# Часть 2: Настройка GitHub Actions и GitLab CI – первый workflow и деплой

> В первой статье мы разобрались с основами CI/CD: что это такое, зачем нужно и какие инструменты существуют. Теперь пришло время перейти от теории к практике – создадим наши первые рабочие CI/CD-конвейеры на GitHub Actions и GitLab CI.

> Вот небольшая карта для навигации между частями:
>
> **Часть 1: Основы CI/CD – что это и зачем нужно; GitHub Actions и GitLab CI**
>
> **Часть 2: Настройка GitHub Actions и GitLab CI – первый workflow и деплой**
>
> **^ Вы сейчас здесь ^**
>
> **Часть3: CI/CD – ветки, условия, секреты и окружения**

## Введение

Помните, как в первой статье мы говорили о CI/CD как о вашем личном роботе-помощнике? Сегодня мы этого робота соберём и запрограммируем. Мы настроим репозитории на GitHub и GitLab, напишем первые CI/CD-скрипты и проверим их работу.

Не переживайте, если вы никогда раньше не сталкивались с DevOps-инструментами. Все шаги будут описаны настолько подробно, что справится даже новичок. Нам не потребуется глубокое образование или многолетний опыт – только желание автоматизировать рутину и немного терпения.

Даже с нуля вы справитесь, по порядку распишем каждое действие. Это как собирать конструктор по инструкции – сначала может показаться сложным, но когда всё встанет на свои места, вы удивитесь, насколько это просто.

Наш первый workflow будет таким же простым, как программа "Hello, World" – но это уже настоящий шаг вперёд. Вы же помните свою первую программу? Такая простая, но какую гордость вы испытали, когда она заработала. Сегодня вы испытаете то же самое, только с автоматизацией.

Итак, приступим.

---

## Создание репозитория

Прежде чем настраивать CI/CD, нам нужно место, где будет храниться наш код и конфигурации. Таким местом станет репозиторий на GitHub или GitLab. Если у вас уже есть аккаунт и репозиторий – отлично! Если нет – давайте создадим.

# На GitHub:

1. Войдите в свой аккаунт на GitHub.
2. В правом верхнем углу нажмите на "+" и выберите "New repository".
3. Дайте репозиторию понятное имя, например, "my-first-cicd".
4. Поставьте галочку "Initialize this repository with a README" – это создаст начальный файл README.md.
5. По желанию можете добавить LICENSE (лицензию) и .gitignore.
6. Нажмите "Create repository".

# На GitLab:

1. Войдите в свой аккаунт на GitLab.
2. Нажмите на кнопку "New project" (или "+" в верхнем меню).
3. Выберите "Create blank project".
4. Укажите имя проекта, например, "my-first-cicd".
5. Поставьте галочку "Initialize repository with a README".
6. Нажмите "Create project".

Что такое репозиторий? Это не просто хранилище кода – это ваша мастерская, где будут храниться не только исходники, но и инструкции для нашего "робота" (CI/CD-конфигурации). Представьте репозиторий как рабочий стол мастера: здесь лежат и материалы, и чертежи, и инструменты.

По умолчанию в репозитории создаётся ветка `main` (или `master` в более старых репозиториях). Большинство примеров в этой статье будет привязано именно к этой ветке. Вы можете использовать её по умолчанию для наших экспериментов.

**Совет:** Если вы предпочитаете работать в cli, можно создать репозиторий вот таким способом:

```
# Создаём папку и инициализируем Git
mkdir my-first-cicd
cd my-first-cicd
git init
echo "# My First CI/CD Project" &gt; README.md
git add README.md
git commit -m "Initial commit"

# Связываем с удалённым репозиторием (замените URL на свой)
git remote add origin https://github.com/username/my-first-cicd.git
git push -u origin main
```

Отлично! Теперь у нас есть место, где будет жить наш код и CI/CD-конфигурации. Переходим к самому интересному – настройке автоматизации.

---

## Первый workflow на GitHub Actions

GitHub Actions – это встроенный в GitHub инструмент для автоматизации. Он позволяет запускать различные задачи при определённых событиях в репозитории. Настройка происходит через YAML-файлы, которые хранятся в специальной папке.

Давайте создадим наш первый workflow:

1. В вашем репозитории создайте папку `.github/workflows/`. Для этого можно использовать веб-интерфейс GitHub:
   - Нажмите "Add file" > "Create new file"
   - В поле имени введите `.github/workflows/ci.yml` (GitHub автоматически создаст папки)
2. В этот файл добавьте следующий код:

```
name: CI for project

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
```

```
  hello-world:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Say Hello
      run: echo "Hello, GitHub Actions! This is my first workflow!"

    - name: Show date
      run: date
```

3. Нажмите "Commit new file" внизу страницы.

Что мы только что сделали? Давайте разберём этот файл по частям:

- `name: CI for project` – это название нашего workflow, которое будет отображаться в интерфейсе GitHub.
- `on: push: branches: [ main ]` – здесь мы указываем, когда запускать workflow. В данном случае – при каждом push в ветку `main` или при создании pull request в эту ветку.
- `jobs:` – раздел, где описываются задачи, которые нужно выполнить.
- `hello-world:` – название нашего job-а (можно придумать любое).
- `runs-on: ubuntu-latest` – указывает, на какой операционной системе запускать задачу. GitHub предоставляет виртуальные машины с разными ОС.
- `steps:` – последовательность шагов, которые нужно выполнить в рамках job-а.
- `uses: actions/checkout@v3` – этот шаг клонирует ваш репозиторий в виртуальную машину, чтобы последующие шаги могли работать с вашим кодом.
- `run: echo "Hello, GitHub Actions!"` – простая команда, которая выводит текст в консоль.

После того как вы закоммитите этот файл, GitHub автоматически обнаружит его и запустит workflow. Чтобы увидеть результаты:

1. Перейдите на вкладку "Actions" в вашем репозитории.
2. Вы увидите запущенный workflow с названием "CI for project".
3. Кликните на него, чтобы увидеть детали выполнения.
4. Нажмите на job "hello-world", чтобы увидеть вывод каждого шага.

Поздравляю! Вы только что создали и запустили свой первый CI/CD-workflow. Он пока очень простой, но это уже настоящая автоматизация – код запускается автоматически при изменении репозитория.

**Важно:** YAML очень чувствителен к отступам. Если workflow не запускается или выдаёт ошибку, первым делом проверьте, правильно ли расставлены пробелы в начале строк. В YAML используются именно **пробелы**, а не табуляции.

---

## Первый pipeline на GitLab CI

GitLab CI работает похожим образом, но имеет некоторые отличия в синтаксисе и организации. Давайте создадим наш первый pipeline на GitLab:

1. В вашем GitLab-репозитории создайте файл `.gitlab-ci.yml` в корне проекта:
   - Нажмите "+" > "New file"
   - Назовите файл `.gitlab-ci.yml`
2. Добавьте в файл следующий код:

```
stages:
  - test
  - deploy

hello_job:
  stage: test
  image: alpine:latest
  script:
    - echo "Hello, GitLab CI! This is my first pipeline!"
    - date

# Пока закомментируем этап деплоя, мы вернёмся к нему позже
#deploy_job:
#  stage: deploy
#  script:
#    - echo "This is where we would deploy our application"
#  only:
#    - main
```

3. Нажмите "Commit changes".

Разберём, что мы написали:

- `stages:` – здесь мы определяем этапы выполнения pipeline. Они будут выполняться последовательно: сначала все job-ы из этапа `test`, затем из `deploy`.
- `hello_job:` – название нашего job-а.
- `stage: test` – указывает, к какому этапу относится этот job.
- `image: alpine:latest` – Docker-образ, в котором будет выполняться job. Alpine – это лёгкий Linux-дистрибутив.

- `script:` – команды, которые нужно выполнить в рамках job-а.

После коммита GitLab автоматически обнаружит файл `.gitlab-ci.yml` и запустит pipeline. Чтобы увидеть результаты:

1. Перейдите в раздел "CI/CD" > "Pipelines" в вашем проекте.
2. Вы увидите запущенный pipeline.
3. Кликните на него, чтобы увидеть детали выполнения.
4. Нажмите на job "hello_job", чтобы увидеть вывод каждой команды.

Отлично! Теперь у вас есть работающие CI/CD-конфигурации и на GitHub, и на GitLab. Давайте сделаем что-то более полезное – настроим автоматический деплой простого сайта.

---

## Пример: деплой статического сайта

Статические сайты – идеальный первый проект для CI/CD, потому что их легко развернуть и они не требуют сложной инфраструктуры. И GitHub, и GitLab предоставляют бесплатный хостинг для статических сайтов через GitHub Pages и GitLab Pages соответственно.

Давайте создадим простой HTML-файл и настроим его автоматический деплой при каждом изменении.

# Подготовка проекта

1. В корне вашего репозитория создайте файл `index.html`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Мой первый CI/CD проект</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 800px;
            margin: 0 auto;
            padding: 20px;
            line-height: 1.6;
        }
        h1 {
            color: #2c3e50;
        }
        .container {
            border: 1px solid #ddd;
```

```
            padding: 20px;
            border-radius: 5px;
            background-color: #f9f9f9;
        }
    </style>
</head>
<body>
    <h1>Привет, CI/CD!</h1>
    <div class="container">
        <p>Эта страница была автоматически развёрнута с помощью CI/CD.</p>
        <p>Время последнего обновления: <span id="timestamp"></span></p>
    </div>

    <script>
        document.getElementById('timestamp').textContent = new Date().toLocaleString();
    </script>
</body>
</html>
```

# Деплой на GitHub Pages

Обновите ваш файл `.github/workflows/ci.yml`:

```
name: CI/CD for Static Website

on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Deploy to GitHub Pages
      uses: peaceiris/actions-gh-pages@v3
      with:
```

```
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./
```

Этот workflow использует готовый action `peaceiris/actions-gh-pages`, который автоматизирует деплой на GitHub Pages. `secrets.GITHUB_TOKEN` – это токен, который GitHub автоматически создаёт для каждого workflow, так что вам не нужно настраивать его вручную.

После успешного выполнения workflow:

1. Перейдите в настройки вашего репозитория (Settings).
2. Прокрутите вниз до раздела "GitHub Pages".
3. Убедитесь, что в качестве источника выбрана ветка `gh-pages`.
4. GitHub предоставит URL, по которому доступен ваш сайт (обычно это `https://username.github.io/repository-name/`).

# Деплой на GitLab Pages

Обновите ваш файл `.gitlab-ci.yml`:

```
stages:
  - deploy

pages:
  stage: deploy
  script:
    - mkdir -p public
    - cp index.html public/
  artifacts:
    paths:
      - public
  only:
    - main
```

Этот pipeline создаёт папку `public` (это специальное имя для GitLab Pages) и копирует туда наш HTML-файл. Затем он сохраняет эту папку как артефакт, который GitLab автоматически публикует.

После успешного выполнения pipeline:

1. Перейдите в "Settings" > "Pages" вашего проекта.
2. GitLab предоставит URL, по которому доступен ваш сайт (обычно это `https://username.gitlab.io/repository-name/`).

Поздравляю! Теперь у вас есть настоящий CI/CD-конвейер, который автоматически публикует ваш сайт при каждом изменении кода. Это уже не просто "Hello, World!" – это полноценный процесс доставки.

---

## Пример: тестовый скрипт

CI – это не только о деплое, но и о проверке качества кода. Давайте добавим простой тест, который будет проверяться автоматически при каждом изменении.

# Для JavaScript-проекта

1. Создайте файл `app.js`:

```
function sum(a, b) {
    return a + b;
}

module.exports = { sum };
```

2. Создайте файл `test.js`:

```
const { sum } = require('./app');

if (sum(2, 3) !== 5) {
    console.error('Test failed: 2 + 3 should equal 5');
    process.exit(1);
}

if (sum(-1, 1) !== 0) {
    console.error('Test failed: -1 + 1 should equal 0');
    process.exit(1);
}

console.log('All tests passed!');
```

3. Создайте файл `package.json`:

```
{
  "name": "my-first-cicd",
  "version": "1.0.0",
  "scripts": {
```

```
    "test": "node test.js"
  }
}
```

## Для Python-проекта

1. Создайте файл `app.py`:

```
def sum(a, b):
    return a + b
```

2. Создайте файл `test.py`:

```
from app import sum

def test_sum():
    assert sum(2, 3) == 5, "2 + 3 should equal 5"
    assert sum(-1, 1) == 0, "-1 + 1 should equal 0"
    print("All tests passed!")

if __name__ == "__main__":
    test_sum()
```

## Обновление GitHub Actions workflow

Обновите ваш файл `.github/workflows/ci.yml` для JavaScript:

```
name: CI/CD for Project

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
```

```
      uses: actions/checkout@v3

    - name: Setup Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '16'

    - name: Run tests
      run: npm test

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Deploy to GitHub Pages
      uses: peaceiris/actions-gh-pages@v3
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./
```

Или для Python:

```
name: CI/CD for Project

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
```

```
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Setup Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.10'

    - name: Run tests
      run: python test.py

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Deploy to GitHub Pages
      uses: peaceiris/actions-gh-pages@v3
      with:
        github_token: ${{ secrets.GITHUB_TOKEN }}
        publish_dir: ./
```

## Обновление GitLab CI pipeline

Обновите ваш файл `.gitlab-ci.yml` для JavaScript:

```
stages:
  - test
  - deploy

test:
  stage: test
  image: node:16
  script:
    - npm test
```

```
pages:
  stage: deploy
  script:
    - mkdir -p public
    - cp index.html public/
  artifacts:
    paths:
      - public
  only:
    - main
  needs:
    - test
```

Или для Python:

```
stages:
  - test
  - deploy

test:
  stage: test
  image: python:3.10
  script:
    - python test.py

pages:
  stage: deploy
  script:
    - mkdir -p public
    - cp index.html public/
  artifacts:
    paths:
      - public
  only:
    - main
  needs:
    - test
```

Теперь наш CI/CD-процесс стал ещё более полноценным:

1. При каждом изменении кода автоматически запускаются тесты.

2. Если тесты проходят успешно, код автоматически публикуется.
3. Если тесты не проходят, деплой не выполняется – это защищает нас от публикации сломанного кода.

Это базовый, но уже вполне рабочий CI/CD-конвейер. В реальных проектах тесты будут более сложными, но принцип остаётся тем же: автоматическая проверка → автоматический деплой.

---

## Полезные советы и выводы

Теперь, когда у вас есть работающие CI/CD-конфигурации, вот несколько полезных советов:

# 1. Проверяйте статус

После каждого пуша убедитесь, что ваш workflow или pipeline действительно запускается:

- На GitHub: вкладка "Actions"
- На GitLab: "CI/CD" > "Pipelines"

Зелёный статус означает, что всё прошло успешно. Красный – что-то пошло не так.

# 2. Читайте логи

Если что-то не работает, логи – ваш лучший друг. Они покажут, на каком именно шаге произошла ошибка и что пошло не так. Кликните на job, чтобы увидеть подробный вывод каждой команды.

# 3. Быстро правьте

Одно из преимуществ CI/CD – быстрая обратная связь. Если вы видите ошибку, исправьте её и снова отправьте изменения. Через несколько минут вы узнаете, помогло ли исправление.

# 4. Экспериментируйте

Не бойтесь экспериментировать! Специально сломайте тест, чтобы увидеть, как система отреагирует. Добавьте новые шаги в workflow. Чем больше вы экспериментируете, тем лучше понимаете, как всё работает.

# 5. Используйте готовые actions и templates

И GitHub, и GitLab предлагают множество готовых actions и templates для типичных задач. Не изобретайте велосипед – ищите готовые решения в маркетплейсе GitHub Actions или в документации GitLab CI.

# 6. Постепенно усложняйте

Начните с простого, как мы сделали в этой статье, и постепенно добавляйте новые возможности: больше тестов, статический анализ кода, автоматическую сборку более сложных приложений.

Поздравляю, вы только что построили конвейер! Ну почти, но уже уверенно движетесь в правильном направлении. Теперь у вас есть автоматизированный процесс, который проверяет и публикует ваш код без ручного вмешательства. Это уже настоящий DevOps!

---

## Терминология

В мире CI/CD используется множество специфических терминов. Вот краткий словарик, который поможет вам лучше понимать документацию и обсуждения:

- **Workflow** – сценарий автоматизации, описанный в YAML-файле на GitHub Actions.
- **Job** – отдельная задача в workflow или pipeline, которая выполняется на одном runner-е.
- **Runner** – виртуальная машина или контейнер, на котором исполняются задачи.
- **Step** – отдельный шаг внутри job-а, например, команда или action.
- **YAML** – формат файла, используемый для описания workflow и pipeline. Отступы в нём критически важны!
- **Script** – раздел с командами в GitLab CI-файле.
- **Action** – готовое действие на GitHub, которое можно использовать в своих workflow, например, `actions/checkout`.
- **Stage** – этап pipeline в GitLab, например, build, test, deploy.
- **Artifact** – файл или набор файлов, созданный во время выполнения job-а и сохранённый для использования в других job-ах или для скачивания.
- **Environment** – среда, в которую выполняется деплой, например, staging или production.

Не переживайте, если сразу не запомните все термины – они станут понятнее по мере практики.

---

## Итоги

В этой статье мы:

1. Создали репозитории на GitHub и GitLab.
2. Настроили первые workflow и pipeline.
3. Автоматизировали деплой статического сайта.
4. Добавили автоматические тесты.
5. Построили полноценный CI/CD-конвейер, который проверяет и публикует код при каждом изменении.

Теперь при каждом пуше в ветку `main` автоматически выполняются тесты, и если они проходят успешно, сайт обновляется. Команде больше не нужно вручную запускать

проверку – теперь ошибки видны сразу, а деплой происходит автоматически.

Это только начало вашего путешествия в мир CI/CD. В следующей статье мы рассмотрим более сложные сценарии: работу с разными ветками, условное выполнение шагов, использование секретов для безопасного хранения чувствительных данных и лучшие практики настройки CI/CD для реальных проектов.

А пока – экспериментируйте с тем, что вы узнали сегодня. Добавьте больше тестов, попробуйте другие actions, настройте деплой для своего проекта. Практика – лучший способ закрепить знания.

---

Revision #2
Created 2026-03-05 08:15:24 UTC by Эд Кукса
Updated 2026-03-05 09:11:00 UTC by Эд Кукса