**Создание бота в Телеграм[[1]](#footnote-1)**

Боты — это небольшие приложения, которые работают внутри Телеграм (далее тг). Возможности ботов безграничны — от простых скриптов до сложных мини-приложений. Что можно делать с ботами?

* Замена целых веб-сайтов
* Управляйте своим бизнесом
* Принимайте платежи
* Создание пользовательских инструментов
* Интеграция с сервисами и устройствами
* Организатор игр
* Создание социальных сетей
* Монетизируйте свой сервис
* Продвигайте свой проект
* Что-нибудь ещё!

Независимо от того, являетесь ли вы новичком или профессиональным программистом, вы можете создавать персонализированные инструменты с помощью платформы ботов.

Для особенно пытливых и тех, кто не боится гуглить термины буду вот так выделять сложную и необязательную информацию:

**Тг-боты** — это специальные аккаунты, для настройки которых не нужен номер телефона. Боты подключаются к серверу своего владельца, который обрабатывает входные данные и запросы от пользователей. Промежуточный сервер тг обрабатывает все шифрование и обмен данными. Разработчики общаются с этим сервером через Bot API. **Bot API** — это интерфейс (фреймворк, можно сказать – набор инструментов, а также язык обращения к тг) на основе **HTTP** (стандартного интернет протокола обмена информацией), созданный для разработки ботов, мини-приложений и даже своего собственного Телеграма.

**Регистрация в BotFather**

Чтобы начать работу, напишите @BotFather в тг в поиске каналов (выбирайте с синей галочкой). **@BotFather** – это официальный бот-ассистент, который помогает пользователям управлять ботами. Когда вы начнете диалог, бот предложит вам ссылку на страницу с мануалом, где можно изучить всю информацию самостоятельно, или можно просто следовать инструкциям, которые он дает.

Чтобы взаимодействовать с @BotFather, необходимо писать команду ему прямо в диалог, или нажимать на нее как на ссылку. Команды (теги) пишутся на английском языке (латиница) без пробелов, а в начале слова ставится знак «/».

Чтобы создать (зарегистрировать) нового бота, введите первую команду «/newbot». Следующим сообщением введите имя бота, которое будет отображаться всем пользователям. Третье сообщение должно содержать имя пользователя бота, которое будет использоваться как ссылка на него, когда вы будете упоминать его в диалогах, делиться с ним друзьям или настраивать его работу. Для имен пользователей ботов обычно требуется суффикс «\_bot» в конце имени.

Поздравляю, вы зарегистрировали бота и получаете его токен аутентификации. Токен вашего бота — это его уникальный идентификатор. Храните его в безопасном месте и делитесь им только с людьми, которым нужен прямой доступ к боту. Все, у кого есть ваш токен, будут иметь полный контроль над ним.

**Настройка и основные команды**

Так как подразумевается, что вы можете владеть сразу несколькими ботами – теперь, после каждой введённой команды (почти) потребуется ввести (или указать из списка) название бота, к которому вы намерены применить команду. Например, если вы ошиблись с именем бота, вы пишете соответствующую команду «/setname», чтобы изменить его, вторым сообщением указываете имя пользователя бота, а третьим – новое имя, которое теперь будут видеть вы и его пользователи.

Давайте настроем вашего бота, будем использовать команды, которые преобразят его диалог с пользователями. Команда «/setdescription» добавит описание к боту в начале диалога, которое всегда будет начинаться со слов «Что умеет этот бот?» и состоит не более чем из 520 символов. Не используйте эту команду для списка команд для вашего бота, для этого есть команда «/setcommands». Введите все команды одним сообщением по шаблону «команда - описание», не используя знак «/». Теперь команды будут предлагаться автозаполнением при общении с ботом. Команда «/setabouttext» добавит еще одно короткое описание для бота в его профиль (не более 120 символов). Командой «/setuserpic» можно изменить фото профиля бота.

Некоторых ботов можно вызвать в любом диалоге, чтобы сделать запрос, напишите в любом чате @gif, @wiki, @youtube (заметьте, они не используют суффикс «\_bot», так как он желателен, но не обязателен). Вы увидите, что бот предлагает вам автозаполнение или сделать свой запрос. Ваш бот можно настроить подобным образом, для этого в @BotFather напишите команду «/setinline», и укажите имя своего бота. Чтобы знать, какие запросы самые популярные для вашего бота, используйте для него команду «/setinlinefeedback» и включите функцию сообщением «Enabled» или другим из списка.

**Подготовка к программированию[[2]](#footnote-2)**

Перед началом разработки нужно, чтобы на вашем ПК были установлены следующие программы:

* **Visual studio code** (далее Code) – или любой другой редактор кода на усмотрение. Сделать это просто – открываете официальный сайт, выбираете версию для своего устройства, скачиваете файл, открываете его, устанавливаете эту программу, как устанавливали любую другую. Если вас попросят выбрать язык – выбираете javascript (не путать с java).
* **Node.js** – это среда выполнения кода, написанного на языке javascript. Визуально он представляет собой терминал (консоль для команд). Скачать Node.js можно с официального сайта, а npm установится автоматически вместе с ним, **npm** – это менеджер пакетов (как проводник, но без визуального интерфейса). Его вы не увидите, но будете к нему обращаться через VScode.
* Так как я изучаю для себя **Github** (платформа, сайт, где публикуется открытый код со всего мира), для меня также важно установить на компьютере **Git** со всеми его инструментами. По сути – это интернет-конспект, где можно проследить все версии и этапы разработки проекта. Если вы хотите также, надо открыть Github в браузере и завести себе новый профиль. Заходите, смотрите – <https://github.com/MalkovYar/Spirit_tg_bot>.

Если всё установлено, можно приступать к разработке бота. Если во время установки вы столкнулись с вопросами и проблемами – можете написать мне, контакты в профиле.

Если вы не хотите заводить Github, просто создайте и назовите папку, где будет храниться ваш код, как вы делаете это обычным образом, иначе блок ниже для вас:

Для начала, я создам на GitHub новый приватный **репозиторий** (такое место, где будет храниться все, что относится к проекту), назову его, добавлю описание, и в пункте «Add .gitignore» выберу «Node», в качестве категории.

Теперь надо скопировать это репозиторий себе на компьютер. Сейчас будут использоваться команды для командной строки Windows (для других ОС команды будут отличаться). Чтобы открыть терминал, нажмите сочетание клавиш Win+R, введите в окно «Выполнить» строку «cmd» без кавычек и нажмите «ОК».

Определитесь, в какой папки вы хотите иметь папку с **проектом** вашего бота (в проект входят файлы с кодом и все относящиеся к нему элементы) и скопируйте её путь в отдельное место, чтобы использовать многократно в дальнейшем.

В терминал напишите команду «cd *путь к папке*» и нажмите клавишу «Enter». По умолчанию внутри Терминала вы находитесь в папке своего пользователя, но после выполнения команды, вы перемещаетесь в указанную папку. Следующей командой будет «git clone *ссылка на репозиторий в github*», которая скопирует все файлы из Github в папку, в которой вы находитесь. Переместитесь в папку-копию репозитория командой «cd *название репозитория*».

Откройте папку через правую кнопку мыши (далее ПКМ), выберите пункт «Открыть с помощью Code», или того редактора, который вы установили себе. Слева у меня находится список файлов в папке (пока ничего нет, кроме тех, что относятся к Github), по центру рабочая область, где будем писать код.

Запустите Node.js, чтобы он работал фоном. Теперь, вам нужно найти и открыть терминал внутри редактора и написать в нем команду «npm init», чтобы запустить пакетный менеджер, Вам предложат ввести имя пакета, версию, описание, главный файл по умолчанию, тест команду, Git репозиторий, ключевые слова, автора и лицензию. Вы можете оставить значения по умолчанию, нажимая «Enter». Я поступил так в графе «главный файл по умолчанию», поэтому файл с кодом для бота будет называться index.js. Создайте это файл внутри папки проекта. Когда снова появится пустая строка, введите команду «npm install node-telegram-bot-api» чтобы установить библиотеку, отвечающую за обмен данными между вашим кодом и тг по API. Проверьте, чтобы не было никаких ошибок во время установки.

**Пишем бота на JS**

В данном конспекте я не буду учить вас языку программирования JS: вроде типов данных, каких-то общих команд, циклов и прочих алгоритмический структур. Возможно, когда-то я напишу про это отдельный конспект, но сделать это можно легко и самому, информации в интернете много. Сейчас буду объяснять только то, что буду использовать я в написании своего бота. Если вы пишите своего бота, вы сможете использовать мой код как основу и изменить значения сообщений на свои.

Начнем с кода, без которого наш бот не будет работать.

const TelegramBot = require('node-telegram-bot-api');

// Создаем постоянную переменную (константу), в которой будет храниться ссылка на функцию из библиотеки node-telegram-bot-api.

const bot = new TelegramBot('3451123866:AAE5ZGv8x7n1Fsf345bZZA32YUGEh5r7bA0', { polling: true });

// Создаем константу, через которую будем обещаться к боту, когда захотим изменить его свойство или что он совершил действие. В кавычках указываем токен, который вам дал @BotFather, вместо моего.

const ADMIN\_ID = '1798265000';

// Создаем константу, в которой будет ваш ID в Телеграме. Узнать его можно в боте @getmyid\_bot написав ему команду «/start». Эта строка не такая уж обязательная, но если вы хотите, чтобы бот присылал вам сообщения от других пользователей, её обязательно стоит добавить.

Рассмотрим, какие основные функции боту доступны.

1. Ответить на команду текстовым сообщением

bot.onText(/\/start/, (msg) => {

    const chatId = msg.chat.id;

    bot.sendMessage(chatId, "Привет!");

  });

// вместо «/start» без кавычек можно написать любую команду. Чтобы ответ был на конкретное сообщение, напишите его текст, например «Привет» между / и /. Если хотите, чтобы ответ был на любое сообщение, напишите «'message'» без кавычек.

//Вторая строка выполняется при условии, что первая выполнилась успешно, так как находится внутри этой функции. Она создает переменную и записывает в нее значение ID персоны, которая вызвала эту команду,

//

npm start

Вся последующая информация сложная и необязательная, однако, вам стоит уделить ей внимание, если хотите шагнуть дальше в неизвестность, и вы заинтересованы в том, чтобы ваш бот был максимально оптимизирован, а процесс разработки – удобный и широкий на выбор инструментов.

**Структурирование и дебаг с Webpack[[3]](#footnote-3)**

С развитием сайтов, количество используемых в них JS-скриптов росло, так же как и число зависимостей между отдельными элементами веб-страниц. Фронтендер должен был уже не просто загрузить файлы в HTML, а сделать это в определённом порядке, учитывая их взаимосвязи. Иначе сайт мог работать неправильно. Для решения этой проблемы был разработан **Webpack** — модульный сборщик, объединяющий JS-скрипты и их зависимости в один файл. Мы его будем использовать для дебага, так как стандартные средства могут сообщить только о факте присутствия ошибки, но не о её характере. Более того, Webpack «помнит» версии кода между запусками, а значит можно посмотреть изменения в «разрезе». На этапе сборок релиза Webpack оптимизирует работу кода и ускорит его обработку.

Как всегда начинаем с установки. В терминале внутри VSCode пишем команду *«npm install webpack webpack-cli --save-dev»* и дожидаемся окончания процесса. Флаг *«--save-dev»* указывает на то, что сборщик будет применяться для разработки промежуточных версий сайта, а не для сборки релиза.

Создайте файл webpack.config.js в корневой папке проекта. Туда будем писать параметры сборки, и подключать при желании плагины. Добавим в этот файл следующий код:

const path = require('path');

module.exports = {

  entry: './index.js',

  output: {

    filename: 'main.js',

    path: path.resolve(\_\_dirname, 'dist')

  }

};

Логика здесь такая – мы определяем, какой файл в проекте у нас главный, откуда будет запускаться наш бот (*«src/web\_project/index.js»*), а также имя выходного файла *(«main.js»)* в который будут собираться все скрипты, перед запуском. Там же указываем путь к папке для сохранения собранного проекта *(«dist»).* Она будет создана автоматически при первом запуске этого кода.

Откроем package.json, находим строчку, в которой написано *«"start": "node index.js",»* и на следующей строке добавим в него команду сборки Webpack, указав через флаг *«--config»* наш конфигурационный файл *«webpack.config.js»*.

"build": "webpack --config webpack.config.js",

Теперь у нас для терминала есть новая команда «npm build», если написать её, запустится Webpack и, если есть ошибки в коде, он их покажет, опишет и предложить автоматическое решение. Кстати, чтобы каждый раз не запускать терминал, чтобы ввести команду «npm build» или «npm start» вручную, нажмите на скрытый список действий над деревом каталога напротив надписи «EXPLORER», и включите «NPM SCRIPTS».

Для расширения функциональности Webpack можно использовать загрузчики — loaders. Они добавляют возможность работать с кодом, написанным на других языках программирования, и файлами разных типов. Плагины Webpack используются для эффективного управления зависимостями, менеджмента модулей и оптимизации кода. Они ускоряют разработку и повышают её удобство. Изучить подборку из плагинов можно в источнике, пока – я не установил ни один из них.

**Автозапуск бота и Task runners[[4]](#footnote-4)**

Так как мой бот стоит на простом стационарном компьютере, любое его выключение означает – что обратное включение обязательно будет сопровождаться ручным запуском Node.js и выполнением команды запуска. Чтобы на этот случай бот включался самостоятельно нужно сделать несколько действий:

Через Поиск или меню Пуск открываем «Планировщик задач». В библиотеке планировщика создадим папку «tg\_bot» в ней создадим простую задачу «Autorun\_Node.js», в качестве триггера выбираем «При входе в систему», а действием выбираем «Запуск программы» и указываем путь к исполнительному файлу Node.js. В параметрах убираем галочку из пункта «Остановить задачу, выполняемую дольше…», все остальные настройки оставляем как есть и нажимаем клавишу «ОК». Думаю, как вы уже поняли, после этих действий Node.js будет сам открываться, когда вы будете запускать компьютер.

Настала очередь сделать выполнение NMP команды для запуска автоматическим, а для этого нам понадобится новый инструментарий.

**Task runners** — это инструменты автоматизации, которые используют, чтобы выполнять часто повторяющиеся задач. Они работают на основе предварительно настроенных сценариев – **тасков**, которые запускаются в определённой последовательности. Эти инструменты ускоряют и упрощают процесс разработки и минимизируют возможность человеческой ошибки.

Task runners выполняют рутинные задачи, такие как минификация CSS и JavaScript файлов, компиляция SASS или LESS в CSS, оптимизация изображений и перезагрузка браузера в реальном времени. Это позволяет разработчику сфокусироваться на творческих аспектах работы и сократить время, которое он тратит на монотонные операции.

Самые популярные инструменты автоматизации — это **Gulp** и **Grunt**, эти task runners автоматизируют практически любых задачи. Gulp имеет стриминговый подход, который облегчает чтение и запись файлов. Grunt предлагает более обширную экосистему плагинов и преднастроенных задач, что делает его идеальным выбором для проектов с большим количеством разнородных задач.

Чтобы установить Gulp, нужно написать в терминале команду «*npm install --global gulp-cli*» и дождаться окончания глобальной установки. Теперь надо добавить его как зависимость в файл «package.json» вашего проекта через npm командой «*npm install gulp --save-dev*».

C помощью расширения Task Runner (скачать его можно внутри VScode) вы можете запускать команды одним щелчком мыши на боковой панели, там же, где вы нашли расширение «NPM SCRIPTS». Расширение автоматически обнаруживает задачи, созданные в файлах конфигурации Gulp и Grunt, а также любые скрипты, доступные вам.

Теперь, когда мы всё установили, создадим внутри корневой папки проекта файл «tasks.json», где будем хранить нашу задачу по запуску NMP команды:

{

    "version": "0.0.1",

    "tasks": [

        {

            "label": "Build\_bot",

            "type": "shell",

            "command": "npm run webpack",

            "windows": {

              "command": "npm run webpack"

            },

            "presentation": {

              "reveal": "always",

              "panel": "new"

            },

            "runOptions": { "runOn": "folderOpen" }

        },

        {

            "label": "Start\_bot",

            "type": "shell",

            "command": "npm run",

            "args": [

                "start"

            ]

        }

    ]

}

Да, я сразу добавил и автозапуск Webpack, чтобы не забывать его включать каждый раз, когда открываю проект.

**Загрузка проекта на Github**

После сохранения и перед запуском бота, необходимо загрузить файлы проекта на GitHub. Вводите эти команды в консоль последовательно. Добавляем все изменения в текущей директории к следующему коммиту (фиксация изменения репозитория) командой «git add .» в терминале компьютера. Создаем коммит с сообщением «first commit», фиксируя все изменения командой «git commit -m "first commit"» и отправляем изменения на GitHub «git push». Проверяем.

1. Основная информация взята с официального ресурса <https://core.telegram.org/bots>. Документ является конспектом и не претендует на оригинальность. [↑](#footnote-ref-1)
2. Информация взята со статьи <https://timeweb.cloud/tutorials/nodejs/kak-sozdat-telegram-bota-na-node-js?ysclid=m5yo490cd3917944905>. Документ является конспектом и не претендует на оригинальность. [↑](#footnote-ref-2)
3. Информация взята со статьи [Основы Webpack: подробный гайд для новичков / Skillbox Media](https://skillbox.ru/media/code/osnovy-webpack-samyy-podrobnyy-gayd/?ysclid=m72qa1ir4a165730316). Документ является конспектом и не претендует на оригинальность. [↑](#footnote-ref-3)
4. Информация взята со статьи [Task Runners и оптимизация. Как ускорить frontend-разработку | Университет frontend-разработки](https://result.school/roadmap/frontend/article/task-runners-i-optimizaciya-kak-uskorit-frontend-razrabotku). Документ является конспектом и не претендует на оригинальность. [↑](#footnote-ref-4)