Урок 2. GitHub: секреты, тесты и деплой на практике

О чём этот урок

В первом уроке мы познакомились с основами работы в GitHub: создали репозиторий, сделали первые коммиты и отправили код. Сегодня — практика на реальном проекте `work-list-planner`: устранили уязвимость с секретным ключом, добавили автотесты и оформили деплой. И главное — всё это зафиксировали и проверили именно через GitHub.

1. Проблема: секрет в публичном репозитории

При проверке репозитория `work-list-planner` на GitHub мы обнаружили, что в файле `main.py` жёстко прописан `SECRET_KEY` — ключ для подписи JWT-токенов. Этот ключ был в коде и уехал в публичный репозиторий.

Почему это критично. Любой человек может открыть репозиторий на GitHub и увидеть этот ключ. Даже если позже удалить строку из кода, она навсегда останется в истории коммитов. Восстановить или «стереть» её из GitHub нельзя.

Правило: секреты никогда не должны попадать в репозиторий GitHub. Для них используют переменные окружения на сервере.

2. Решение: выносим секрет из GitHub

Шаг 1. Меняем код, чтобы он брал ключ из переменной

В `main.py` заменили жёсткое значение на чтение из переменной окружения:

python

```
SECRET_KEY = os.environ.get("SECRET_KEY", "")
if not SECRET_KEY:
    raise RuntimeError("SECRET_KEY is not set. Check .env / systemd EnvironmentFile")
```

Так код перестаёт хранить секрет внутри себя.

Шаг 2. Защищаем `.env` от попадания в GitHub

На сервере создали файл `.env` с новым ключом и сразу добавили его в `.gitignore`:

bash

```
echo "SECRET_KEY=<новый-ключ>" > .env
chmod 600 .env
```

В `.gitignore` добавили строку:

text

```
.env
```

Теперь GitHub никогда не предложит закоммитить этот файл. Проверить можно через `git status`: `.env` не должен появляться в списке файлов для коммита.

Шаг 3. Фиксируем изменения в GitHub

Коммитим и отправляем изменения:

bash

```
git add main.py .gitignore
git commit -m "Move SECRET_KEY to environment variable, add .env to gitignore"
git push
```

После этого в репозитории на GitHub в файле `main.py` уже нет старого ключа — только логика чтения из переменной.

Шаг 4. Проверяем историю коммитов на GitHub

Чтобы убедиться, что больше нигде нет секретов, можно либо использовать терминал (как раньше), либо посмотреть историю прямо на GitHub:

1. На странице репозитория нажмите **Commits**.
2. В списке коммитов найдите тот, где меняли `SECRET_KEY`.
3. Кликните на хеш коммита — откроется страница с диффами (изменениями).
4. Просмотрите изменения в `main.py`: старый ключ должен быть зачёркнут (удалена строка), а новая логика — добавлена.

> **Важно:** старый ключ всё ещё виден в истории, но он больше не используется. Новый ключ нигде не хранится в репозитории — только на сервере в `.env`.

---

3. Тесты: добавляем страховку и фиксируем в репозитории

Подготовка: изолируем тестовую базу

Чтобы тесты не трогали реальную базу данных, путь к ней тоже вынесли в переменную окружения:

python

```
DB_PATH = os.environ.get("DB_PATH", os.path.join(os.path.dirname(__file__), "planner.db"))
```

Это позволяет тестам использовать временную базу, а продакшену — обычную.

Пишем тесты и добавляем их в репозиторий

Тесты покрывают: регистрацию, логин (успешный и неудачный), защиту эндпоинтов и полный CRUD задач. Тесты хранятся в папке `tests/`.

Для корректной работы тестов в корне проекта добавили `conftest.py`, чтобы pytest мог находить модуль `main`.

После написания тестов их добавили в репозиторий:

bash

```
git add conftest.py tests/ main.py
git commit -m "Add API tests: auth and task CRUD"
git push
```

Как смотреть тесты на GitHub

1. В репозитории откройте папку `tests/` — там будет файл `test_api.py` со всеми тестами.
2. На странице любого коммита можно посмотреть диффы: какие строки добавлены (зелёные) и удалены (красные).
3. Если нужно увидеть только изменения по тестам, используйте фильтр или просто откройте историю файла `tests/test_api.py`.

Результат запуска тестов (`6 passed`) — это локальная проверка. Но сам факт наличия тестов и их код теперь навсегда зафиксирован в репозитории GitHub. Это и есть главная страховка: любой разработчик, который откроет проект, сразу увидит, что API покрыт тестами.

---

4. Деплой: оформляем как артефакт репозитория

Скрипт `deploy.sh` делает обновление сайта одной командой: подтягивает код из GitHub, перезапускает сервис и проверяет, что сайт отвечает.

Содержимое `deploy.sh`:

bash

```
#!/bin/bash
set -e

APP_DIR=/var/www/planner-app
SERVICE=planner.service

cd "$APP_DIR"
git pull
systemctl restart "$SERVICE"
sleep 2
HTTP_CODE=$(curl -s -o /dev/null -w "%{http_code}" http://127.0.0.1:8000/)
if [ "$HTTP_CODE" = "200" ]; then
    echo "OK: сайт отвечает (HTTP $HTTP_CODE)"
else
    echo "ОШИБКА: сайт ответил $HTTP_CODE — смотри логи:"
    echo "journalctl -u $SERVICE -n 20 --no-pager"
```

```
    exit 1
fi
```

Добавляем скрипт в репозиторий:

bash

```
chmod +x deploy.sh
git add deploy.sh
git commit -m "Add deploy script"
git push
```

Как это выглядит в GitHub:

- В корне репозитория появляется файл `deploy.sh`.
- В коммитах видно, когда и кем он был добавлен.
- Любой человек, который клонирует репозиторий или смотрит его на GitHub, сразу видит: есть готовый скрипт для обновления сайта.

---

5. Обновляем README и статус проекта

В `README.md` обновили раздел «Статус», чтобы он отражал текущее состояние:

> MVP-версия учебного проекта, развёрнута на VPS (work-list.ru). Секреты вынесены в переменные окружения, API покрыт автотестами, деплой — одной командой.

Также добавили разделы про тесты и деплой:

markdown

```
## Тесты
Запуск: `pytest tests/ -v` — покрытие авторизации и CRUD задач.
```

```
## Деплой
На сервере: `./deploy.sh` — подтягивает код, перезапускает сервис и проверяет, что сайт отвечает.
```

Коммит:

bash

```
git add README.md
git commit -m "Update project status in README"
git push
```

Зачем это в GitHub. README — это первое, что видит человек, открывший репозиторий. Теперь там честно написано, что проект не просто «работает», а имеет тесты и безопасный деплой. Это сильно повышает ценность проекта для портфолио.

Выводы урока

- **GitHub — это публичное место.** Всё, что попадает в репозиторий, считается общедоступным. Секреты туда не должны попадать никогда.
- **История коммитов в GitHub — навсегда.** Если секрет попал в репозиторий, он остаётся в истории. Единственный безопасный путь — заменить ключ и вынести его из кода.
- **Тесты — часть проекта.** Код тестов хранится в репозитории и показывает другим разработчикам, что функциональность проверена.
- **Деплой-скрипт — тоже артефакт.** Он живёт в репозитории как обычный файл и делает обновление предсказуемым.
- **README — лицо проекта.** Обновляя статус и добавляя описание тестов и деплоя, мы делаем проект понятным и профессиональным.

Что дальше (на следующий раз)

- Подробно разберём, как смотреть историю изменений именно через интерфейс GitHub: вкладки Commits, History у файлов, Compare для сравнения веток.
- Настроим GitHub Actions, чтобы тесты запускались автоматически при каждом пуше.
- Добавим в репозиторий дополнительные меры безопасности: сузим CORS и добавим защиту от брутфорса.

Revision #2
Created 2026-10-07 22:58:00 UTC by im
Updated 2026-10-07 22:58:40 UTC by im