Бот для чайников

daria-tsareva.com · 4 апреля 2026 г.

Пошаговый гайд: от нуля до рабочего Telegram-бота

Привет! Я собрала этот гайд после того, как сама прошла путь от "а как вообще запустить бота" до рабочих систем в продакшене — HR-бот на 700+ сотрудников, бот для подарков на дни рождения, автоматизация Telegram → Asana. Всё собирала руками и Cursor'ом.

Этот гайд — для тех, кто не программист, но хочет сделать бота сам. Пройдём от регистрации аккаунтов до рабочего бота за один вечер. А дальше — разберёмся, как сделать его по-настоящему полезным.

Что получится в итоге: работающий Telegram-бот, который отвечает на сообщения, показывает кнопки, подключён к базе данных и развёрнут на сервере.

Сколько времени: 1–2 часа на базового бота, ещё пару часов на кнопки, базу и деплой.

Что понадобится

Зарегистрируй аккаунты:

- **GitHub** (github.com) — хранилище кода
- **Vercel** (vercel.com) — по желанию, **бесплатный** слой для веб-страниц и API рядом с ботом; у каждого деплоя свой URL (подробнее — шаг 7)
- **Railway** (railway.app) — когда понадобится, чтобы бот работал **24/7 как процесс** (платно по факту использования; шаг 7)
- **Supabase** (supabase.com) — база данных, понадобится позже

Установи:

- Cursor — редактор кода со встроенным ИИ. Залогинься через GitHub.

Python или TypeScript — что выбрать

В гайде ниже — пример на **Python**, но ветка с **TypeScript** такая же рабочая. Коротко, по сути (без ссылок на доки — их найдёшь, когда понадобится):

Критерий	Python	TypeScript (Node.js)
Чем писать бота	Библиотека **python-telegram-bot** — много примеров в интернете, проще искать «как сделать X» на русском и английском.	Фреймворк **grammY** — привычный стек, если уже трогала сайты на TypeScript или хочешь один язык и для бота, и для веб-части.
Когда удобнее	Первый раз в коде, хочется максимум пошаговых туториалов.	Уже есть опыт с JS/TS или планируешь админку на том же стеке, что и сайт.

Оба варианта нормальны. Если сомневаешься — бери тот язык, на котором тебе спокойнее читать чужой код; Cursor подскажет синтаксис.

Часть 1: Привет, мир!

Цель — бот, который отвечает на любое сообщение. Примерно 30–60 минут.

Шаг 1. Создать бота в Telegram

1. В Telegram найди **@BotFather**
2. Напиши `/start`, потом `/newbot`
3. Назови бота — Name (отображаемое имя) и Username (уникальный, заканчивается на `bot`)
4. Скопируй токен вида `123456:ABC-DEF…`

Токен — это ключ от твоего бота. Кто его знает — тот управляет ботом. Никогда не вставляй токен прямо в код и не отправляй его в чат. Через минуту настроим безопасное хранение.

Шаг 2. Создать репозиторий на GitHub

1. На GitHub: **New repository** → дай имя (например, `my-telegram-bot`), выбери Public или Private
2. В Cursor: **Open from GitHub** → выбери этот репозиторий

Шаг 3. Настроить проект в Cursor

Вот какая структура файлов должна получиться:

```
my-telegram-bot/
├── .env              ← секреты (токен бота)
├── .gitignore        ← чтобы секреты не попали на GitHub
├── main.py           ← код бота
├── requirements.txt  ← список зависимостей
└── README.md         ← описание проекта
```

В Cursor открой ИИ-чат и напиши:

> «Создай минимальный Telegram-бот на python-telegram-bot (polling mode), который отвечает "Привет!" на любое сообщение. Токен бери из переменной окружения TELEGRAM_BOT_TOKEN. Создай requirements.txt, .env с плейсхолдером для токена и .gitignore.»

Cursor сгенерирует все файлы. Проверь три вещи:

- В `.env` — подставь свой настоящий токен
- В `.gitignore` — есть строка `.env`
- В коде — нигде нет токена напрямую, только `os.getenv("TELEGRAM_BOT_TOKEN")`

Шаг 4. Защити свои секреты

Это критически важный шаг, который многие пропускают.

> **Важно:** токен в открытом репозитории — это доступ к боту для любого, кто его увидел. GitHub умеет подсвечивать утечки, но рассчитывать на это не стоит: лучше не коммитить `.env` вообще.

Убедись, что в `.gitignore` есть:

```
.env
__pycache__/
*.pyc
```

Если уже случайно закоммитил `.env` — недостаточно просто удалить файл. Нужно сменить токен в @BotFather командой `/revoke`. Старый токен навсегда остаётся в истории Git.

Шаг 5. Запустить бота локально

Открой встроенный терминал в Cursor. Попроси ИИ:

> «Напиши команды, чтобы установить зависимости и запустить бота локально.»

Обычно это:

```
pip install -r requirements.txt
python main.py
```

Если всё ок — напиши что-нибудь своему боту в Telegram. Он должен ответить.

Как бот вообще узнаёт о новых сообщениях (без терминов из учебников)

Здесь мы делаем так: программа **сама по кругу спрашивает** у Telegram: «есть что-то новое?». Тебе **не нужен** адрес сайта в интернете — поэтому так проще начать и тестировать у себя на компьютере. Когда выложишь бота на хостинг, который умеет держать программу

включённой всё время (см. шаг 7, Railway), обычно можно оставить тот же способ — ничего заново не придумывать.

Если же захочешь **именно Vercel** (бесплатный слой «по запросу»), там нельзя оставить программу «крутящейся вечно». Тогда путь другой: **Telegram сам присылает** событие на твою ссылку в интернете — в чатах с Cursor это часто называют *webhook*. Попроси: «сделай бота под Vercel: Telegram стучится по моему URL, без вечного цикла опроса». Это уже отдельная настройка, не тот же код, что в самом начале гайда.

> **Принцип:** первая версия может быть совсем простой — один ответ на всё. Это нормально. Сначала добейся, чтобы ничего не падало, потом наращивай сценарии.

У тебя работающий бот — это уже серьёзно.

Часть 2: Делаем полезного бота

Шаг 6. Добавить реальную логику

Бот отвечает "Привет" — пора научить его делать что-то полезное. В Cursor формулируй задачу так, как объяснял бы коллеге:

> «Хочу, чтобы бот показывал меню из трёх кнопок: "FAQ", "Контакты", "Обратная связь". При нажатии на FAQ — показывал список частых вопросов.»

Cursor найдёт нужные файлы и предложит изменения (зелёные/красные строки). Читай summary, жми **Apply**, если всё ок.

Рабочий цикл (его стоит запомнить):

1. Формулируешь задачу в чате Cursor
2. Cursor предлагает изменения
3. Проверяешь и применяешь
4. Тестируешь в Telegram
5. Коммитишь: «Сделай commit и push с сообщением "Добавлено меню с кнопками"»

Cursor сам выполнит `git add`, `git commit`, `git push`.

> **На будущее:** меню со временем разрастается. Удобно сразу попросить Cursor держать тексты кнопок и ответы в одном месте (отдельный файл или структура данных), чтобы не плодить копипасту в коде.

Шаг 7. Развернуть бота на сервере

Пока бот работает только на твоём компьютере. Закроешь Cursor — бот замолчит. Чтобы он работал 24/7, нужен хостинг.

Многие хотят начать с **чего-то полностью бесплатного** — это логично. Ниже — два разных сценария: сначала про **бесплатный** вариант и его границы, потом про **платный** хостинг, когда бот должен крутиться как маленький сервер.

Вариант А: Vercel (бесплатный тариф)

Vercel заточен под **сайты и куски кода, которые запускаются, когда кто-то открыл страницу или дернул API**. После ответа всё может «уснуть» — **нельзя** оставить программу, которая круглосуточно сама опрашивает Telegram (тот способ, с которого мы начали в гайде).

- **Что сюда хорошо ложится:** лендинг, простая админ-страница, API для отчётов. У **каждого деплоя свой адрес** — удобно сравнивать версии.
- **Если хочешь, чтобы сам бот жил на Vercel:** нужна схема, где **Telegram приходит к тебе по ссылке** (см. блок выше про «webhook») — это другой код и другая настройка, не копипаста из шагов 3–5. После простоя первый ответ иногда приходит с паузой; очень долгие ответы (например, тяжёлый AI) упираются в лимит времени на один запуск — длинную генерацию обычно выносят в фон.

Итого: Vercel — сильный **бесплатный** слой для веб-части рядом с ботом. Сценарий «бот как в начале гайда» без переделок — скорее не сюда.

Вариант Б: Railway (платный по сути, зато «как сервер»)

Здесь поднимаешь сервис, который **работает всё время** — как компьютер в облаке. Тот способ, с которого мы начали (бот сам спрашивает Telegram по кругу), **сюда переносится** без смены логики.

1. Зайди на railway.app, залогинься через GitHub
2. **New Project** → **Deploy from GitHub repo** → выбери репозиторий с ботом
3. Если нужно — укажи **Start Command** (спроси Cursor: «Дай команду запуска для Railway для моего `main.py`»)
4. В **Variables** добавь `TELEGRAM_BOT_TOKEN` = твой токен (не коммить в репозиторий)
5. Дождись деплоя и проверь бота в Telegram

Про деньги: у Railway обычно есть стартовый кредит; дальше списание по факту использования — смотри **Usage** в дашборде. Маленький бот часто укладывается в пару–несколько долларов в месяц, но это уже не «вечно ноль», а плата за нормальный always-on.

> **Частая ошибка:** бот работал локально, а на сервере — нет. Чаще всего забыли `TELEGRAM_BOT_TOKEN` в **Variables** или в коде остался хардкод вместо `os.getenv()`. Логи: **Railway → сервис → Logs** (или у конкретного деплоя — **View logs**).

Шаг 8. Подключить базу данных

Когда это нужно: если бот должен что-то запоминать между сообщениями.

Примеры:

- Запоминать, что пользователь уже сделал в сценарии (шаг диалога, статус заявки)
- Хранить список сотрудников и их данные
- Вести историю обращений
- Отправлять отложенные сообщения (напоминания, поздравления)

Если бот просто отвечает на вопросы из фиксированного списка — база не нужна, не усложняй.

Подключение Supabase:

1. Создай проект на supabase.com
2. Создай таблицу (например, `users` с полями `telegram_chat_id`, `name`, `created_at`)
3. Скопируй URL и ключи проекта
4. Добавь их в `.env` (и проверь, что `.env` в `.gitignore`!)

В Cursor:

> «Подключи бота к Supabase. URL и ключ в .env. Сделай функции для записи и чтения пользователей из таблицы users.»

> **На что обратить внимание:** таблицы и поля, которые задала в начале, потом не всегда удобно менять — когда данных много, миграции и правки связей отнимают время. Если схема кажется «на вырост», имеет смысл хотя бы набросать её на бумаге или в Notion до кодинга: что храним, что уникально, что может повторяться.

Часть 3: Что дальше

Ты прошёл путь от нуля до рабочего бота с кнопками, базой данных и деплоем. Это серьёзный результат.

Но между "бот работает у меня" и "бот работает в команде на 50+ человек" — большая дистанция. Вот что нужно реальным production-ботам:

Надёжность

- Обработка ошибок — что если Supabase недоступен? что если пользователь прислал стикер вместо текста?
- Логирование — понимать, что пошло не так, без гадания
- Мониторинг — узнавать о проблемах раньше пользователей

Масштаб

- Многошаговые сценарии — бот ведёт диалог, помнит контекст
- Роли — пользователь видит одно, админ — другое
- Нагрузка — когда 500 человек пишут одновременно

Безопасность

- Защита от спама и злоупотреблений
- Ограничение доступа — только сотрудники компании
- Безопасное хранение персональных данных

Интеграции

- Связка с CRM, ATS, Google Sheets, Notion
- Автоматические отчёты и дашборды
- Webhook-цепочки между сервисами

Всё это решаемо, но обычно не за один вечер — появляются очереди, ретраи, отдельные сервисы под тяжёлые задачи.

Шпаргалка: как говорить с Cursor

Описывай что хочешь, а не какой код написать

- «Добавь кнопку "Отпуск" — бот должен спросить даты и записать в таблицу vacations»
- «Сделай HTML-страницу, где я как HR вижу список обращений в бота за неделю»
- «Обнови логику: напоминание не за 10 дней до ДР, а раз в месяц — список именинников»

Проси разбить на шаги

- «Сделай пошаговый план: что изменить, в каких файлах, какие команды выполнить»
- «Какие файлы изменятся, если я попрошу добавить X?»

Когда ломается

- «Вот логи с Railway *(вставь логи)*. Найди причину и исправь код.»
- «Бот перестал отвечать после последних изменений. Что могло сломаться?»
- «Откати последнее изменение и объясни, что пошло не так.»

Для понимания

- «Объясни этот код простыми словами — что он делает и зачем»
- «Я не понимаю эту ошибку *(вставь ошибку)* — объясни, что не так и как починить»

Для качества

- «Добавь обработку ошибок — что если пользователь отправит не то, что ожидается?»
- «Этот файл слишком длинный — разбей на короткие куски по смыслу и убери повторяющийся код»
- «Проверь, нет ли в коде проблем с безопасностью»

Когда имеет смысл позвать человека

Гайд закрывает старт. Дальше часто всплывают задачи, где один Cursor не заменит контекст компании:

- бот внезапно перестал отвечать, а по логам непонятно;
- нужно стыковать Telegram с CRM, ATS, таблицами, внутренними API;
- бот не для себя, а для команды — роли, доступы, админка;
- многошаговые сценарии и хранение состояния между сообщениями;
- процесс под конкретный регламент (онбординг, опросы, FAQ по политикам).

Если хочется разобрать свой кейс голосом или в переписке — Telegram @dadariat.