**Краткое описание**

Для связи с блокчейном мы должны использовать **клиент** блокчейна. Клиент — это часть программного обеспечения, способная устанавливать канал связи p2p с другими клиентами, подписывать и транслировать транзакции, осуществлять майнинг, развертывать и взаимодействовать с интеллектуальными контрактами и т. д. Клиент часто называют **узлом** .

Желтая бумага (список конфиурации) определяет требуемые функции узлов в сети, алгоритм майнинга, параметры [ECDSA с](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm&xid=17259,15700023,15700186,15700191,15700259,15700271,15700302&usg=ALkJrhiJ_k1yHpMxSKVOk_XlUqO6CZT9Gw) закрытым / открытым ключом. Он определяет все функции, которые делают узлы полностью совместимыми с клиентами Ethereum.

Основываясь на желтой бумаге, каждый может создать собственную реализацию узла Ethereum на любом языке, который он считает нужным.

На сегодняшний день самыми популярными клиентами являются [Geth](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://github.com/ethereum/go-ethereum/wiki&xid=17259,15700023,15700186,15700191,15700259,15700271,15700302&usg=ALkJrhgdfup28YfMfNceMKtC_dBwTORwbA) и [Parity](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.ru&sl=en&sp=nmt4&tl=ru&u=https://wiki.parity.io/&xid=17259,15700023,15700186,15700191,15700259,15700271,15700302&usg=ALkJrhj5LRM7YmgTq7JcqMmlqX_odmyThw) . Реализации различаются в основном по выбору языка программирования — где Geth использует Golang, а Parity использует Rust.

Поскольку Geth является самой популярной клиентской реализацией, доступной на данный момент, мы сосредоточимся на ней сейчас.

В общем виде, мы можем разделить программное обеспечение узлов на два типа: полные узлы и легкие (весовые) узлы. Полные узлы проверяют блок, который транслируется в сеть. Таким образом, они гарантируют, что транзакции, содержащиеся в блоках (и сами блоки ), следуют правилам, определенным в спецификациях Ethereum. Они поддерживают текущее состояние сети (как определено в соответствии со спецификациями Ethereum).

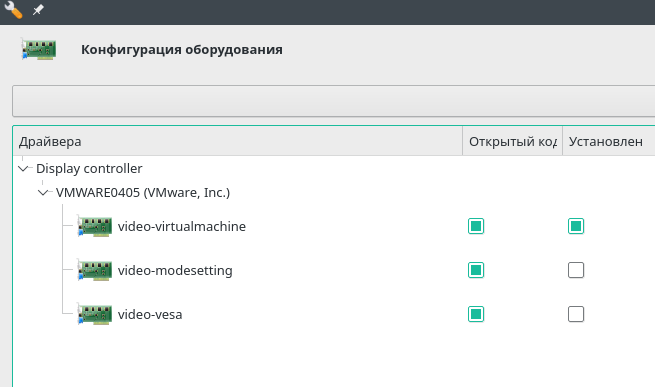
Транзакции и блоки, которые не соответствуют правилам, не используются для определения текущего состояния сети Ethereum. Например, если A пытается отправить 100 эфиров в B, но A имеет 0 эфиров, и блок включает эту транзакцию, полные узлы поймут, что это не соответствует правилам Ethereum, и отклонят этот блок как недействительный. В частности, выполнение смарт-контрактов является примером транзакции. Всякий раз, когда в транзакции используется умный контракт (например, отправка токенов ERC-20), все полные узлы должны будут выполнить все инструкции, чтобы убедиться, что они достигли правильного, согласованного следующего состояния блокчейна.

Есть несколько способов достичь одного и того же состояния. Например, если бы А имел 101 эфир и отдал сто из них Б за одну транзакцию, заплатив 1 эфир за газ, конечный результат был бы таким же, как если бы А отправил 100 транзакций по 1 эфиру каждая, заплатив 0,01 эфира за транзакцию (игнорируя, кто получил комиссию за транзакцию). Чтобы узнать, разрешено ли теперь B посылать 100 эфира, достаточно знать, каков текущий баланс B. Полные узлы, которые сохраняют всю историю транзакций, называются полными узлами архивации. Они должны существовать в сети, чтобы быть здоровыми.

Узлы также могут отказаться от старых данных; если B хочет отправить 100 эфира в C, не имеет значения, как был получен эфир, только учетная запись B содержит 100 эфира. Легкие узлы, напротив, не проверяют каждый блок или транзакцию и могут не иметь копии текущего состояния блокчейна. Они полагаются на полные узлы, чтобы предоставить им недостающие детали (или просто не хватает определенной функциональности). Преимущество легких узлов заключается в том, что они могут гораздо быстрее запускаться и работать, могут работать на устройствах с большим количеством вычислительных ресурсов / памяти и не поглощают почти столько же памяти. С другой стороны, существует элемент доверия в других узлах (он варьируется в зависимости от клиента, и вероятностные методы / эвристика могут использоваться для снижения риска). Некоторые полные клиенты включают функции для более быстрой синхронизации (например, синхронизация деформации Parity).

**Работа с GETH**

Начиная работать с geth учтите, что большинство библиотек, которые будут установлены в вашу систему или требуются для работы (за исключением python-библиотек), требуют последней стабильной версии geth. Ubuntu, Mint, Debian, OpenSuse, Fedora, CentOS требуют дополнительной установки и манипуляций при установке. Принцип установки описан в приложении 1. Текущая инструкция покажет, как установить и запустить простейший смарт-контракт. Будут разобраны решения наиболее часто встречающихся проблем при работе и установке. Выбранная ОС – Manjaro KDE 20 Версия ядра 5.9.10.

1. Войдите в систему в терминале. Обновите все существующие пакеты. Команда: sudo pacman –Suy.
2. Во время обновления НЕ устанавливайте множество драйверов для вашей видеокарты, если для майнинга планируется использовать ее. Достаточно установки проприетарного драйвера, например, nvidia-440 или nvidia-450. Если производитель видеокатры – АМД, сделать аналогичную операцию, однако пример привести невозможно ввиду конфигурации оборудования.
   1. Если используется виртуальная машина, можно использовать драйвер vesa, как на скриншоте
3. Установите geth. Команда **sudo pacman –S geth.** В репозитории Manjaro находится последняя стабильная версия geth. Для контроля введите команду geth version. Сравните номер полученной версии и версии на сайте <https://geth.ethereum.org/>. Версии должны совпадать. Все версии ниже 1.9.20 – гарантированно не работают с python-библиотеками, библиотеками golang.
4. В корневом каталоге создайте файл **genesis.json**
   1. Содержание файла genesis.json должно включать следующее:

**---------------------------------genesis.json----------------------------------------**

{

"config": {

"chainId": 4777,

"homesteadBlock": 0,

"eip150Block": 0,

"eip155Block": 0,

"eip158Block": 0

},

"difficulty": "20",

"gasLimit": "2100000",

"alloc": {

"Ваш номер кошелька 1": { "balance":"300000" },

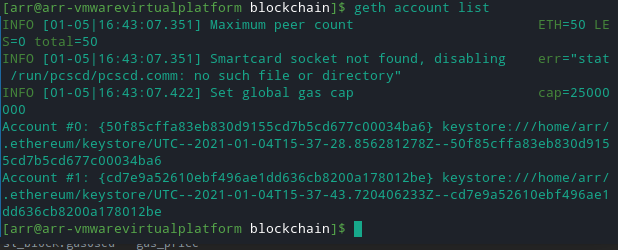
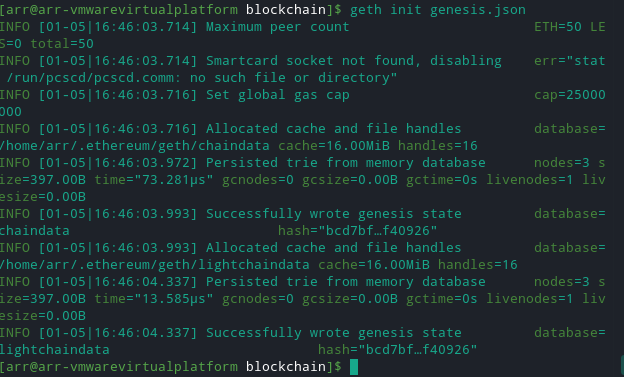
"Ваш номер кошелька 2":{ "balance":"400000"}

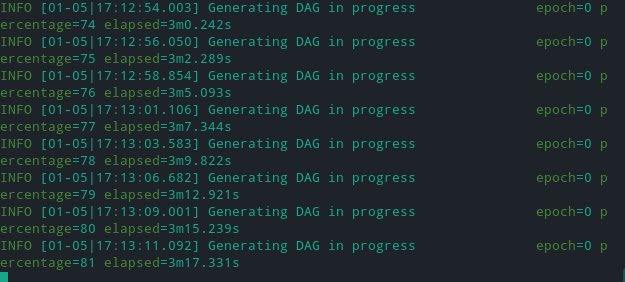
}

}

**--------------------------------------------------------------------------------------**

Значения внутри **config НЕ** должны быть изменены, но могут быть добавлены необходимые параметры. В случае изменения **config**, в пункте 5 вы увидите ошибку «Fatal, permission denied». Ее будет невозможно решить правами доступа в файл.

1. Создайте аккаунты. Для этого удалите старую базу данных, если она имеется. Команда: **rm -rf data/ - удалит ВСЕ данные geth,**
2. Удалите базу данных, команда: **geth removedb.**
3. Создайте необходимое количество аккаунтов, команда для создания одного аккунта: **geth account new.**
4. Проверьте созданные аккаунты, команда geth account list. Пример вывода для 2 аккаунтов :
5. Откройте терминал в основном каталоге, введите команду: geth init genesis.json. Пример корректного вывода: 
6. Попробуйте начать майнинг, для этого используйте команду**:**

***geth --mine --miner.threads=1 --nodiscover --verbosity=4 --unlock=0x"Аккаунт майнера" --password=password ----etherbase=0x"Адрес начисления вознаграждения".*** *В случае успеха вы увидите сообщения вида*.

***Возможные ошибки***

1. ***Fata,l Error password not recognized –*** *проверьте файл password. Пароль, содержащийся там должен соответствовать паролю аккаунта.*
2. ***Fatal, Error Password not contains in password –*** *ошибка связана с версией golang(<1.10) и geth(<1.9.20). Обновите библиотеки и повторите команду*
3. ***Fatal, no address found –*** *такого аккаунта не существует, введите geth account list и скопируйте аккаунты в 16-ричном виде заново (возможно из-за отсутствия обозначения 0х перед 16-ричным числом). Ошибка также возникает при несоответсвии системных переменных стандартным. Geth может прописать себя при майнинге по любому адресу, но читает только по стандартным путям PATH. В случае несовпадения с маской стандартной переменной, процесс будет удален(kill) с этой ошибкой. Пример получения такой ошибки – копирование с промежуточным хранением (Кодировка при помещении в вспомогательное хранилище может быть изменена. В Manjaro, Ubuntu, Mint типичное изменение кодировки происходит в пределах UTF-8-UTF-16. Такое изменение незначительно, однако изменение на win1251, например, (достаточно даже изменения символа пробела и окончания строки) приводят к текущей ошибке (подобная ошибка возможна при работе с виртуальной операционной системой при изменении конфигурационных файлов в блокноте windows).*

***Работа c несколькими node***

*Работа с несколькими нодами малоотличима от работы с одной нодой при обращении через терминал. Работа при использовании смарт-контрактов отличается. Для работы с конкретной нодой, в окне терминала в основную команду необходимо добавить ключевое слово* **--identity="Node№".**  Общий вид команды при работе с несколькими нодами будет иметь следующий вид: ***geth --mine –identity=”NodeNumber” --miner.threads=1 --nodiscover --verbosity=4 --unlock=0x"Аккаунт майнера" --password=password --etherbase=0x"Адрес начисления вознаграждения".***

*Основные команданты обозначают:*

|  |  |
| --- | --- |
| ***--mine*** | Начать майнинг |
| ***--identity=”NodeNumber”*** | Использовать ноду с именем «NodeNumber» |
| ***--miner.threads=1*** | Определить число потоков майнинга.(рекомендуется не ставить более числа ядер поцессора) |
| ***--nodiscover --verbosity=4*** | Сложность счета блока и определение возможности создания новых блоков автоматически. |
| ***--unlock=0x"Аккаунт майнера"*** | Разблокировать аккаунт майнера для работы |
| ***--password=password*** | Указание пути к файлу с паролем к аккаунту майнера |
| ***--etherbase=0x"Адрес начисления вознаграждения".*** | Указание адреса начисления вознаграждения |

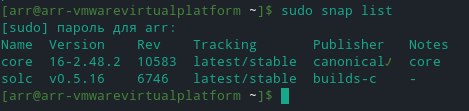
***Работа со смарт-контрактами solc***

*Для работы со смарт-контрактами может потребоваться solc – компилятор языка смарт-контрактов solidity.*

*Для установки последней версии в Manjaro linux следуйте следующей инструкции:*

* *Установите snapd:* ***sudo pacman -S snapd***
* *Запустите Демона snapd:* ***sudo systemctl enable --now snapd.socket***
* *Включите доступ к классическим snap-пакетам****: sudo ln -s /var/lib/snapd/snap /snap***
* *Установите Solc****: sudo snap install solc***
* *Проверьте установку пакета****: sudo snap list***

*Примерный вывод команды* ***sudo snap list*** *показан на скриншоте*



***Приложение 1***

***Сборка geth из исходников***

С установленным golang и наличии переменной GOPATH вы можете добавить рабочее пространство командой:

**go get -d github.com/ethereum/go-ethereum**

Вы можете также получить конкретную версию продукта командой:

**go get -d github.com/ethereum/go-ethereum@v1.9.21**

Имея собранный пакет geth в репозитории, вы можете установить его командой:

**go install github.com/ethereum/go-ethereum/cmd/geth**

Если возникают ошибки вида go: **cannot use path@version** **syntax in GOPATH mode** или похожие ошибки – включите модули go командой: **export GO111MODULE=on.**