd64/go1.10.4",
 whisper: undefined,
 getEthereum: function(callback),
 getNetwork: function(callback),
 getNode: function(callback),
 getWhisper: function(callback)
}
```

Использование нестабильной версии Geth будет отмечено в поле node:

```
> web3.version
{
 api: "0.20.1",
 ethereum: "0x3f",
 network: "98760",
  node:  "Geth/v1.8.11-unstable/linux-
amd64/go1.9.6",
 whisper: undefined,
 getEthereum: function(callback),
 getNetwork: function(callback),
```

```
  getNode: function(callback),
  getWhisper: function(callback)
 }
```

Управление майнингом и проверка баланса

При запуске узла мы автоматически запускаем майнинг. Текущий баланс вы можете проверить при помощи следующей команды:

```
 >
web3.fromWei( eth.getBalance(eth.coinbase)
 0
```

Позже мы опишем экономику Ethereum и эти команды детальнее.

Сразу после инициализации сети баланс нашего аккаунта равен нулю. Однако по мере того, как будут «добыты» новые блоки, баланс будет расти:

```
 >
web3.fromWei( eth.getBalance(eth.coinbase)
 15
```

В первой консоли, где мы запустили узел, добавление каж-

дого блока будет отмечено такими сообщениями:

```
  INFO [02-13|09:37:10.577] 🔗 block
reached canonical chain          number=45
hash=924ce1…d8b5a2

  INFO [02-13|09:37:10.577] 🔨 mined
potential block                  number=52
hash=a80a36…153593
  INFO [02-13|09:37:10.577] Commit new
mining work                      number=53
sealhash=3acb6c…0ecd19  uncles=0  txs=0
gas=0 fees=0 elapsed=130.557µs
  INFO [02-13|09:37:11.223] Successfully
sealed new block                 number=53
sealhash=3acb6c…0ecd19 hash=14e0fa…575494
elapsed=645.999ms
  INFO [02-13|09:37:11.223] 🔗 block
reached canonical chain          number=46
hash=c5ff7a…da8069

  INFO [02-13|09:37:11.224] 🔨 mined
potential block                  number=53
hash=14e0fa…575494
  INFO [02-13|09:37:11.224] Commit new
mining work                      number=54
sealhash=96235b…f3fc50  uncles=0  txs=0
```

```
gas=0 fees=0 elapsed=124.053µs
  INFO [02-13|09:37:11.723] Successfully
sealed new block                  number=54
sealhash=96235b…f3fc50 hash=e5438e…2f6f2e
elapsed=498.975ms
```

С помощью команд miner.start и miner.stop можно запускать и останавливать майнинг.

При ручном запуске майнинга нужно указать количество потоков для поиска новых блоков:

```
> miner.start(4)
```

Здесь мы запускаем майнинг на четырех ядрах виртуальной машины. Перед запуском проверьте, сколько ядер доступно на вашем сервере.

Заметим, что при отладке смарт-контрактов процесс майнинга останавливать не нужно, иначе ваш узел не сможет обрабатывать транзакции, публиковать контракты и вызывать методы контрактов. Тем не менее, вы всегда сможете остановить майнинг с помощью такой команды:

```
> miner.stop()
```

Если скорость майнинга недостаточна и вам приходится ждать появления новых блоков более 20-30 секунд, попро-

буйте увеличить размер оперативной памяти и количество процессорных ядер на виртуальной машине. Облачные хостинги, как правило, позволяют сделать это очень просто через Web-интерфейс вашего личного кабинета.

Завершение работы консоли Geth

Для завершения работы Geth введите в приглашении команду exit:

```
> exit
```

Итоги урока

На втором уроке вы подготовили рабочую среду, необходимую для дальнейшего изучения Ethereum и смарт-контрактов Solidity. Вы создали сервер с ОС Ubuntu или Debian, установили Go, Geth и программу распределенного хранилища данных Swarm.

Далее вы создали приватный блокчейн, состоящий из одного узла, выполнили инициализацию этого узла и убедились, что в вашем блокчейне работает майнинг. Вы научились запускать свой узел Ethereum и подключаться к нему в консоли. Вы также научились выдавать простейшие консольные команды Geth и теперь готовы для первых экспериментов с вашим приватным блокчейном.

Урок 3. Подготовка рабочей среды на Raspberry Pi 3

Цель урока: создать узел собственного приватного блокчейна Ethereum для дальнейшей работы в рамках этого курса на микрокомпьютере Raspberry Pi 3 с операционной системой Rasberian. Изучить способ запуска узла приватной сети Ethereum при помощи параметра –dev утилиты Geth.

Практические задания: установка операционной системы Rasberian на Raspberry Pi 3, установка ПО Geth, обеспечивающего работу узла нашего блокчейна, а также демона децентрализованного хранилища данных Swarm. Запуск узла приватного блокчейна Ethereum с помощью параметра –dev утилиты geth.

На предыдущем уроке вы научились создавать свой приватный блокчейн Ethereum на сервере с ОС Ubuntu и Debian. Возможно, для первых опытов и отладки смарт-контрактов вам больше подойдет очень недорогой микрокомпьютер Raspberry Pi 3 (или его более новая модель Raspberry Pi 3 model B), который создавался как раз для обучения.

К сожалению, ресурсы микрокомпьютера Raspberry Pi довольно ограничены, в частности, объем оперативной памяти составляет всего 1 Гбайт (уже есть в продаже Raspberry Pi 4 с 4 Гбайт памяти на борту). Поэтому на Raspberry Pi не по-

лучится запустить майнинг в Geth аналогично тому, как мы это делали на предыдущем уроке в ОС Ubuntu и Debian.

Тем не менее, выход есть.

Если раньше вы создавали файл первичного блока, инициализировали блокчейн и запускали майнинг, то на этом уроке мы будем использовать другой, возможно, более удобный способ создания тестовой сети. Мы создадим ее, запустив Geth с параметром –dev, специально предназначенным для этого случая. Разумеется, такой способ можно применять и на обычных серверах, физических и виртуальных.

Подготовка Raspberry Pi 3 к работе

Для первоначальной установки ОС Rasberian вам нужно подключить к микрокомпьютеру блок питания, монитор (лучше всего через интерфейс HDMI), клавиатуру (через интерфейс USB). Также нужно подключить микрокомпьютер кабелем Ethernet к вашему роутеру.

Установите в микрокомпьютер карту памяти MicroSD объемом не меньше 8 Гбайт.

После того как установка и настройка Rasberian будут завершены, монитор и клавиатуру можно будет отключить.

Хорошую инструкцию по начальной подготовке Raspberry Pi 3 к работе можно найти здесь: http://dmitrysnotes.ru/raspberry-pi-3-obzor-i-nachalo-raboty.

Установка Rasberian

Прежде всего нужно отформатировать карту памяти MicroSD. В ОС Microsoft Windows воспользуйтесь для этого бесплатной утилитой SD Memory Card Formatter, загрузив ее с сайта https://www.sdcard.org/downloads/formatter_4/. Стандартные средства форматирования ОС Microsoft Windows не подойдут.

Далее нужно скачать дистрибутив Raspbian с сайта https://www.raspberrypi.org/downloads/raspbian/. Для наших целей

подойдет версия Raspbian Stretch Lite. Загрузите архив Zip и распакуйте его. В итоге вы получите файл с расширением имени img, например, 2018-04-18-raspbian-stretch-lite.img.

Запишите этот файл на предварительно отформатированную карту памяти MicroSD. Это можно сделать с помощью бесплатной утилиты Rufus. Загрузите ее с сайта https://rufus.akeo.ie/.

Далее установите подготовленную таким способом карту в Raspberry Pi и включите питание. Проследите процесс начальной загрузки Rasberian на мониторе.

В первый раз для подключения через консоль используйте имя пользователя pi и пароль raspberry. После подключения сразу смените пароль с помощью утилиты passwd.

Установка обновлений

Сразу после установки микрокомпьютер получит динамический адрес IP от сервера DHCP вашего роутера, и вы сможете продолжить установку.

Прежде всего установите обновления пакетов до актуальной версии:

```
sudo apt-get update
sudo apt-get upgrade
```

Включение доступа SSH

Чтобы с микрокомпьютером можно было работать удаленно через SSH, нужно включить такую возможность. Для этого в консоли введите следующую команду:

```
sudo raspi-config
```

Выберите в списке строку **5. Interfacing options**, найдите в списке **SSH** и включите.

Установка статического адреса IP

Для удаленной работы через SSH будет удобно, если микрокомпьютеру будет выделен статический адрес IP. Чтобы это сделать, отредактируйте файл /etc/dhcpcd.conf:

```
sudo nano /etc/dhcpcd.conf
```

Уберите символы комментария # со следующих строк:

```
interface eth0
static ip_address=192.168.0.38/24
static ip6_address=fd51:42f8:caae:d92e::
static routers=192.168.0.1
```

```
static domain_name_servers=192.168.0.1 8
```

Также в строке static ip_address укажите нужный адрес IP. Заметим, что этот адрес нужно исключить из DHCP соответствующей настройкой вашего роутера.

После изменения файла перезагрузите микрокомпьютер:

```
sudo shutdown -r now
```

Теперь вы можете подключиться к нему удаленно с помощью любого клиента SSH по статическому адресу. Если удаленный доступ работает, вы можете отключить клавиатуру и монитор от Raspberry Pi и продолжить работу через SSH.

Дальнейшие действия очень похожи на то, что мы делали в предыдущем уроке. Однако есть и отличия.

Установка необходимых утилит

Как и на прошлом уроке, мы рекомендуем вам установить редактор vim, утилиты sudo и git:

```
# apt-get install vim
# apt-get install sudo
# apt-get install git
```

Создайте пользователя book и при помощи команды visudo добавьте этому пользователю возможность работать с командой sudo, как мы это делали на предыдущем уроке.

Установка Go

Мы будем работать с Go версии 1.9.6. Установка для Rasbian очень похожа на установку для Debian, однако вам потребуется дистрибутив, рассчитанный на архитектуру процессора ARM.

Загрузка дистрибутива Go

Как мы уже говорили, дистрибутивы Go различных версий и для различных платформ можно найти здесь: https://golang.org/dl/.

Подключимся к нашему микрокомпьютеру Raspberry Pi пользователем book и загрузим архив Go нужной версии:

```
$ curl -O https://dl.google.com/go/go1.9.6.linux-armv6l.tar.gz
```

Как и на предыдущем уроке, вам нужно добавить возможность работать пользователю book с командой sudo, для чего нужно воспользоваться командой visudo.

Добавьте в конец файла /etc/sudoers строку:

```
book ALL=(ALL) ALL
```

Подключитесь к консоли с правами пользователя book и распакуйте загруженный архив в каталог /usr/local:

```
$ sudo tar -C /usr/local -xzf
go1.9.6.linux-armv6l.tar.gz
```

Обратите внимание, что здесь мы распаковываем файл дистрибутива для процессоров ARM.

Установка переменных окружения

Создайте в домашнем каталоге пользователя book каталог go и установите переменные окружения:

```
$ mkdir -p ~/go; echo "export GOPATH=
$HOME/go" >> ~/.bashrc
$ echo "export PATH=$PATH:$HOME/go/bin:/
usr/local/go/bin" >> ~/.bashrc
$ source ~/.bashrc
```

Проверьте, что переменные окружения установлены:

```
$ printenv | grep go

GOPATH=/root/go
PATH=/usr/local/sbin:/usr/local/bin:/
```

```
usr/sbin:/usr/bin:/sbin:/bin:/root/go/
bin:/usr/local/go/bin
```

Проверка версии Go

Прежде чем перейти собственно к установке Geth и Swarm, нужно проверить версию go. Теперь должно быть указано, что установлена версия 1.9.6 для процессоров ARM:

```
$ go version
go version go1.9.6 linux/arm
```

Установка Geth и Swarm

Установка Geth и Swarm выполняется аналогично тому, как мы это делали на предыдущем уроке.

Загрузите исходный код Geth из репозитория на GitHub:

```
$ mkdir -p $GOPATH/src/github.com/ethereum
$ cd $GOPATH/src/github.com/ethereum

$ git clone https://github.com/ethereum/go-ethereum
$ cd go-ethereum
$ git checkout master

$ go get github.com/ethereum/go-ethereum
```

Запустите компиляцию клиента Geth и Swarm:

```
go install -v ./cmd/geth
go install -v ./cmd/swarm
```

Если при компиляции появились ошибки, попробуйте установить Go другой версии. Перед этим удалите все каталоги, созданные в процессе предыдущей установки.

Если же все хорошо, то осталось только проверить версию установленного Geth и Swarm:

```
$ geth version
Geth
Version: 1.8.9-unstable
Architecture: arm

Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.9.6
Operating System: linux
GOPATH=/home/book/go
GOROOT=/usr/local/go

$ swarm version
Swarm
Version: 1.8.9-unstable
Network Id: 0
Go Version: go1.9.6
OS: linux
GOPATH=/home/book/go
GOROOT=/usr/local/go
```

Обратите внимание, что команда geth version сообщает о том, что она работает на процессоре с архитектурой ARM.

Создаем приватный блокчейн

На предыдущем уроке мы создавали приватный блокчейн, подготовив для него первичный блок в файле genesis.json. Затем мы создали в домашнем каталоге пользователя book рабочий каталог node1, создали аккаунт, запустили инициализацию узла и, наконец, запустили узел нашего блокчейна. При этом был создан файл DAG с направленным ациклическим графом и запущен майнинг. В другой консоли мы подключились к нашему узлу и выдали там несколько команд Web3.

На этот раз мы сделаем все намного проще. Создадим рабочий каталог node1 для размещения данных блокчейна:

```
$ mkdir node1
```

Теперь запустим узел приватной сети при помощи следующей команды:

```
$ geth -datadir node1 -networkid 98760 -dev -rpc -rpcapi="db,eth,net,web3,personal,web3" console
```

В окне консоли появятся сообщения о запуске узле сети.

Сохраните команду запуска узла в пакетном файле с именем start_node.sh (листинг 3.1.).

Листинг 3.1. Файл start_node.sh

```
geth -datadir node1 -networkid 98760 -dev -rpc -rpcapi="db,eth,net,web3,personal,web3" console
```

Обратите внимание, что мы указали здесь те же значения параметров datadir, networked, rpc и rpcapi, что и на предыдущем уроке.

Для подключения к запущенному таким способом узлу вы можете использовать ту же самую команду, что и раньше:

```
$ geth -datadir node1 -networkid 98760 attach ipc://home/book/node1/geth.ipc
```

Запишите команду подключения в файл attach_node.sh для удобства (листинг 3.2.).

Листинг 3.2. Файл attach_node.sh

```
geth -datadir node1 -networkid 98760 attach ipc://home/book/node1/geth.ipc
```

Проверка учетной записи и баланса

Запустите файл attach_node.sh и в приглашении консоли Geth введите команду accounts:

```
> web3.eth.accounts
["0xd902f8405a6108e8bd9343d1bfccf21a081d
```

Оказывается, в нашем тестовом узле уже определен пользователь, хотя мы не создавали его явным образом и не задавали пароль, как на прошлом уроке. У этого пользователя пустой пароль.

Попробуйте также ввести команду web3.version:

```
> web3.version
{
 api: "0.20.1",
 ethereum: "0x3f",
 network: "98760",
  node: "Geth/v1.8.9-unstable/linux-arm/
go1.9.6",
 whisper: "6.0",
 getEthereum: function(callback),
 getNetwork: function(callback),
 getNode: function(callback),
```

```
 getWhisper: function(callback)
}
```

Здесь будут показаны те же самые данные приватной сети, что и на предыдущем уроке.

Проверьте текущий баланс при помощи следующей команды:

```
>
web3.fromWei( eth.getBalance(eth.coinbase)
1.157920892373161954235709850086879078532
+59
```

Как видите, на нашем аккаунте есть уже довольно много монет Ether, хотя мы и не запускали майнинг явным образом. Это все благодаря тестовому режиму geth, выбранному с помощью параметра –dev.

Итоги урока

На третьем уроке вы научились устанавливать узел приватного блокчейна Ethereum на микрокомпьютер Raspberry Pi 3, обладающего оперативной памятью размеров всего 1 Гбайт.

Хотя на этом компьютере не удается запустить майнинг утилитой Geth, вы использовали параметр –dev для создания тестового узла. На таком узле майнинг запускать не нужно, т.к. там уже предусмотрен аккаунт с очень большим балансом.

Урок 4. Учетные записи и перевод средств между аккаунтами

Цель урока: научиться просматривать список аккаунтов, создавать новые аккаунты, познакомиться с криптовалютными единицами в сети Ethereum, а также научиться переводить средства с одного аккаунта на другой с помощью транзакции из консоли Geth.

Практические задания: нужно просмотреть список аккаунтов, добавить новый аккаунт и перевести на него средства с другого аккаунта. Далее нужно посмотреть состояние транзакции и получить ее квитанцию.

К четвертому уроку мы подготовили серверы Ubuntu и Debian и установили на них Go Ethereum (программу Geth), создали узел приватной сети. Мы проделали аналогичную операцию с микрокомпьютером Raspberry Pi 3, установив на него ОС Rasberian и Geth для запуска сети в тестовом режиме с параметром –dev.

Мы научились подключаться к узлу сети в отдельном окне консоли, получив приглашение Geth в виде символа “>” (далее мы будем назвать это приглашение консолью Geth). Напомним, что для выхода из консоли Geth и возврата к системному приглашению ОС нужно ввести команду exit.

Детальное описание программного интерфейса фрейм-

ворка Web3 JavaScript API версии 0.2x.x, вы найдете здесь: https://github.com/ethereum/wiki/wiki/JavaScript-API.

На момент подготовки книги версия 1.0 еще не реализована, но вы можете изучить документацию на нее по адресу http://web3js.readthedocs.io/en/1.0/. В нашем руководстве мы будем приводить примеры использования и стабильной версии 0.2x.x, и бета-версий 1.0-beta.xx.

Просмотр и добавление аккаунтов

В сети Ethereum, как тестовой, так и «настоящей», аккаунты играют очень важную роль. Именно аккаунтам принадлежат криптовалютные средства, необходимые для выполнения транзакций. Все взаимодействия в сети Ethereum происходят в виде транзакций между аккаунтами.

Заметим, что помимо аккаунтов пользователей, в сети Ethereum предусмотрены аккаунты смарт-контрактов, загруженных в блокчейн, но об этом мы будем говорить позже.

В процессе инициализации приватной узла сети Ethereum на втором уроке мы добавляли первую учетную запись (аккаунт) явным образом, задавая для нее пароль:

```
$ geth –datadir node1 account new
```

На третьем уроке мы запускали тестовую приватную сеть, задавая geth параметр –dev. При этом на узле уже была создана одна учетная запись, к тому же с большим положительным балансом.

Когда вы добавляете аккаунт, создаются приватный и публичный ключ. Приватный ключ хранится на диске вашего сервера, публичный используется для идентификации аккаунта другими пользователями.

Просмотр списка аккаунтов

Откройте две консоли и подключитесь в каждой из них пользователем book. Далее в первой консоли запустите узел следующей командой:

```
$ sh start_node.sh
```

Во второй консоли подключитесь к запущенному узлу:

```
$ sh attach_node.sh
```

Скрипты start_node.sh и attach_node.sh мы подготовили ранее на втором уроке нашего курса.

Теперь, чтобы посмотреть список аккаунтов, достаточно выдать такую команду в консоли второй Web3:

```
> web3.eth.accounts
["0x4f744742ac711fd111c7a983176db1d48d29
```

Если аккаунтов несколько, они будут показаны через запятую как элементы массива:

```
> web3.eth.accounts
["0x4f744742ac711fd111c7a983176db1d48d29
```

```
"0xf212d0180b331a88bd3cafbd77bbd0d56398ae0
```

Каждый аккаунт обладает своим идентификатором, или адресом, представляющим собой длинное шестнадцатеричное число. Именно эти адреса и показывает команда web3.eth.accounts.

Добавление аккаунта

Вы можете добавить аккаунты в консоли Geth или в консоли ОС, запустив там Geth с соответствующим параметром.

Давайте добавим новый аккаунт, указав для него пароль test (никогда не используйте такие простые пароли в рабочих проектах):

```
> web3.personal.newAccount("test")
"0x346cc69a63f9b84c45f17e337574c0150ab6b
```

Здесь мы обратились к объекту web3.personal. Чтобы такие обращения были возможны, при запуске узла сети мы указали название этого объекта наряду с другими объектами в параметре rpcapi (см. урок 2). При использовании параметра –dev параметр rpcapi задавать явным образом не требуется.

Если раньше массив аккаунтов содержал два элемента, то теперь в нем будет на один элемент больше:

```
> web3.eth.accounts
["0x4f744742ac711fd111c7a983176db1d48d29
"0xf212d0180b331a88bd3cafbd77bbd0d56398ae0
"0x346cc69a63f9b84c45f17e337574c0150ab6bc0
```

Для каждого аккаунта на узле создается ключ. Другой неотъемлемый параметр аккаунта – это его пароль. Пароль при создании аккаунта никуда не записывается, поэтому его невозможно восстановить.

Теперь для добавления аккаунта воспользуемся командой geth account с параметром new. Откройте новое консольное окно пользователем book и введите там следующую команду:

```
$ geth –datadir node1 account new
```

Будет добавлен новый аккаунт, а его идентификатор (адрес), будет показан в фигурных скобках:

```
INFO [02-17|23:01:28.855] Maximum peer
count                              ETH=25
LES=0 total=25
Your new account is locked with a
password. Please give a password. Do not
forget this password.
Passphrase:
```

```
  Repeat passphrase:
  Address:
{ae7bb3649a5c597d44f812b4a636f3cc21ee98e1}
```

При добавлении аккаунта у вас будет запрошен пароль.

Обратите внимание, что во всех этих командах мы указывали путь к рабочему каталогу нашего приватного блокчейна с помощью параметра datadir.

Теперь если вы вернетесь в консольное окно Geth и посмотрите там список аккаунтов, то увидите, что теперь на вашем узле определено целых четыре аккаунта:

```
  > web3.eth.accounts
  ["0x4f744742ac711fd111c7a983176db1d48d29
"0xf212d0180b331a88bd3cafbd77bbd0d56398ae0
"0x346cc69a63f9b84c45f17e337574c0150ab6bc0
"0xae7bb3649a5c597d44f812b4a636f3cc21ee98e
```

Параметры команды geth account

Запуская команду geth account с параметрами в консоли ОС, вы можете полностью управлять аккаунтами. Это самый безопасный способ, так как в процессе управления пользователями вы не обращаетесь ни к каким фреймворкам, а работаете с Geth напрямую.

Команде geth account можно передавать параметры, пере-

численные в табл. 4.1.

Табл. 4.1. Параметры команды geth account

Параметр	Описание
list	Вывод списка аккаунтов
new	Создание нового аккаунта
update	Обновление существующего аккаунта (в том числе смена пароля)
import	Импорт приватного ключа в новый аккаунт

Хорошее описание управления аккаунтами с примерами можно найти здесь: https://github.com/ethereum/go-ethereum/wiki/Managing-your-accounts.

Пароли аккаунтов

Обращаем ваше внимание, что пароли аккаунтов **невозможно восстановить**. Если вы создаете узел для реального проекта или собираетесь использовать аккаунт для хранения криптовалютных средств, обращайтесь с паролем очень осторожно.

Лучше всего запомнить пароль или записать его на обычной бумаге и хранить в безопасном месте. Доступ к вашему аккаунту обеспечивается приватным ключом и паролем.

Приватный ключ хранится на сервере узла и может быть похищен, например, с использованием какой-нибудь уязвимости в ОС или в ПО, установленном на сервере. Но чтобы использовать похищенный ключ, злоумышленнику потребуется еще и пароль. Мы не рекомендуем хранить пароль в

компьютере (на сервере, ноутбуке и любом другом), так как он может быть похищен вместе с ключом.

Также учтите, что вредоносные программы могут перехватить управление клавиатурой, а также получить доступ к буферу обмена данных Clipboard. В этом случае они смогут похитить пароль в процессе его ввода с клавиатурой. Используйте антивирусные программы для защиты от вирусов и вредоносных программ различных типов.

Если злоумышленник сможет похитить приватный ключ и пароль, он получит полный доступ к вашему аккаунту и сможет, например, перевести все имеющиеся там криптовалютные средства на свой аккаунт.

Криптовалюта в Ethereum

Как мы уже говорили, для того чтобы владелец аккаунта мог проводить транзакции, публиковать и запускать смарт-контракты, на аккаунте должны быть средства. В тестовой сети, которую мы создали на втором уроке, эти средства можно получить при помощи майнинга. Если вы создаете тестовую сеть с помощью Geth, передавая ей параметр –dev, уже будет создан аккаунт, имеющий на балансе значительное количество криптовалюты.

Далее вы можете создавать новые аккаунты и переводить на них средства с тех аккаунтов тестовой сети, где уже имеются монеты.

Однако в основной сети Ethereum майнинг – дорогое и долгое занятие. Есть другая возможность, а именно: приобретение криптовалютных средств у владельцев других аккаунтов на биржах и обменниках. При этом владелец аккаунта может перевести средства на ваш аккаунт. Вы должны будете сообщить адрес вашего аккаунта, и если вы ошибетесь, то не будет никакой возможности вернуть ваши средства.

На этом уроке мы будем выполнять операцию перевода средств (разумеется, тестовых) в нашей приватной сети.

Денежные единицы Ethereum

Внутренняя валюта сети Ethereum называется Ether, или эфир. Когда владелец аккаунта выполняет транзакции, ва-

люта, которая имеется на счету аккаунта, тратится.

Помимо Ether, существуют более крупные и более мелкие единицы криптовалюты Ethereum (аналогично тому, как в фиатных деньгах существуют рубли и копейки, доллары и центы).

Самая мелкая единица – это Wei. В одном эфире (т.е. в одном Ether) содержится целых 1 000 000 000 000 000 000 единиц Wei.

Как вы увидите далее, единица Wei удобнее, чем Ether, когда речь идет об оплате транзакций, не отнимающих много ресурсов.

В таблице 4.2. мы привели полный список денежных единиц Ethereum и их ценность в единицах Wei.

Таблица 4.2. Денежные единицы Ethereum

	Значение, Wei
Wei	1
Kwei, Ada, Femtoether	$1\,000 = 10^3$
Mwei, Babbage, Picoether	$1\,000\,000 = 10^6$
Gwei, Shannon, Nanoether, Nano	$1\,000\,000\,000 = 10^9$
Szabo, Microether,Micro	$1\,000\,000\,000\,000 = 10^{12}$
Finney, Milliether,Milli	$1\,000\,000\,000\,000\,000 = 10^{15}$
Ether	$1\,000\,000\,000\,000\,000\,000 = 10^{18}$
Kether, Grand,Einstein	$1\,000\,000\,000\,000\,000\,000\,000 = 10^{21}$
Mether	$1\,000\,000\,000\,000\,000\,000\,000\,000 = 10^{24}$
Gether	$1\,000\,000\,000\,000\,000\,000\,000\,000\,000 = 10^{27}$
Tether	$1\,000\,000\,000\,000\,000\,000\,000\,000\,000\,000 = 10^{30}$

В интернете можно найти сайты с конвертами криптова-

лют Ethereum, вот один из них: https://etherconverter.online/. Здесь же показывается текущий курс Ether по отношению к доллару и евро.

Определяем текущий баланс наших аккаунтов

Выше мы добавили в нашу приватную сеть несколько аккаунтов. Полный список аккаунтов всегда можно посмотреть в консоли Geth при помощи команды web3.eth.accounts.

С помощью функции web3.eth.getBalance мы можем посмотреть баланс аккаунта в единицах Wei. Адрес аккаунта нужно передать функции в качестве параметра:

```
> web3.eth.getBalance("0x4f744742ac711fd111c
2.3085e+22
```

Функция web3.eth.getBalance возвращает достаточно большое число. Вообще при работе с API фреймворков Ethereum мы часто будем иметь дело с очень большими числами. Когда вы будете разрабатывать свое децентрализованное приложение (DApp), это нужно будет учитывать.

Для того чтобы узнать баланс нужного вам аккаунта в Ether, используйте функцию fromWei:

```
> web3.fromWei(eth.getBalance("0x4f744742ac7
23110
```

При этом функции eth.getBalance нужно передать адрес проверяемого аккаунта.

Перевод средств с одного аккаунта на другой

Если подобным образом проверить баланс для аккаунтов, созданных нами дополнительно, то окажется, что сразу после создания он равен нулю, например:

```
> web3.fromWei( eth.getBalance("0xf212d0180b
0
```

Это неудивительно, ведь мы еще не переводили средства на эти аккаунты. Но можно перевести деньги с основного аккаунта нашей приватной сети, которую мы создали на втором уроке. Для нее был запущен майнинг, поэтому там уже должны быть средства.

Метод eth.sendTransaction

Давайте посмотрим, какие у нас есть аккаунты и какой на них баланс:

```
> web3.eth.accounts
["0x4f744742ac711fd111c7a983176db1d48d29
"0xf212d0180b331a88bd3cafbd77bbd0d56398ae0
```

```
"0x346cc69a63f9b84c45f17e337574c0150ab6bc0
"0xae7bb3649a5c597d44f812b4a636f3cc21ee98e

  >
web3.fromWei( eth.getBalance("0x4f744742ac
  23135
  >
web3.fromWei( eth.getBalance("0xf212d0180b
  0
  >
web3.fromWei( eth.getBalance("0x346cc69a63
  0
  >
web3.fromWei( eth.getBalance("0xae7bb3649a
  0
```

Как видите, на первом из этих аккаунтов средства есть, а на остальных – ничего нет.

Давайте переведем часть средств, а именно 0.05 Ether, с первого их этих аккаунтов на другой, где средств нет. Воспользуемся для этого методом eth.sendTransaction:

```
  >
eth.sendTransaction({from:"0x4f744742ac711
to:"0xf212d0180b331a88bd3cafbd77bbd0d56398
value: web3.toWei(0.05, "ether")})
```

При попытке выполнить эту операцию вы, однако, получите сообщение об ошибке:

```
  Error: authentication needed: password or
unlock
     at web3.js:3143:20
     at web3.js:6347:15
     at web3.js:5081:36
     at <anonymous>:1:1
```

Перед выполнением такой операции необходимо разблокировать исходный аккаунт, с которого отправляются средства. Для разблокировки введите такую команду:

```
  >
web3.personal.unlockAccount("0x4f744742ac7
"*********")
  true
```

Вместо звездочек укажите пароль, с которым данный аккаунт создавался. Если аккаунт и пароль были указаны правильно, на консоли вы увидите true.

Теперь повторите вызов функции eth.sendTransaction:

```
  >
```

```
eth.sendTransaction({from:"0x4f744742ac711
to:"0xf212d0180b331a88bd3cafbd77bbd0d56398
value: web3.toWei(0.05, "ether")})
  "0xb6d13a5e915c3af1feabad7caec7b45348146
```

На этот раз функция выполнится успешно и вернет нам так называемый хеш транзакции (Transaction Hash) со значением:

```
0xb6d13a5e915c3af1feabad7caec7b453481466
```

Чтобы перевод средств произошел успешно, эта транзакция должна быть выполнена. Кроме того, нужно дождаться, когда в вашу тестовую сеть будет добавлен новый блок. После этого средства появятся на целевом счету.

Посмотрите на консоль, где мы запустили наш узел. Видим, что там есть сообщение о запуске транзакции с указанным выше хешем, а также о том, что был добавлен новый блок:

```
  INFO    [02-17|23:20:50.917]    Submitted
tcaecs7ac45348146695973b32285df287639717e916
recipient=0xF212D0180B331a88BD3CafbD77bBd0
  INFO   [02-17|23:20:53.018]  Commit  new
mining work                      number=4643
sealhash=0f860c…d73ae1   uncles=0   txs=1
```

```
gas=21000 fees=2.1e-05 elapsed=36.186ms
  INFO [02-17|23:22:10.119] Successfully
sealed new block                number=4643
sealhash=0f860c…d73ae1 hash=3c9761…8b0eea
elapsed=1m17.116s
  INFO [02-17|23:22:10.119] 🔗 block
reached canonical chain         number=4636
hash=3b5237…0e8761
  INFO [02-17|23:22:10.119] 🔨 mined
potential block                 number=4643
hash=3c9761…8b0eea
```

Теперь, если проверить баланс второго аккаунта, на который мы переводили средства, то окажется, что он как раз равен 0.05 Ether, как и должно быть:

```
  >
web3.fromWei( eth.getBalance("0xf212d0180b
  0.05
```

Функции eth.sendTransaction мы передали в параметрах from и to адреса исходного и целевого аккаунта соответственно. В параметре value мы задали количество переводимых средств в единицах Wei. Так как нам удобнее указывать размер средств в более крупных единицах Ether, для пере-

вода в Wei мы воспользовались функцией web3.toWei.

Заметим, что в реальной сети вместе с транзакцией нам нужно указать средства, которые пойдут на обработку этой транзакции (так называемый газ). Для этого команде eth.sendTransaction нужно задать параметры gas (количество газа, заданное для выполнения операции) и gasPrice (стоимость газа в Wei):

```
  eth.sendTransaction({from:"0x208970e5e3d
to:"0x82a4165f21d8f1867d536e81537fc0085e54
web3.toWei(5,   "ether"),   gas:   120000,
gasPrice: 80000000000})
```

Просмотр состояния транзакции

Зная хеш транзакции, вы можете получить о ней определенную информацию. Это можно сделать при помощи метода web3.eth.getTransaction, передав ему в качестве параметра строку хеша транзакции.

В ответ вы получите объект следующего вида:

```
  >
web3.eth.getTransaction("0xb6d13a5e915c3af
  {
                                  blockHash:
"0x3c9761fefa52a0bc563733d87163828c5fe1316
```

```
   blockNumber: 4643,
                                                from:
"0x4f744742ac711fd111c7a983176db1d48d29f41
   gas: 90000,
   gasPrice: 1000000000,
                                                hash:
"0xb6d13a5e915c3af1feabad7caec7b4534814669
   input: "0x",
   nonce: 0,
                                                  r:
"0x1e3519fbca45cc5f6a0804232c8f0362d42c8ab
                                                  s:
"0x69e617eceec461b727a0997fd837264e02242fa
                                                to:
"0xf212d0180b331a88bd3cafbd77bbd0d56398ae0
   transactionIndex: 0,
   v: "0xfc2",
   value: 50000000000000000
  }
```

Поля объекта состояния транзакции перечислены в табл. 4.3.

Таблица 4.3. Состояние транзакции

Поле	Тип данных	Описание
hash	String, 32 байта	Хеш транзакции
nonce	Number	Количество транзакций, совершенных отправителем до этой транзакции
blockHash	String, 32 байта	Хеш блока, в котором размещена транзакция, или значение null, если транзакция все еще находится в состоянии ожидания
blockNumber	Number	Номер блока, в котором размещена транзакция, или значение null, если транзакция все еще находится в состоянии ожидания
transactionIndex	Number	Целочисленное значение индекса позиции транзакции в блоке или значение null, если транзакция все еще находится в состоянии ожидания
from	String, 20 байт	Адрес отправителя
to	String, 20 байт	Адрес получателя или null, если это транзакция создания смарт-контракта
value	BigNumber	Количество Wei, переведенных с одного аккаунта на другой в рамках данной транзакции
gasPrice	BigNumber	Стоимость газа в единицах Wei, заданная отправителем
gas	Number	Количество газа, заданное отправителем для выполнения данной транзакции
input	String	Данные, передаваемые вместе с транзакцией
v, r, s	String, 32 байта	Значения для подписи транзакции

Как видите, здесь есть адреса отправителя **from** и получателя **to**, а также объем переведенных средств в Wei.

Когда вы запускаете транзакцию на выполнение, она выполняется не сразу. Вначале транзакция находится в состоянии ожидания, пока майнеры не создадут для нее новый блок. Анализируя поля номера блока **blockNumber**, в котором размещается транзакция, можно определить, была ли запущена транзакция или еще нет.

Поля v, r, s содержат значения для подписи транзакции. Их содержимое можно использовать для получения публичного ключа аккаунта. Способ их использования с этой целью обсуждается здесь: https://ethereum.stackexchange.com/

questions/13778/get-public-key-of-any-ethereum-account.

Содержимое полей **gasPrice** и **gas** имеет отношение к стоимости транзакции. Пока мы работаем с тестовой сетью, об этом можно не беспокоиться. Если кратко, то **gasPrice** содержит так называемую стоимость газа, который тратится на выполнение транзакции, а поле **gas** – количество газа, которое выделил отправитель для выполнения транзакции.

Вы можете думать о газе, как о бензине, который тратится на выполнение транзакций. Стоимость этого бензина может меняться, а на разные транзакции этого бензина тратится разное количество.

Квитанция транзакции

Когда транзакция выполнилась (о чем можно узнать при помощи только что рассмотренного метода web3.eth.getTransaction), вы можете получить так называемую квитанцию об ее выполнении. Для этого предназначен метод web3.eth.getTransactionReceipt.

Передайте этому методу в качестве параметра хеш завершенной транзакции, и вы получите квитанцию в виде объекта:

```
> web3.eth.getTransactionReceipt("0xb6d13a5e
{
```

```
                                        blockHash:
"0x3c9761fefa52a0bc563733d87163828c5fe1316
   blockNumber: 4643,
   contractAddress: null,
   cumulativeGasUsed: 21000,
                                          from:
"0x4f744742ac711fd111c7a983176db1d48d29f41
   gasUsed: 21000,
   logs: [],
                                       logsBloom:
"0x00000000000000000000000000000000000000000
                                          root:
"0xa5cdcd909d837e937189f4cffc52840111f4430
                                            to:
"0xf212d0180b331a88bd3cafbd77bbd0d56398ae0
                              transactionHash:
"0xb6d13a5e915c3af1feabad7caec7b4534814669
   transactionIndex: 0
  }
```

Поля объекта квитанции перечислены в табл. 4.4.

Таблица 4.4. Квитанция выполненной транзакции

Поле	Тип данных	Описание
blockHash	String, 32 байта	Хеш блока, в котором размещена транзакция
blockNumber	Number	Номер блока, в котором размещена транзакция
transactionHash	String, 32 байта	Хеш транзакции
transactionIndex	Number	Целочисленное значение индекса позиции транзакции в блоке
from	String, 20 байт	Адрес отправителя
to	String, 20 байт	Адрес получателя или null, если это транзакция создания смарт-контракта
cumulativeGasUsed	Number	Общее количество газа, использованное при выполнении транзакции в блоке
gasUsed	Number	Количество газа, использованное только этой конкретной транзакцией
contractAddress	String - 20 байт	Адрес созданного смарт-контракта, если транзакция использовалась для создания смарт-контракта, или значение null
logs	Array	Массив объектов журнала, созданного данной транзакцией
logsBloom	String, 256 байт	Так называемый цветной фильтр, используется для экономии места при хранении журналов, а также для поиска журнала нужного смарт-контракта
status	String	Если в этом поле находится значение "0x1", это означает, что транзакция выполнена успешно. При ошибке здесь находится значение "0x0"
root	String, 32 байта	Хеш корня дерева состояний

Анализируя квитанцию, вы можете узнать номер блока, в котором размещена транзакция **blockNumber**, а также индекс позиции размещения транзакции в блоке **transactionIndex**.

При помощи полей **cumulativeGasUsed** и **gasUsed** вы сможете понять, сколько газа ушло на выполнение транзакции. При запуске транзакции вы можете выделить определенное количество газа, но если газа будет больше, израсходуется только часть ваших средств. В реальной сети Ethereum, если газа будет выделено слишком мало, транзак-

ция будет очень долго находиться в состоянии ожидания или не будет выполнена вовсе.

Также обратите внимание на поле **contractAddress**. Когда мы займемся публикацией смарт-контрактов, нам потребуется адрес размещенного смарт-контракта из этого поля.

Итоги урока

На четвертом уроке вы научились управлять аккаунтами, а также переводить средства с одного аккаунта на другой. Попутно вы изучили криптовалютные единицы, которые применяются в сети Ethereum.

Вы узнали, что созданные транзакции сначала попадают в состояние ожидания и находятся в нем до тех пор, пока в блокчейн не будет добавлен новый блок. Проверку состояния транзакции можно выполнить методом web3.eth.getTransaction.

Когда транзакция выполнена, то с помощью метода web3.eth.getTransactionReceipt вы можете получить ее квитанцию. Из квитанции можно определить, сколько газа было потрачено на выполнение транзакции. Если транзакция была связана с публикацией контракта, то этот метод поможет вам получить адрес опубликованного контракта. Адрес контракта необходим, как вы скоро узнаете, для вызова его методов.

Урок 5. Публикация первого контракта

Цель урока: на этом уроке вы узнаете о смарт-контрактах в сети Ethereum и о том, как они выполняются виртуальной машиной Ethereum. Мы расскажем, как создать смарт-контракт на языке Solidity и как опубликовать его в приватной сети. Также вы научитесь устанавливать и запускать пакетный компилятор solc. Вам нужно узнать о так называемом бинарном интерфейсе приложения Application Binary Interface (ABI) и научится его использовать.

Практические задания: на пятом уроке вы создадите первый контракт HelloSol в среде разработки Remix Solidity IDE. Этот контракт вам предстоит опубликовать в приватной сети и проверить в работе. Также вы установите пакетный компилятор solc и откомпилируете им свой смарт-контракт HelloSol.

Как вы, конечно, знаете, обычные контракты заключаются между двумя (или несколькими сторонами) в бумажном виде. Например, в контракте может быть написано, что одна сторона берет на себя обязательство поставить какой-нибудь товар, а другая обязуется после получения товара выполнить оплату. Либо контракт может предусматривать предоплату, и тогда товар поставляется после получения денег.

В контрактах также обычно предусматривают некие штрафные санкции за невыполнение условий. Это может быть, например, штраф за задержку поставки или оплаты. Если условия контракта нарушаются, пострадавшая сторона может обратиться в суд, при этом в дело включатся юристы. Вся эта процедура может отнять немало времени и средств.

Смарт-контракты в Ethereum

В сети Ethereum имеется возможность создавать так называемые смарт-контракты (Smart Contracts). Смарт-контракт представляет собой программный код, работающий в среде виртуальной машины Ethereum. Он запускается на всех узлах сети, и результаты его работы также реплицируются на все узлы.

С помощью смарт-контрактов очень удобно отслеживать выполнение транзакций, например, имеющих отношение к поставке товаров (особенно электронных, таких как файлы книг или доступы к сервисам), автоматизируя оплату при помощи криптовалютных средств. Если смарт-контракт получил тем или иным способом подтверждение выполнения условий сделки, то он может сам, автоматически, перевести средства поставщику.

Конец ознакомительного фрагмента.