

Створення смарт-контрактів Ethereum

Смарт-контрактом називають програму, яка зберігається в блокчейні. Смарт-контракти дають змогу реалізувати в коді можливості, які блокчейн надає для даних. У таких контрактах представлена певна угода між сторонами. Ця угода виражається у вигляді коду, який виконується під час здійснення певної дії та надає відповідь.

Усі умови цих контрактів визначаються програмним способом. У цьому визначенні укладені всі правила, вимоги та винагороди для учасників блокчейну. Також у ньому вказується спосіб передачі цифрових ресурсів між сторонами. Кожному смарт-контракту призначається 20-байтова адреса, яка слугує його унікальним ідентифікатором.

Смарт-контракти працюють автоматично, відправляючи події для активації переходів стану і викликаючи функції. Вони ідеально поєднуються з технологією блокчейна, оскільки дають змогу забезпечити безпечну співпрацю між незнайомими людьми без посередників.

Смарт-контракти найчастіше використовують у поєднанні з Ethereum. Платформа Ethereum - це перший у світі програмований блокчейн. Вона дає змогу визначати смарт-контракти для спрощення передачі цифрових активів, як, наприклад, Ether.

Для написання контрактів використовується мова програмування Solidity. Solidity є повною за Тьюрингом мовою, а отже, дає змогу створювати складні контракти у форматі чітко визначеного коду.

Оскільки всі переходи станів реєструються і зберігаються в незмінному вигляді, необхідно ретельно протестувати контракт перед його відправленням у робоче середовище. Виправлення помилок можуть виявитися дорогими або навіть внести критичні порушення в роботу системи.

Смарт-контракти мають такі важливі властивості та переваги:

Прозорість. Користувачі блокчейна можуть читати і використовувати смарт-контракти через інтерфейси API.

Незмінність. Під час виконання смарт-контракту створюються журнали, які не можна змінити.

Розподіл. Вихідні дані контракту перевіряються й оцінюються багатьма вузлами в мережі. Відомості про стан контрактів загальнодоступні. У деяких випадках навіть "закриті" змінні доступні для перегляду.

Варіанти використання

Смарт-контракти можна використовувати в безлічі галузей і процесів. Давайте розглянемо кілька варіантів використання.

Страхування. У разі виникнення певних подій смарт-контракти активують заявку автоматично, спрощуючи і прискорюючи процес роботи із заявками. У блокчейні можна зберегти відомості про вимоги, які дадуть змогу визначити розмір компенсації застрахованій особі. Така можливість прискорить обробку і знизить вплив людських помилок.

Голосування. Смарт-контракти допоможуть зробити голосування автоматичним і прозорим. Кожен контракт виконує роль одного бюлетеня, що однозначно ідентифікує виборця. Оскільки блокчейн є незмінним, то й голоси неможливо підробити.

Ланцюжки поставок. Під час переміщення товару ланцюжком поставок смарт-контракти можуть реєструвати права власності та підтверджувати відповідальність за товар тієї чи іншої особи в будь-який конкретний момент часу. На будь-якому етапі процесу смарт-контракт однозначно визначає, де мають перебувати продукти. Якщо будь-яка зі сторін у ланцюжку постачань порушить терміни доставки, всі інші сторони знатимуть, де виникла проблема.

Зберігання записів. Багато галузей можуть застосовувати смарт-контракти для підвищення швидкості та надійності зберігання записів. Технологія блокчейна допоможе оцифрувати архіви, а також забезпечити їхнє безпечне шифрування і зберігання. Крім того, можна регулювати доступ до архівів, дозволяючи його тільки підтвердженим користувачам.

Права власності. У смарт-контрактах можна зберігати дані про те, кому належать права власності. Це швидкий і економічний спосіб збереження прав володіння. Крім того, смарт-контракти забезпечують швидку і безпечну передачу прав власності.

Засоби для роботи зі смарт-контрактами

Багато засобів допоможуть вам ефективно створювати смарт-контракти. У наступних розділах перераховано рекомендовані до вивчення засоби, зокрема інтегровані середовища розробки (IDE), розширення та платформи.

Інтегровані середовища розробки

Visual Studio Code. Редактор коду, який перевизначено й оптимізовано для створення та налагодження сучасних хмарних додатків і веб-додатків. У цьому модулі ми використовуємо для всіх вправ саме Visual Studio Code.

Remix. Браузерний компілятор та інтегроване середовище розробки, які дають змогу створювати контракти Ethereum мовою Solidity, а також налагоджувати транзакції. Remix дає чудову можливість вивчити приклади контрактів. Крім того, ви можете створити, протестувати і розгорнути власні контракти. Ми не використовуємо Remix у цьому модулі, але ви можете самостійно вивчити його на прикладах з контрактів.

Модулі

Розширення Truffle for VS Code. Це розширення дає змогу спростити створення, складання та розгортання смарт-контрактів у реєстрах Ethereum. Воно забезпечує вбудовану інтеграцію з Truffle, Ganache та іншими інструментами і службами. У цьому модулі ми будемо використовувати це розширення, щоб створювати і тестувати смарт-контракти.

Платформи

Пакет Truffle. Використовуйте пакет засобів Truffle для тестування контрактів Ethereum перед їх розгортанням у загальнодоступних реєстрах для роботи з реальними грошима. Щоб спростити розробку, її можна виконувати локально. До цього набору засобів входять Truffle, Ganache і Drizzle.

OpenZeppelin. Використовуйте засоби OpenZeppelin для створення, розгортання та експлуатації децентралізованих додатків. OpenZeppelin включає два продукти: бібліотеку контрактів і пакет SDK.

Truffle - это самая популярная платформа среды разработки и тестирования для Ethereum. Его можно установить с помощью диспетчера пакетов Node (npm).

Информация о Truffle

Truffle предоставляет следующие возможности:

- создание, компиляция, развертывание и тестирование смарт-контрактов;
- управление сетью для развертывания в общедоступных и частных сетях;
- управление пакетами для зависимостей проектов;
- интерактивная консоль для прямого взаимодействия с контрактом и управления им;
- настраиваемый конвейер сборки для автоматического запуска проверок и настройки проектов.

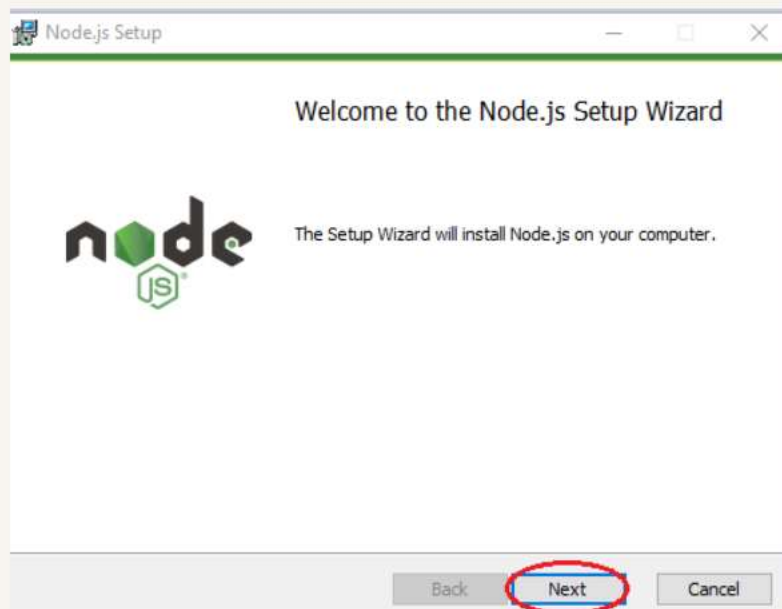
Установка Truffle

Шаг 1. Установите NodeJS версии 16

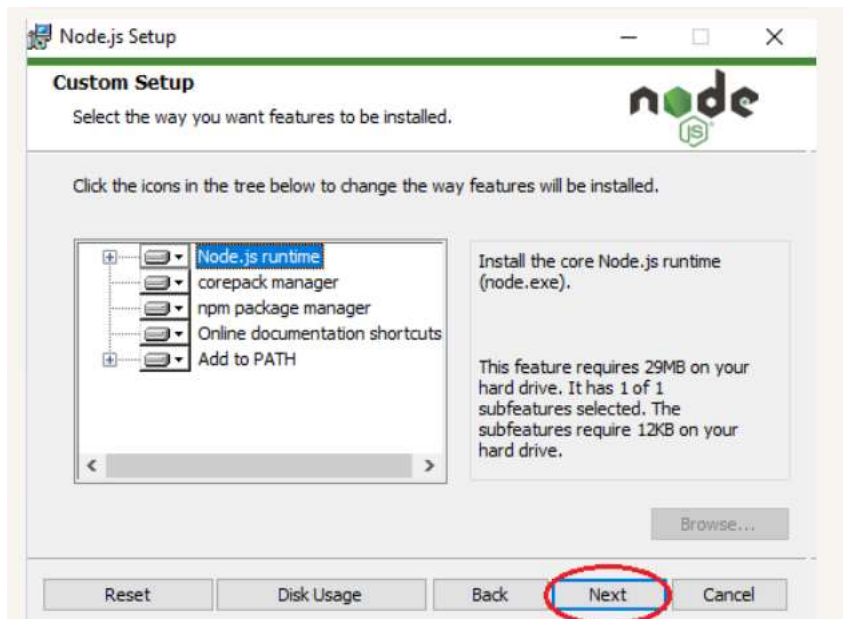
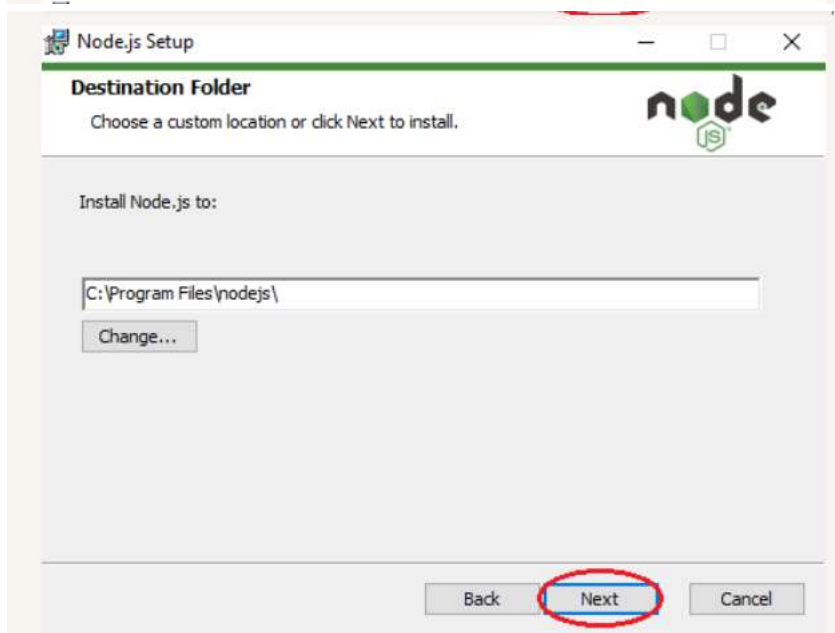
Установите NodeJS с установочным MSI-файлом Windows с NodeJS.org:

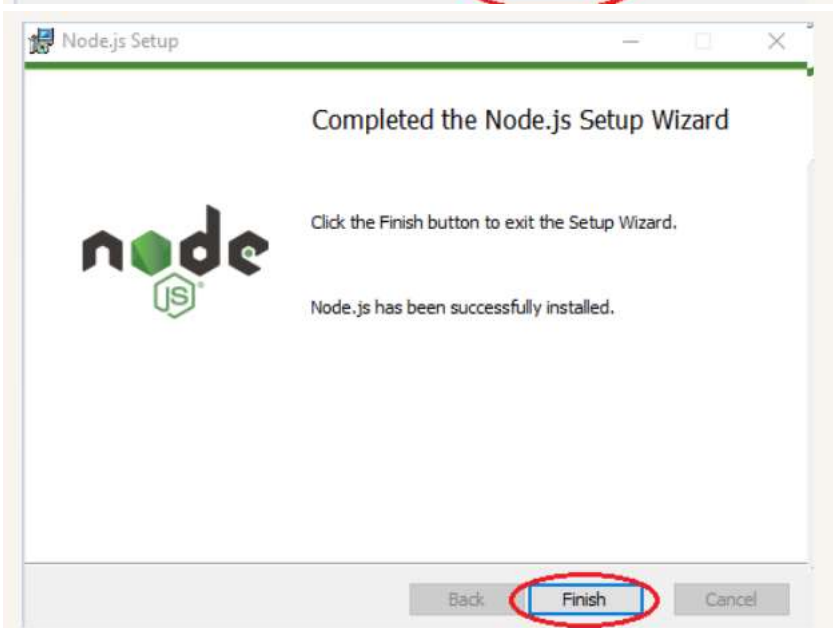
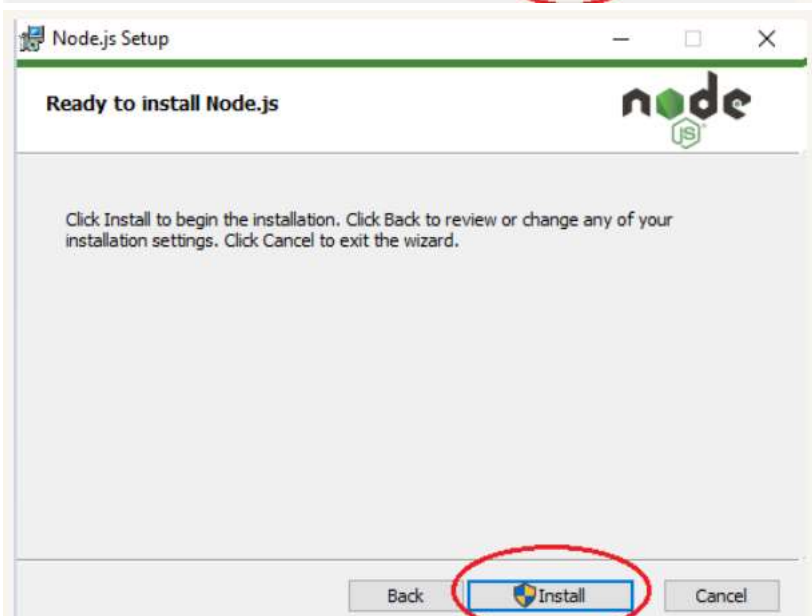
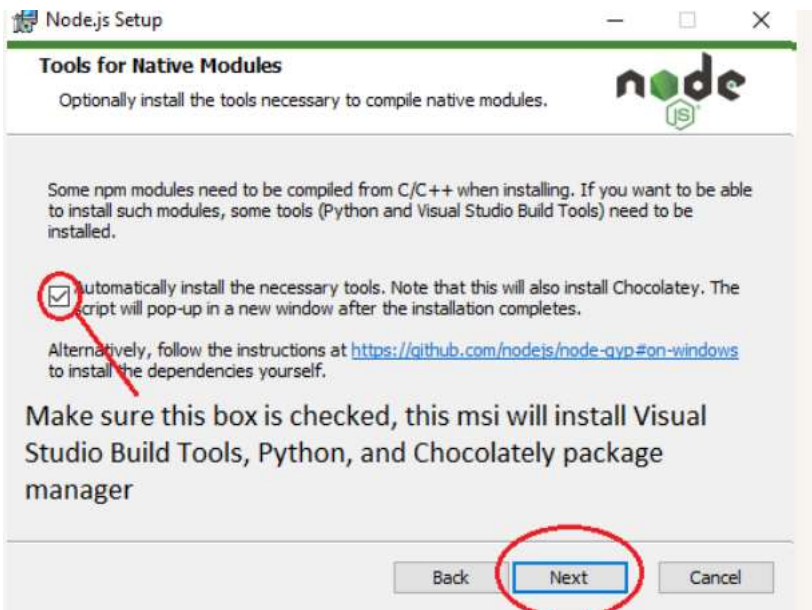
<https://nodejs.org/dist/v16.15.0/node-v16.15.0-x64.msi>

Установщик .msi установит Node.js, NPM, инструменты сборки Python, инструменты сборки Visual Studio C++ и добавит NPM в PATH вашей системы.



Установочный файл - <https://nodejs.org/dist/v16.15.0/node-v16.15.0-x64.msi>





Install Additional Tools for Node.js

Tools for Node.js Native Modules Installation Script

This script will install Python and the Visual Studio Build Tools, necessary to compile Node.js native modules. Note that Chocolatey and required Windows updates will also be installed.

This will require about 3 Gb of free disk space, plus any space necessary to install Windows updates. This will take a while to run.

Please close all open programs for the duration of the installation. If the installation fails, please ensure Windows is fully updated, reboot your computer and try to run this again. This script can be found in the Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these tools manually are available at <https://github.com/nodejs/node-gyp#on-windows>

Press any key to continue . . .

Note: This installer could take up to 3Gb worth of space. The download might also take awhile based on the speed of your ISP

The Windows build tools installer will open in a new cmd line terminal. Press any key to continue installation

Administrator: Windows PowerShell

WARNING: 'choco' was found at 'C:\ProgramData\chocolatey\bin\choco.exe'.
WARNING: An existing Chocolatey installation was detected. Installation will not continue.
For security reasons, this script will not overwrite existing installations.

Please use `choco upgrade chocolatey` to handle upgrades of Chocolatey itself.

Chocolatey v0.12.1

Upgrading the following packages:

python;visualstudio2019-workload-vctools

By upgrading, you accept licenses for the packages.

You have python v3.10.2 installed. Version 3.10.4 is available based on your source(s).

Progress: Downloading python3 3.10.4... 100%

Progress: Downloading python 3.10.4... 100%

python3 v3.10.4 [Approved]

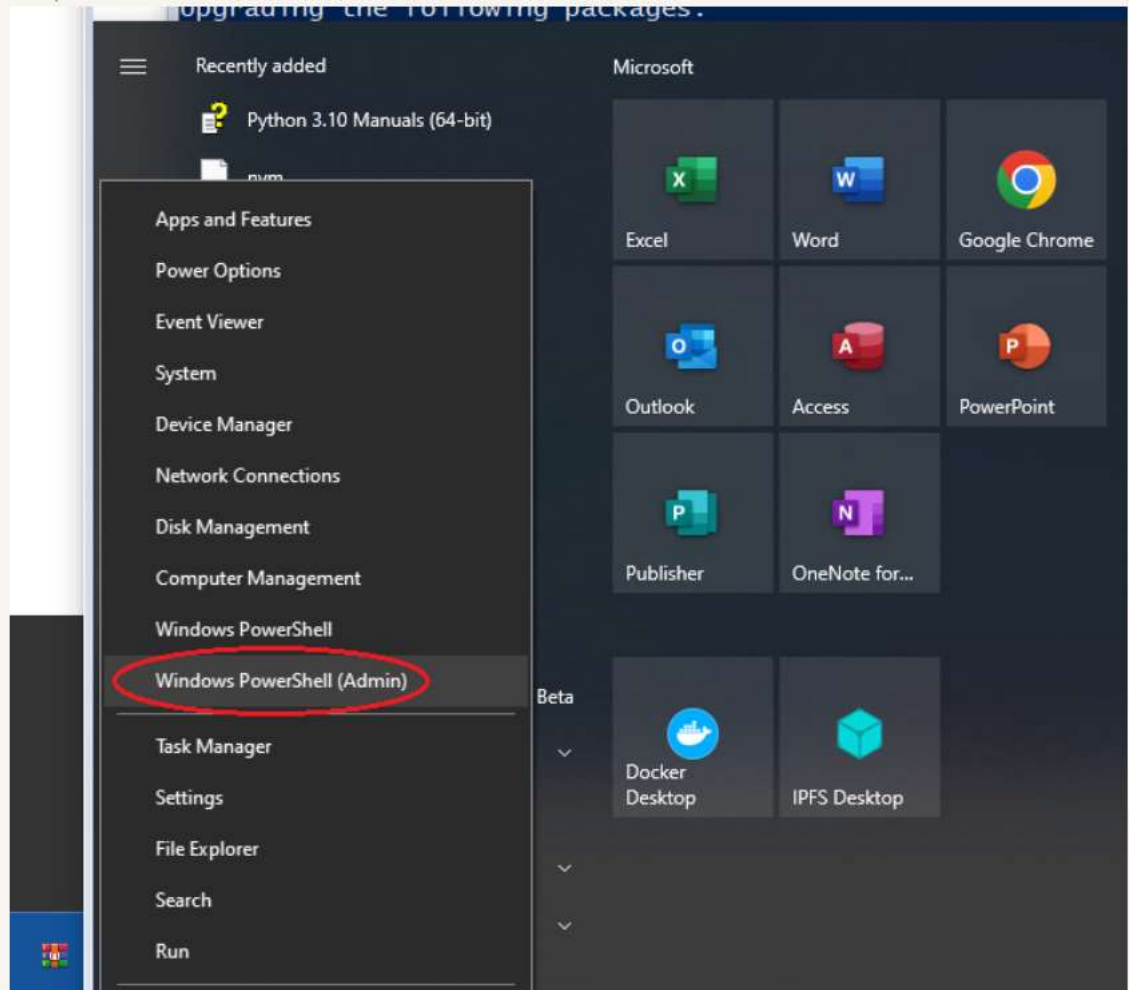
python3 package files upgrade completed. Performing other installation steps.

Installing 64-bit python3...

Chocolatey might find Python or other tools already installed, it will skip and install the next build tools package.

Шаг 2. Установите Truffle с помощью NPM с глобальным флагом в вашем терминале.

Проверьте, правильно ли установлен NodeJS. Откройте терминал Powershell с правами администратора. Вы можете сделать это, щелкнув правой кнопкой мыши кнопку «Пуск» в Windows и выбрав «Windows PowerShell (Admin)».



Введите эту команду в терминал Windows Powershell: `node -v` и `npm -v` эти команды должны вернуть версию Node.js и NPM, если они установлены правильно.

```
Administrator Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\WINDOWS\system32> node -v
v16.15.0
PS C:\WINDOWS\system32> npm -v
8.5.5
PS C:\WINDOWS\system32>
```



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\WINDOWS\system32> node --
v16.15.0
PS C:\WINDOWS\system32> npm --
v5.5.1
PS C:\WINDOWS\system32> npm install -g truffle
npm WARN deprecated mkdirp@0.5.1: This package is broken and no longer maintained. 'mkdirp' itself supports promises now, please switch to that.
npm WARN deprecated har-validator@5.1.3: this library is no longer supported
npm WARN deprecated ipfs-http-client@0.8.0: This module has been superseded by the multiFormats module
npm WARN deprecated cbor@1.1.9: This module has been superseded by the multiFormats module
npm WARN deprecated ipfs-dag-cbor@0.17.1: This module has been superseded by @ipld/dag-cbor and multiFormats
npm WARN deprecated multiaddr@1.8.4: This module has been superseded by the multiFormats module
npm WARN deprecated @ensdomains/ens@0.2.1: Package is deprecated in favour of @ensdomains/ensjs
npm WARN deprecated uuid@2.0.1: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated uuid@1.4.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated multibase@0.6.1: This module has been superseded by the multiFormats module
npm WARN deprecated multibase@0.7.0: This module has been superseded by the multiFormats module
npm WARN deprecated uuid@1.1.1: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated uuid@1.1.1: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated multibase@0.6.0: This module has been superseded by the multiFormats module
npm WARN deprecated uuid@1.1.1: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://v8.dev/blog/math-random for details.
npm WARN deprecated multibase@0.6.0: This module has been superseded by the multiFormats module
npm WARN deprecated multibase@0.6.0: This module has been superseded by the multiFormats module
npm WARN deprecated multibase@0.6.0: This module has been superseded by the multiFormats module
npm WARN deprecated multicodec@0.5.7: This module has been superseded by the multiFormats module
npm WARN deprecated ipfs-dag-pb@0.10.0: This module has been superseded by @ipld/dag-pb and multiFormats
npm WARN deprecated multicodec@2.1: This module has been superseded by the multiFormats module
npm WARN deprecated node-ipc@0.11.0: Please upgrade to @nestjs/node-ipc: the non-scoped node-ipc package is deprecated and only the @nestjs/node-ipc package will receive updates in the future
npm WARN deprecated multicodec@2.1.3: This module has been superseded by the multiFormats module
npm WARN deprecated multicodec@2.1.3: This module has been superseded by the multiFormats module
npm WARN deprecated multicodec@2.1.3: This module has been superseded by the multiFormats module
npm WARN deprecated multicodec@2.1.3: This module has been superseded by the multiFormats module
added 11 packages, removed 100 packages, changed 826 packages, and audited 838 packages in 2s
90 packages are looking for funding
  run `npm fund` for details
11 high-severity vulnerabilities
To address issues that do not require attention, run:
  npm audit fix
Some issues need review, and may require choosing
a different dependency.

You might see some warnings, you can ignore them.
For a successful installation you are looking for "added packages"
```

Проверьте успешность установки Truffle, введя эту команду в терминале Powershell: `truffle`

```
Administrator: Windows PowerShell
PS C:\WINDOWS\system32> truffle
Truffle v5.5.12 - a development framework for Ethereum

Usage: truffle <command> [options]

Commands:
  cli.bundled.js build      Execute build pipeline (if configuration present)
  cli.bundled.js compile    Compile contract source files
  cli.bundled.js config     Set user-level configuration options
  cli.bundled.js console    Run a console with contract abstractions and
                             commands available
  cli.bundled.js create     Helper to create new contracts, migrations and tests
  cli.bundled.js dashboard  Start Truffle Dashboard to sign development
                             transactions using browser wallet
  cli.bundled.js db         Database interface commands
  cli.bundled.js debug      Interactively debug any transaction on the
                             blockchain
  cli.bundled.js deploy     (alias for migrate)
  cli.bundled.js develop    Open a console with a local development blockchain
  cli.bundled.js exec       Execute a JS module within this Truffle environment
  cli.bundled.js help       List all commands or provide information about a
                             specific command
  cli.bundled.js init       Initialize new and empty Ethereum project
  cli.bundled.js install    Install a package from the Ethereum Package Registry
  cli.bundled.js migrate    Run migrations to deploy contracts
  cli.bundled.js networks   Show addresses for deployed contracts on each
                             network
  cli.bundled.js obtain     Fetch and cache a specified compiler
  cli.bundled.js opcode     Print the compiled opcodes for a given contract
  cli.bundled.js preserve   Save data to decentralized storage platforms like
                             IPFS and Filecoin
  cli.bundled.js publish    Publish a package to the Ethereum Package Registry
  cli.bundled.js run        Run a third-party command
  cli.bundled.js test       Run JavaScript and Solidity tests
  cli.bundled.js unbox      Download a Truffle Box, a pre-built Truffle project
  cli.bundled.js version    Show version number and exit
  cli.bundled.js watch      Watch filesystem for changes and rebuild the project
                             automatically
```


Установка Truffle (2 вариант)

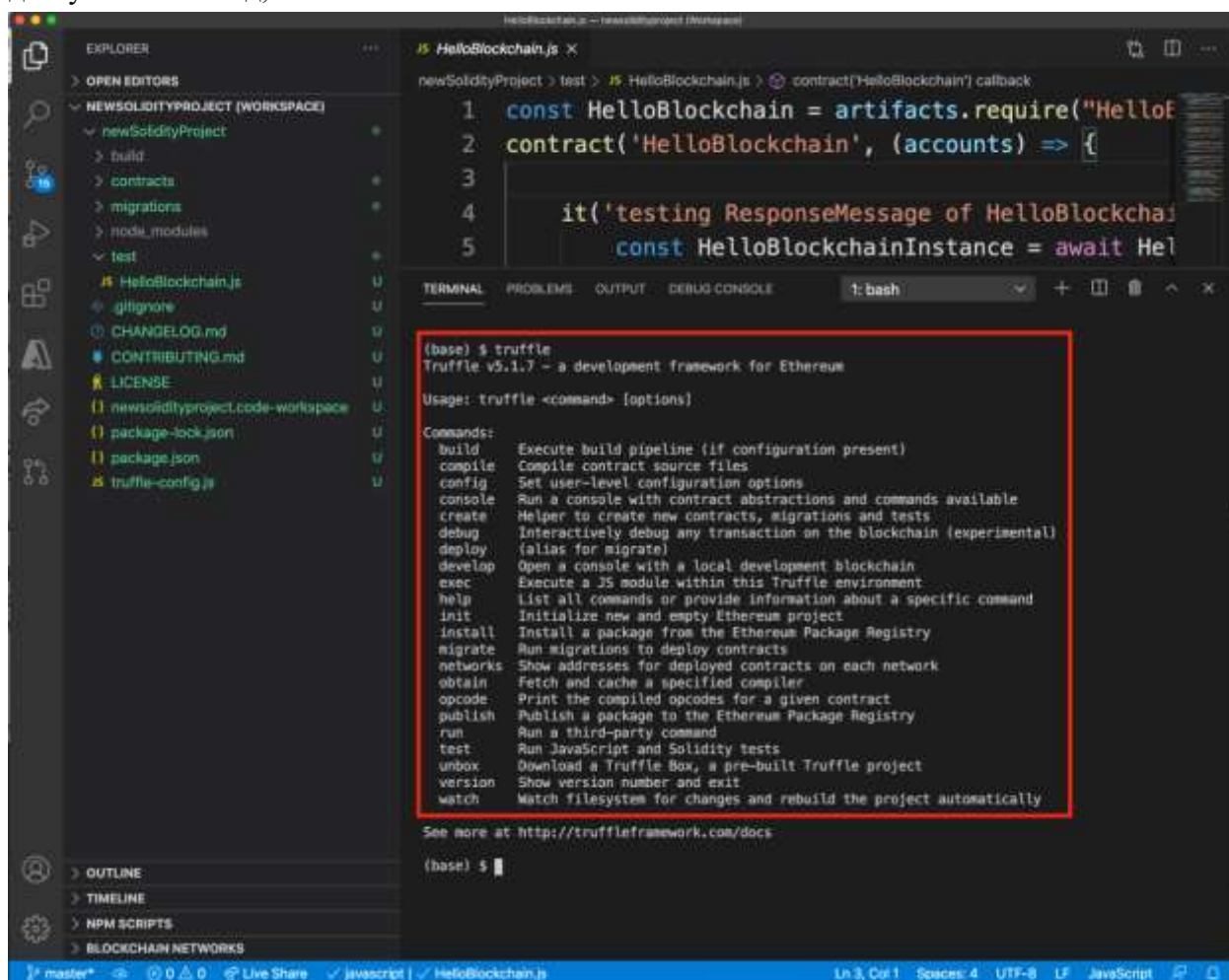
Truffle можно установить с помощью диспетчера пакетов узла (NPM). Для этого введите в терминале следующую команду:

```
npm install -g truffle
```

Чтобы убедиться, что платформа Truffle уже установлена, введите:

```
truffle
```

В выходных данных вы увидите сведения об установленной версии Truffle и список доступных команд, вот так:



```
(base) $ truffle
Truffle v5.1.7 - a development framework for Ethereum

Usage: truffle <command> [options]

Commands:
  build      Execute build pipeline (if configuration present)
  compile    Compile contract source files
  config     Set user-level configuration options
  console    Run a console with contract abstractions and commands available
  create     Helper to create new contracts, migrations and tests
  debug      Interactively debug any transaction on the blockchain (experimental)
  deploy     (alias for migrate)
  develop    Open a console with a local development blockchain
  exec       Execute a JS module within this Truffle environment
  help       List all commands or provide information about a specific command
  init       Initialize new and empty Ethereum project
  install    Install a package from the Ethereum Package Registry
  migrate    Run migrations to deploy contracts
  networks   Show addresses for deployed contracts on each network
  obtain     Fetch and cache a specified compiler
  opcode     Print the compiled opcodes for a given contract
  publish    Publish a package to the Ethereum Package Registry
  run        Run a third-party command
  test       Run JavaScript and Solidity tests
  unbox      Download a Truffle Box, a pre-built Truffle project
  version    Show version number and exit
  watch      Watch filesystem for changes and rebuild the project automatically

See more at http://truffleframework.com/docs

(base) $
```

Дополнительные сведения о начале работы с Truffle см. в [кратком руководстве по Truffle](#).

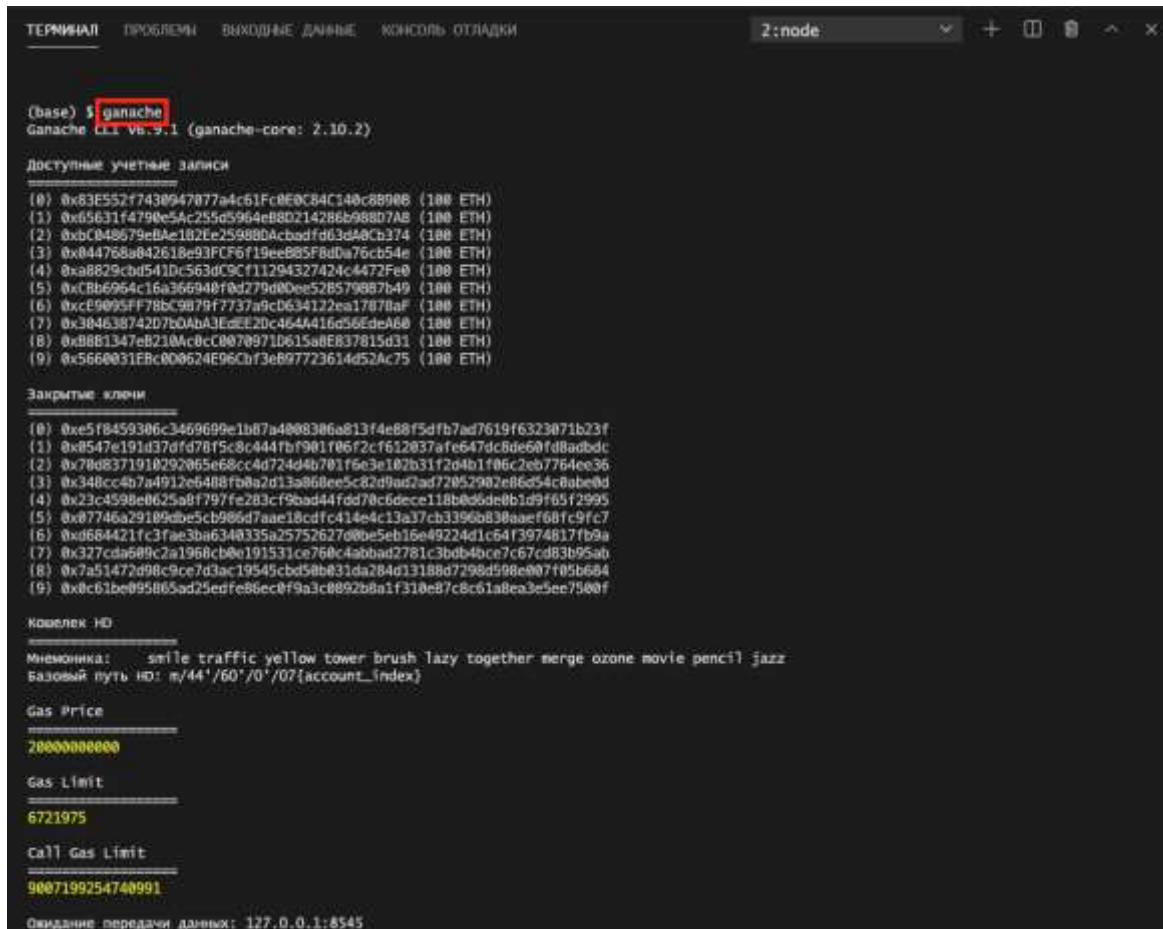
Ganache

Наиболее популярным локальным клиентом блокчейна для Ethereum считается Ganache. Ganache обеспечивает безопасную и детерминированную среду для разработки, развертывания и тестирования. Ganache можно использовать из командной строки, программно с помощью Node.js или в браузере. В этом модуле мы применим [версию командной строки](#), с которой можно взаимодействовать напрямую из терминала.

Чтобы установить Ganache в проекте, перейдите в терминал. Щелкните в этом окне правой кнопкой мыши и выберите New Terminal. Когда откроется новое окно терминала, выполните следующий код:

npm install ganache --global

Когда установка Ganache завершится, выполните следующий код:
ganache



```
TERMINAL  ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  2:node

(base) $ ganache
Ganache CLI v6.9.1 (ganache-core: 2.10.2)

Доступные учетные записи
=====
(0) 0x83E552f7430947077a4c61Fc0E0C84C140c8B906 (100 ETH)
(1) 0xd5631f4790e5Ac255d5964e8B0214286b988D7A8 (100 ETH)
(2) 0xbC040679eBAe182Ec259800AcbaDfd63dA0Ch374 (100 ETH)
(3) 0x044768a842618e93FCF6f19ee885F8dDa76cb54e (100 ETH)
(4) 0xa8829cbd541Dc563dC9cf1129432742c4472Fe0 (100 ETH)
(5) 0xcBb6964c16a366948f0d279d00ee52B579887b49 (100 ETH)
(6) 0xc9095FF78bC9B79f7737a9cD634122ea17878aF (100 ETH)
(7) 0x384638742D7b0AbA3EdE20c464416d56EdeA60 (100 ETH)
(8) 0xB8B1347eB230Ac0c0070971D615a8E837815d31 (100 ETH)
(9) 0x5666031EBc00624E96Cb73e897723614d52Ac75 (100 ETH)

Закрытые ключи
=====
(0) 0xe5f8458306c3469699e1b87a4008306a813f4e88f5dfb7ed7619f6323071b23f
(1) 0x8547e191d37dfd78f5c8c444fbf901f06f2cf612037afe647dc8de60fd8adbdc
(2) 0x78d8371910292065e68cc4d724d4b701f6e3e102b31f2d4b1f06c2eb7764ee36
(3) 0x348cc4b7a4912e6488fb0a2d13a060ee5c82d9ad7ad72052902e86c54c8abe0d
(4) 0x23c4598e0625a8f797fe283cf9bad44fdd70c6dece118bd6de0b1d9f65f2995
(5) 0x07746a29109dbe5cb986d7aae18cdfc414e4c13a37cb3396b830aaef68f9fc7
(6) 0xd664421f3fae3ba6340335a25752627d00e5eb16e49224d1c64f3974817fb9a
(7) 0x327cda600c2a1968cb0e191531ce760c4abbad2781c3b0b4bce7c67cd83b95ab
(8) 0x7a51472d98c9ce7d3ac19545cbd50b831da284d13188d7298d598e007f05b684
(9) 0x8c61be95865ad25edfe86ec0f9a3c0892b8a1f310e87c8c61a8ea3e5ee7500f

Кошелек HD
=====
Мнемоника: smile traffic yellow tower brush lazy together merge ozone movie pencil jazz
Базовый путь HD: m/44'/60'/0'/0'/0/[account_index]

Gas Price
=====
20000000000

Gas Limit
=====
6721975

Call Gas Limit
=====
9007199254740991

Ожидание передачи данных: 127.0.0.1:8545
```

Обратите внимание, что в этом блокчейне создано 10 учетных записей, каждая из которых получила 100 тестовых единиц эфира. С каждой учетной записью также сопоставлен определенный закрытый ключ. И у каждой учетной записи есть *мнемоника*. Мнемоникой называется уникальная фраза из 12 слов, которая предоставляет доступ к кошельку и позволяет выполнять транзакции в этой учетной записи.

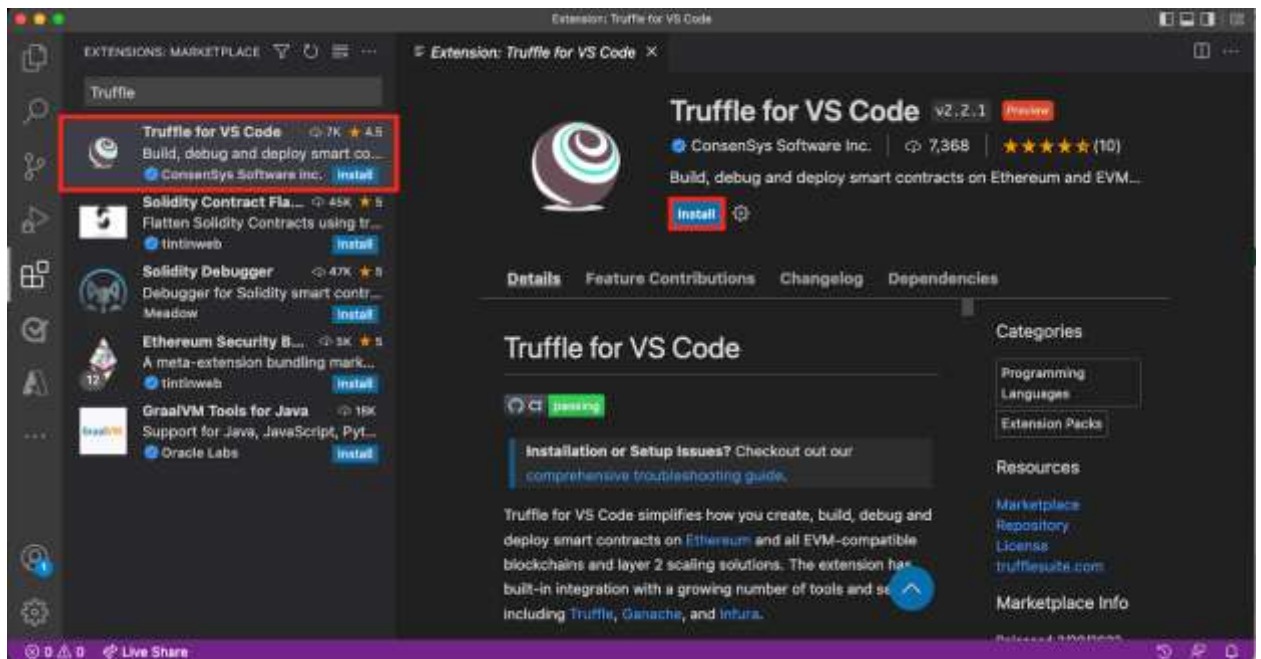
Кроме того, в выходных данных отображается адрес блокчейна. Этот адрес мы применим для подключения к блокчейну. По умолчанию используется адрес 127.0.0.1:8545.

Установка расширения Truffle для VS Code

[Расширение Truffle for VS Code](#) упрощает создание, сборку и развертывание смарт-контрактов в реестрах Ethereum. Это расширение имеет встроенную интеграцию с Truffle, Ganache, а также другими инструментами и службами. В этом модуле мы будем использовать его, чтобы создавать и тестировать смарт-контракты.

Установка расширения

В Visual Studio Code выберите пункт **Расширения** на боковой панели слева. Найдите **Truffle для VS Code** и выберите расширение, чтобы установить.



Прежде, чем использовать расширение Truffle убедитесь, что установлено следующее:

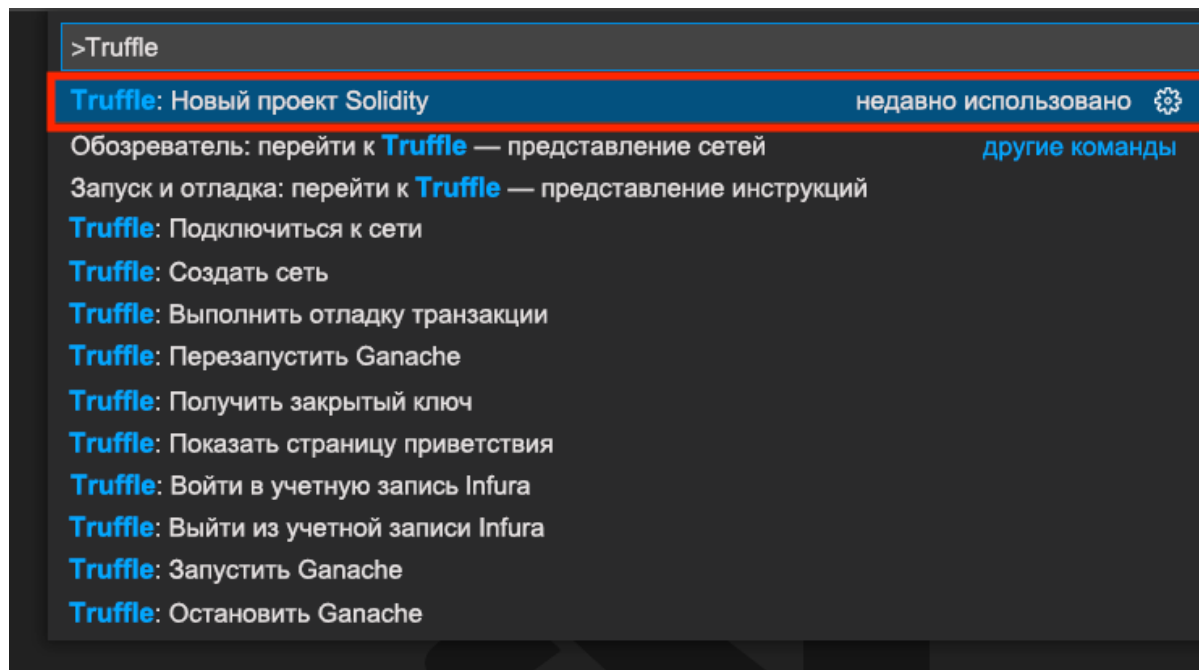
- **Node.js и npm** — чтобы проверить наличие Node.js, откройте терминал и введите `node`. Если Node.js установлен, в окне терминала отобразятся сведения об установленной на компьютере версии Node.js. Кроме того, можно убедиться, что установлен диспетчер пакетов Node (npm). Для этого введите `npm` в терминале.
- **Git**. Чтобы проверить наличие Git, откройте терминал и введите `git`. Если Git установлен, в окне терминала отобразится список доступных команд git.
- **Truffle Suite**: расширение предоставляет ссылку на установку средств разработчика Truffle Suite (требуется, пока расширение находится в общедоступной предварительной версии).
- **Ganache** — расширение предоставляет ссылку на установку Ganache (требуется, пока расширение предоставляется в общедоступной предварительной версии).

Если у вас не установлено необходимое программное обеспечение или его версии не соответствуют минимальным требованиям, расширение предоставит советы по установке этих инструментов.

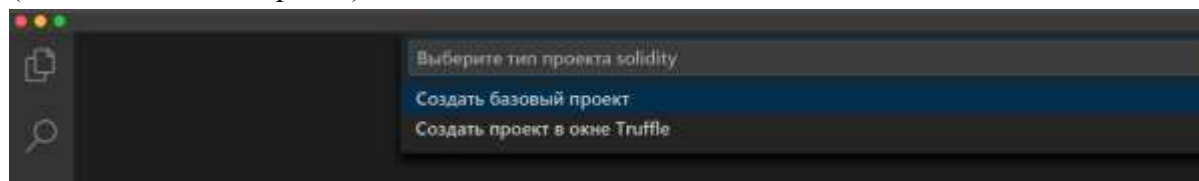
Начало работы

Убедившись в наличии всех установленных зависимостей, создайте первый проект с помощью расширения Truffle. Для этого выполните следующие действия:

1. Добавьте на компьютер пустой каталог для проекта. Чтобы создать каталог из среды Visual Studio Code, последовательно выберите элементы **Terminal (Терминал)** > **New Terminal (Новый терминал)**, а затем введите `mkdir newSolidityProject`. Запишите расположение этого каталога. Он понадобится вам позднее.
2. В Visual Studio Code последовательно откройте разделы **View (Просмотр)** > **Command Palette (Палитра команд)**. В поле поиска введите `Truffle: New Solidity Project`. По мере ввода появляется список предложений.

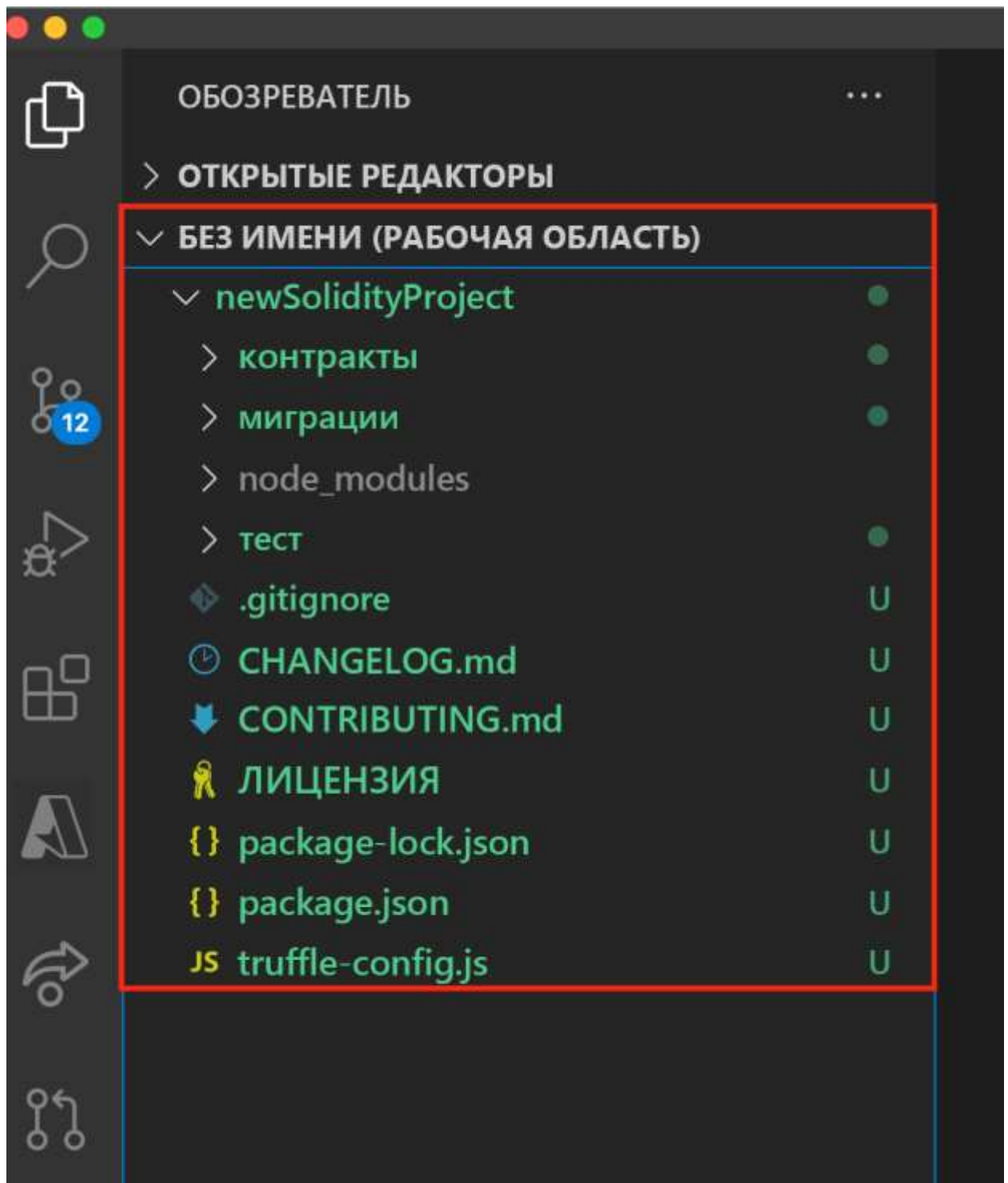


3. В качестве типа для проекта Solidity выберите вариант **Create basic project** (Создать базовый проект).



4. С помощью панели обозревателя найдите папку, созданную на шаге 1. Выберите эту папку. Справа в нижнем углу окна отображается состояние **Creating new project** (Создание проекта).

Когда завершится создание проекта Solidity, откройте панель обозревателя и просмотрите файлы проекта.



Созданный проект содержит стандартный код для Solidity. В нем вы увидите следующие каталоги:

- **contracts** — содержит контракт *HelloBlockchain.sol*.
- **migrations** — содержит код миграции для контракта *HelloBlockchain*, написанный на JavaScript.
- **test:** содержит тест для контракта *HelloBlockchain*, написанного на JavaScript.

Кроме того, здесь есть файлы конфигурации.

- **package.json:** определение сведений о проекте и зависимостей.
- **truffle-config.json:** определение зависимостей и конфигурации для Truffle.

Чтобы сохранить рабочую область, последовательно выберите элементы **File**

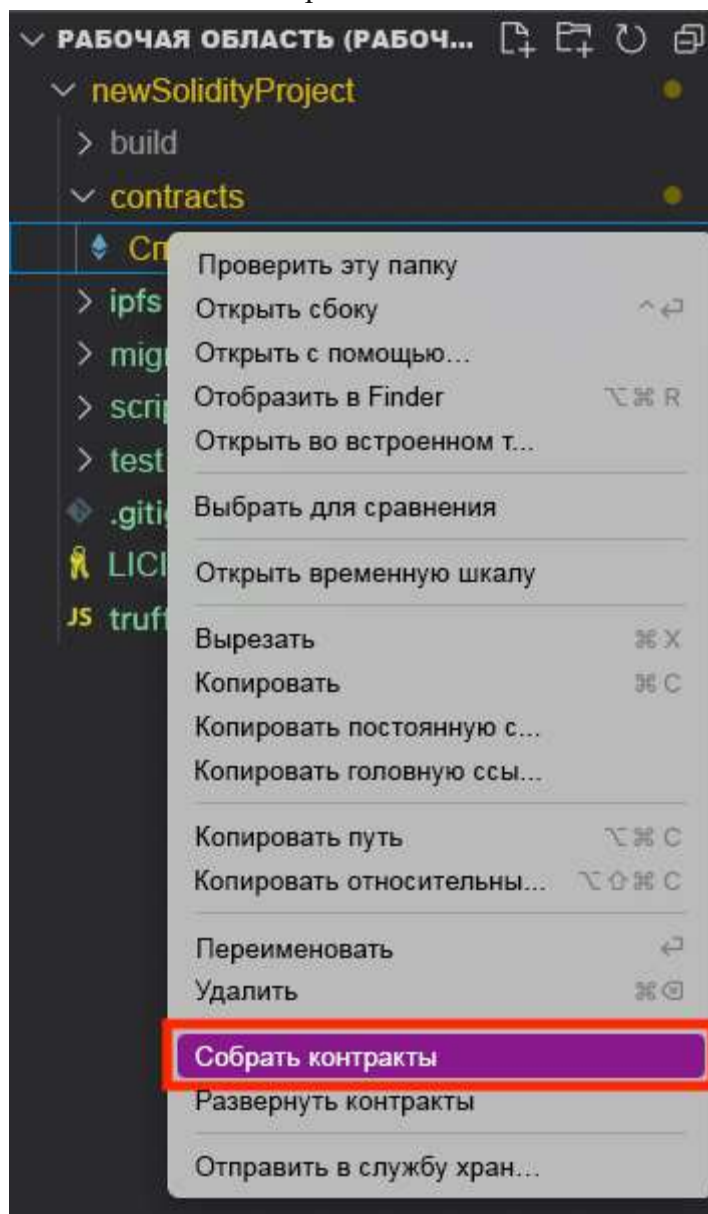
(Файл)>Save Workspace As (Сохранить рабочую область как). Присвойте ей имя *newSolidityProject*.

Теперь вернемся к проекту. Изучение мы начнем с каталога *contracts*.

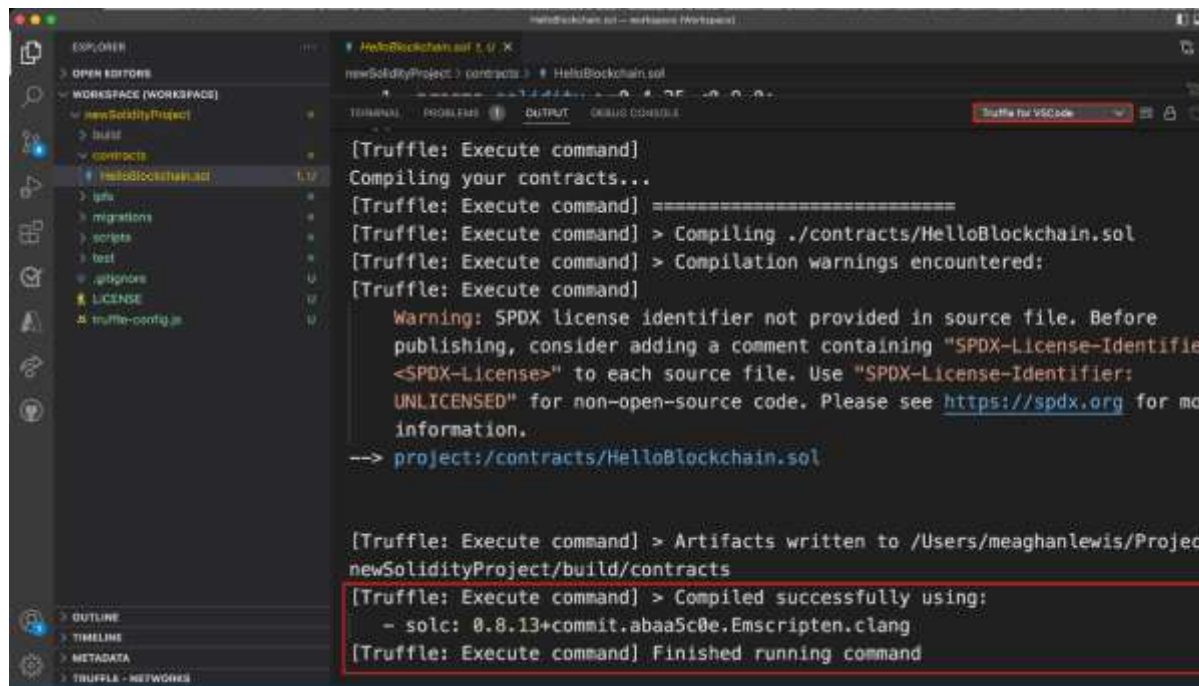
Компиляция контрактов

И здесь мы выберем смарт-контракт *HelloBlockchain.sol* из папки *contracts*.

1. На панели обозревателя в папке **contracts** щелкните имя контракта **HelloBlockchain.sol** правой кнопкой мыши.
2. Выберите элемент **Build Contracts** (Компилировать контракты), чтобы скомпилировать смарт-контракт. В окне в нижнем углу справа отобразится ход компиляции контрактов.



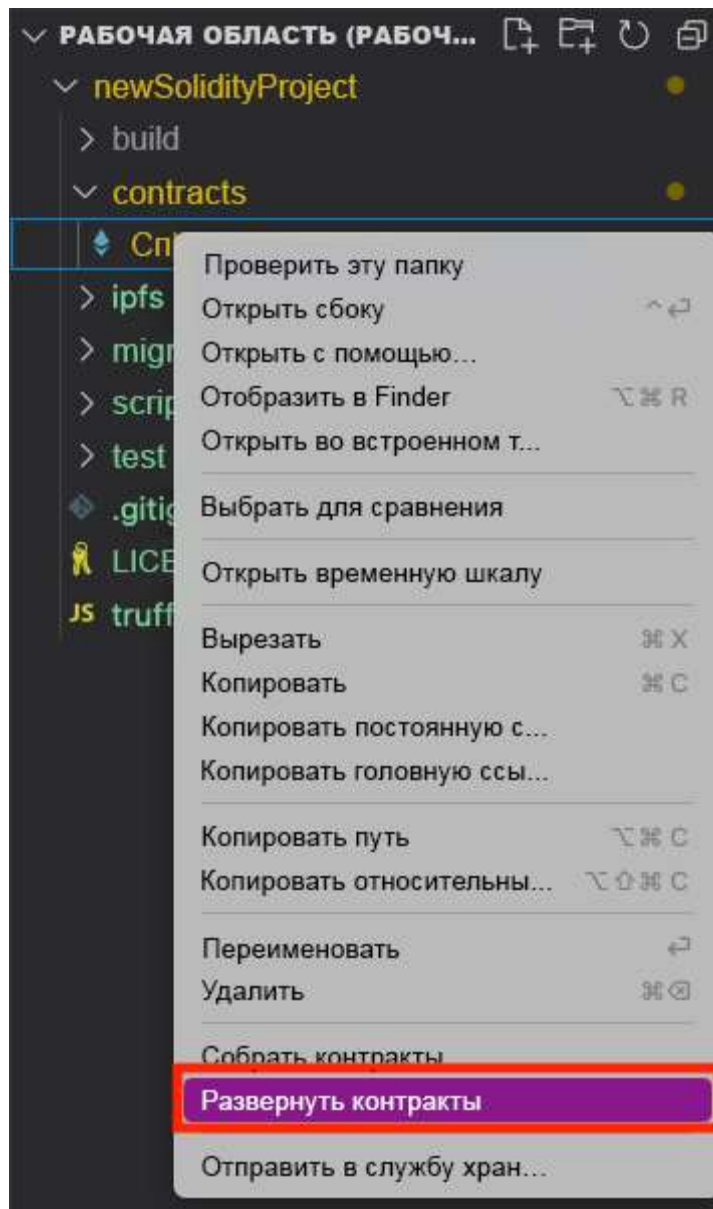
3. Щелкните **View (Просмотреть)** > **Output (Выходные данные)**, чтобы увидеть сведения о скомпилированном контракте. Возможно, в этом окне понадобится выбрать **Truffle for VS Code** (Truffle для VS Code), чтобы просмотреть выходные данные расширения.



Развертывание контрактов

После успешной компиляции контракта его можно развернуть локально.

1. На панели обозревателя перейдите к папке **contracts** и щелкните имя контракта **HelloBlockchain.sol** правой кнопкой мыши.
2. Выберите **Deploy Contracts** (Развернуть контракты).



В окне выходных данных (**View (Просмотр)**—>**Output (Выходные данные)**) отображаются сведения о развернутых миграциях и контрактах.

```
# HelloBlockchain.sol x
newSolidityProject > contracts > HelloBlockchain.sol
1 pragma solidity >=0.4.25 <0.9.0;

[Truffle: Execute command]
Deploying 'HelloBlockchain'

[Truffle: Execute command] > transaction hash:
0x5b2f0fa2a58b030ca843ec0bf5e27121602cef9ebe25bb17b9d372b23f6c562f
[Truffle: Execute command] - Blocks: 0 Seconds: 0
[Truffle: Execute command] > Blocks: 0 Seconds: 0
> contract address: 0x5B9b4cfCfCe2b4A3Ffe746F73F7CC31292E5A74
> block number: 1
> block timestamp: 1653347292
> account: 0xde9D95097ef1e396260c116C3e75588e784A1e83
> balance: 999.998058561625
> gas used: 575241 (0x8c709)
> gas price: 3.375 gwei
> value sent: 0 ETH
> total cost: 0.001941438375 ETH

[Truffle: Execute command] > Saving artifacts

> Total cost: 0.001941438375 ETH

Summary
=====
> Total deployments: 1
> Final cost: 0.001941438375 ETH

[Truffle: Execute command]
[Truffle: Execute command] Finished running command
[Truffle for VSCode] Deploy succeeded
```

Здесь вы увидите важные сведения и метаданные для развернутого контракта:

- адрес контракта;
- метка времени того блока, в который включена транзакция создания контракта;
- адрес учетной записи, с помощью которой развернут контракт;
- остаток на счете этой учетной записи (в единицах эфира) после транзакции. Этот остаток на счете равен 100 ETH (начальное значение по умолчанию) за вычетом общей стоимости;
- количество и стоимость использованного газа. Термин *gas* здесь определяет стоимость выполнения транзакции или контракта на платформе блокчейна Ethereum. Можно провести параллель между ним и топливом для автомобиля. Общая стоимость вычисляется как произведение цены газа и использованного объема газа.

Создание смарт-контракта

Мы добавим новый смарт-контракт к созданному ранее проекту `newSolidityProject`.

Смарт-контракт, который вы сейчас создадите, позволяет отслеживать состояние элементов, приобретенных в Интернет-магазине. При создании контракта устанавливается состояние доставки `Pending`. При отправке элемента устанавливается состояние доставки `Shipped` и создается соответствующее событие. После доставки для элемента устанавливается состояние доставки `Delivered` и создается другое событие.

Чтобы начать это упражнение, сделайте следующее:

1. В созданном ранее проекте перейдите к каталогу `contracts`, щелкните правой кнопкой мыши папку, выберите действие создания файла и введите для него

имя *Shipping.sol*.

2. Скопируйте приведенное ниже содержимое контракта и вставьте его в новый файл.

```
pragma solidity >=0.4.25 <0.9.0;

contract Shipping {
    // Our predefined values for shipping listed as enums
    enum ShippingStatus { Pending, Shipped, Delivered }

    // Save enum ShippingStatus in variable status
    ShippingStatus private status;

    // Event to launch when package has arrived
    event LogNewAlert(string description);

    // This initializes our contract state (sets enum to Pending once the
    program starts)
    constructor() public {
        status = ShippingStatus.Pending;
    }
    // Function to change to Shipped
    function Shipped() public {
        status = ShippingStatus.Shipped;
        emit LogNewAlert("Your package has been shipped");
    }

    // Function to change to Delivered
    function Delivered() public {
        status = ShippingStatus.Delivered;
        emit LogNewAlert("Your package has arrived");
    }

    // Function to get the status of the shipping
    function getStatus(ShippingStatus _status) internal pure returns
(string memory) {
        // Check the current status and return the correct name
        if (ShippingStatus.Pending == _status) return "Pending";
        if (ShippingStatus.Shipped == _status) return "Shipped";
        if (ShippingStatus.Delivered == _status) return "Delivered";
    }

    // Get status of your shipped item
    function Status() public view returns (string memory) {
        ShippingStatus _status = status;
        return getStatus(_status);
    }
}
```

3. Просмотрите контракт и изучите сведения о том, какие задачи выполняются с его помощью. Убедитесь, что вы можете успешно создать контракт.

4. На панели обозревателя щелкните имя контракта правой кнопкой мыши и выберите элемент **Build Contracts** (Компилировать контракты), чтобы скомпилировать смарт-контракт.

Добавление миграции

Теперь добавьте файл миграции, чтобы получить возможность развернуть контракт в сети Ethereum.

1. В области обозревателя наведите указатель мыши на папку **migrations** и выберите элемент **New File** (Новый файл). Назовите файл `2_Shipping.js`.
2. Добавьте в этот файл следующий код:

```
const Shipping = artifacts.require("Shipping");

module.exports = function (deployer) {
  deployer.deploy(Shipping);
};
```

Развертывание

Теперь убедитесь, что вы можете успешно развернуть контракт, прежде чем продолжать работу. Для этого щелкните имя контракта правой кнопкой мыши и выберите элемент **Deploy Contracts** (Развернуть контракты). В появившемся окне выберите элемент **development** (разработка) и дождитесь завершения развертывания.

Просмотрите сведения в окне выходных данных. Найдите развертывание `2_Shipping.js`.

```
Shipping.sol 3, 11 X 2_Shipping.js 11
newSolidityProject > contracts > Shipping.sol
[Truffle: Execute command]
2_Shipping.js
=====
[Truffle: Execute command]
Deploying 'Shipping'
=====
[Truffle: Execute command] > transaction hash:
0xe14b285acdeb480330ad5c9a74e57c08e4af5b218477f2f808ff4ef33d210687
[Truffle: Execute command] - Blocks: 0 Seconds: 0
[Truffle: Execute command] > Blocks: 0 Seconds: 0
> contract address: 0xD4cba6514898E961af9a6564d8A5F28D66930C70
> block number: 3
> block timestamp: 1653416546
> account: 0xde9095097ef1e396260c116C3e75588e784A1e83
> balance: 999.995195383292261186
> gas used: 309146 (0x4b79a)
> gas price: 3.177282298 gwei
> value sent: 0 ETH
> total cost: 0.000982244113297508 ETH

[Truffle: Execute command] > Saving artifacts
=====
> Total cost: 0.000982244113297508 ETH

Summary
=====
> Total deployments: 2
> Final cost: 0.002863178332738814 ETH

[Truffle: Execute command]
[Truffle: Execute command] Finished running command
[Truffle for VSCode] Deploy succeeded
```

Тестирование смарт-контракта

В этой части мы напишем на JavaScript новый тест для контракта на доставку. Для написания теста лучше всего использовать Solidity, но язык JavaScript более распространен. Чтобы протестировать смарт-контракт, мы используем Truffle.

Начало тестирования

Для начала нам нужно создать новый тестовый файл.

1. В панели "Обозреватель" правой кнопкой мыши щелкните папку **test**. Выберите **Новый файл** и создайте файл с именем *TestShipping.js*.
2. Скопируйте и вставьте в тестовый файл следующий код:

```
const ShippingStatus= artifacts.require("./Shipping.sol");
contract('Shipping', () => {

  it("should return the status Pending", async ()=> {
    // Instance of our deployed contract
    const instance = await ShippingStatus.deployed();
    // Checking the initial status in our contract
    const status = await instance.Status();
    // Checking if the status is initially Pending as set in the
```

```

constructor
    assert.equal(status, "Pending");
  });
it("should return the status Shipped", async () => {
  // Instance of our deployed contract
  const instance = await ShippingStatus.deployed();

  // Calling the Shipped() function
  await instance.Shipped();

  // Checking the initial status in our contract
  const status = await instance.Status();

  // Checking if the status is Shipped
  assert.equal(status, "Shipped");
});

it("should return the status Delivered", async () => {

  // Instance of our deployed contract
  const instance = await ShippingStatus.deployed();

  // Calling the Shipped() function
  await instance.Delivered();

  // Checking the initial status in our contract
  const status = await instance.Status();

  // Checking if the status is Delivered
  assert.equal(status, "Delivered");
});
});

```

Тестирование события

Мы применим пакет `truffle-assertions` для проверки событий, отправляемых в контракте. Этот пакет позволит нам гарантировать, что события создаются во время транзакции.

1. В окне терминала установите библиотеку с помощью команды `npm install truffle-assertions`.
2. Добавьте следующий код в тестовый файл в строку 2 после инструкции `require` для контракта `ShippingStatus`:
 JavaScriptКопировать

```
const truffleAssert = require('truffle-assertions');
```
3. Добавьте тест для проверки получения ожидаемого описания от события. Поместите этот тест в конец файла, после последнего теста. Для него нужно создать новую строку прямо перед набором закрывающих фигурных скобок, расположенных в последней строке.

```

it('should return correct event description', async()=>{

    // Instance of our deployed contract
    const instance = await ShippingStatus.deployed();

    // Calling the Delivered() function
    const delivered = await instance.Delivered();

    // Check event description is correct
    truffleAssert.eventEmitted(delivered, 'LogNewAlert', (event) =>{
        return event.description == 'Your package has arrived';
    });
});

```

Использование async/await

Функция **.deployed()** возвращает обещание. Поэтому перед этой функцией мы укажем **await**, а перед тестовым кодом — **async**. Такая схема означает, что после развертывания контракта тест приостанавливается до того момента, когда будет выполнено это обещание.

Этот шаблон широко применяется в тестах, так как почти все транзакции смарт-контракта являются асинхронными. Это связано с тем, что перед добавлением транзакции в реестр блокчейна требуется проверка или майнинг.

Обычно следует стремиться к 100-процентному покрытию контракта тестами, особенно если он будет развертываться в основной сети Ethereum (*Main Net*).

Выполнение теста

Для этого введите в терминале следующую команду:

```
truffle test
```

Вы должны получить подтверждение, что все тесты пройдены успешно:

```
Contract: HelloBlockchain
```

- ✓ testing ResponseMessage of HelloBlockchain
- ✓ testing Responder of HelloBlockchain
- ✓ testing RequestMessage of HelloBlockchain
- ✓ testing State of HelloBlockchain
- ✓ testing Requestor of HelloBlockchain
- ✓ testing SendRequest of HelloBlockchain (51ms)
- ✓ testing SendResponse of HelloBlockchain (46ms)

```
Contract: Shipping
```

- ✓ should return the status Pending
- ✓ should return the status Shipped (59ms)
- ✓ should return the status Delivered (58ms)
- ✓ should return correct event description (39ms)