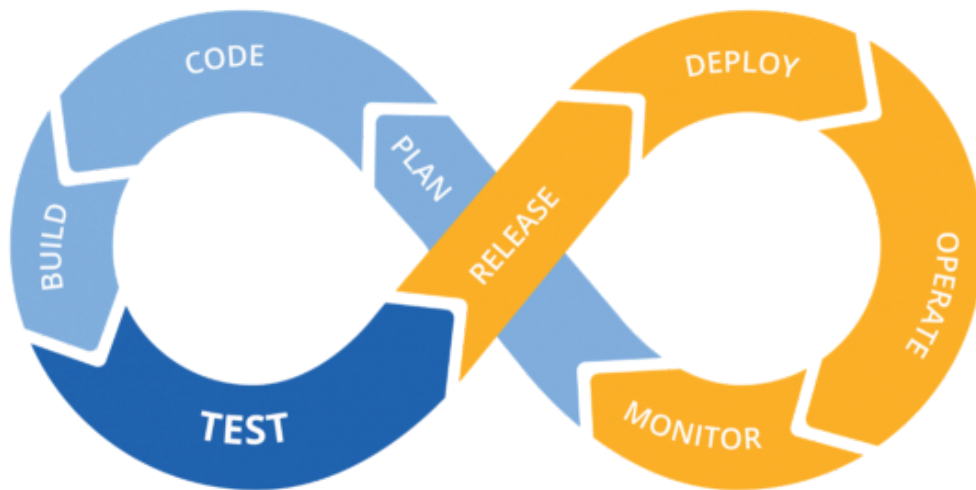


CI/CD

# Основы CI/CD. Знакомство с GitLab

---





## На этом уроке

1. Узнаем, что представляет собой CI/CD. Разберём основные термины.
2. Посмотрим, какие инструменты используются в CI/CD.
3. Познакомимся с архитектурой и настройкой GitLab.
4. Напишем первый gitlab\_ci.

## Оглавление

[Введение](#)

[Основные принципы управления релизами](#)

[Что такое CI/CD](#)

[Continuous Integration](#)

[Continuous Delivery и Continuous Deployment](#)

[Знакомство с GitLab](#)

[Как работает GitLab CI/CD](#)

[Архитектура GitLab](#)

[Gitaly](#)

[GitLab Pages](#)

[GitLab Shell](#)

[PostgreSQL](#)

[Registry](#)

[Pipeline](#)

[Переменные](#)

[GitLab Pages](#)

[Pipeline для GitLab Pages](#)

[Доступ к GitLab Pages](#)

[Страницы с ошибками](#)

[Практическое задание](#)

[Глоссарий](#)

[Дополнительные материалы](#)

[Используемые источники](#)

## Введение

**Управление выпуском новых версий или релизами (release engineering)** — это относительно новое и быстро развивающееся направление в IT, которое может быть кратко определено как разработка и доставка программного обеспечения.

**Релиз-инженеры или SRE (Site Reliability Engineer)** — это ребята, которые определяют все шаги, необходимые для выпуска ПО: от способа хранения исходных текстов в репозитории до написания скриптов (правил) компиляции и сборки, методов выполнения тестирования, формирования пакетов и развёртывания. Также они формулируют для разработчиков практические рекомендации по использованию инструментов, чтобы убедиться, что проекты выпускаются с учётом соответствующих и проверенных методологий. Это позволяет командам сосредотачиваться на функциональности и пользователях вместо того, чтобы тратить время на повторное изобретение колеса.

## Основные принципы управления релизами

1. **Самообслуживание.** Каждая команда должна иметь возможность контролировать процесс релизов: как часто и когда выпускать новые версии своих продуктов.
2. **Высокая скорость.** Код часто изменяется, поэтому нужно иметь возможность разворачивать его в бою настолько быстро, насколько это возможно. Частые релизы приводят к меньшему количеству изменений между версиями. Это упрощает тестирование и поиск ошибок.
3. **«Герметичность» релизов. Идемпотентность.** Если два разных человека захотят собрать код с одним и тем же релизным тегом (версией ревизии), мы ожидаем, что результат будет одинаковым.
4. **Политики безопасности и разграничение доступов.** Должны быть правила, устанавливающие, кто может выполнять специфические операции при релизе проекта, а кто нет. К таким операциям относятся:

- code review — просмотр кода и подтверждение сделанных другим разработчиком изменений;
- инструкции по релизам;
- подтверждение запроса на интеграцию, то есть на выполнение сборки с заданным номером ревизии в репозитории исходного кода;
- развёртывание нового релиза и т. п.

## Что такое CI/CD

CI/CD — концепция непрерывной интеграции и доставки кода на продакшн сервера.

CI расшифровывается как Continuous Integration, а с CD всё немного сложнее. Под этой аббревиатурой могут скрываться два подхода: Continuous Delivery и Continuous Deployment.

Разберёмся с каждым из них.

## Continuous Integration

Считается хорошей практикой, если девелопер разрабатывает код в отдельной ветке репозитория.

В этой же ветке происходит тестирование, и после успешного прохождения всех тестов код считается готовым к тому, чтобы быть смерженным в мастер-ветку. Никакие правки и пуши напрямую в мастер не допускаются, так как мастер-ветка — это ветка, которая находится в бою, рабочая и протестированная версия кода.

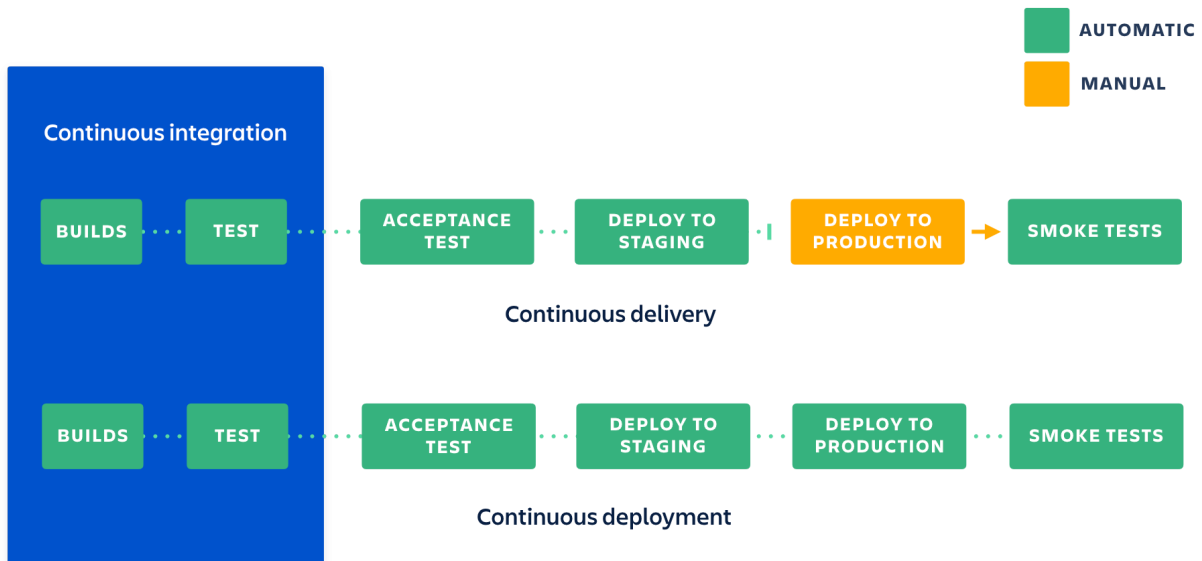
Под Continuous Integration (CI) или непрерывной интеграцией понимается концепция, при которой разработчик может мержить код в мастер так часто, как это ему требуется. Конечно, после прохождения тестирования.

Идея кажется простой и очевидной, но многие компании до сих пор к ней не пришли. Мы знакомы с компаниями, в которых разработчики могут мержить код всего два раза в день (иногда реже).

## Continuous Delivery и Continuous Deployment

Continuous Delivery — это непрерывная доставка. Концепция, при которой код, успешно прошедший pipeline — тестирование, мерж в мастер, сборку образа и т. п., — может быть выкачен в продакшн с минимальными усилиями. В идеале — нажатием одной кнопки.

Continuous Deployment — это непрерывное развёртывание. Концепция, при которой код, успешно прошедший pipeline, сразу попадает в продакшн. О pipeline мы расскажем подробнее, когда будем знакомиться с GitLab.



#### [Continuous integration vs continuous delivery vs continuous deployment](#)

Плюс подхода CI/CD в том, что есть возможность тестировать и выкатывать в продакшн небольшие изменения в коде. Это упрощает разработку, тестирование и откат кода в случае проблем.

Рассмотрим случай, в котором разработчики могут мержить код только два раза в день. Какие проблемы могут возникнуть:

1. Мерж-конфликты. Их будет очень много, и с каждым конфликтом нужно разбираться отдельно.
2. Изменения всех разработчиков за полдня сливаются вместе в одно время. Это намного сложнее тестировать, и увеличивается шанс, что что-то пойдёт не так, даже при успешном прохождении тестов.
3. Если мы всё-таки оттестировали и смержили все изменения за полдня и теперь хотим выкатить в продакшн, то в случае проблем самым быстрым способом восстановления работоспособности системы будет откат всего этого добра. То есть откат всего кода, который разработчики писали полдня, и потом разбирательства, какой же именно коммит из всей этой пачки привёл к фатальным проблемам.

Если придерживаться CI/CD-концепции, всё становится намного проще. Разработчик вносит изменения в код, тут же отдаёт его на тестирование, и, если всё хорошо, код попадает в бой. При проблемах понятно, какой именно коммит привёл к ним, так как в конкретный момент времени выгружается всего одно изменение. Его достаточно просто откатить, и ревертнуть в мастере код.

# Знакомство с GitLab

Для практической реализации CI/CD есть множество инструментов: отдельно для CI и CD и таких, где можно реализовать и CI, и CD.

С инструментом, в котором возможна реализация CI и CD, мы сегодня познакомимся.

## Как работает GitLab CI/CD

Чтобы реализовать CI/CD в GitLab, необходимо, чтобы код хранился в git-репозитории.

Весь CI/CD — в основном это build, test и deploy, скрипты — хранятся в корне репозитория, в файле `gitlab_ci.yml`.

В этом файле могут быть: скрипты для запуска, различные зависимости, команды для выполнения. Может быть определено, куда будет выгружаться приложение (docker-образ), и много чего ещё.

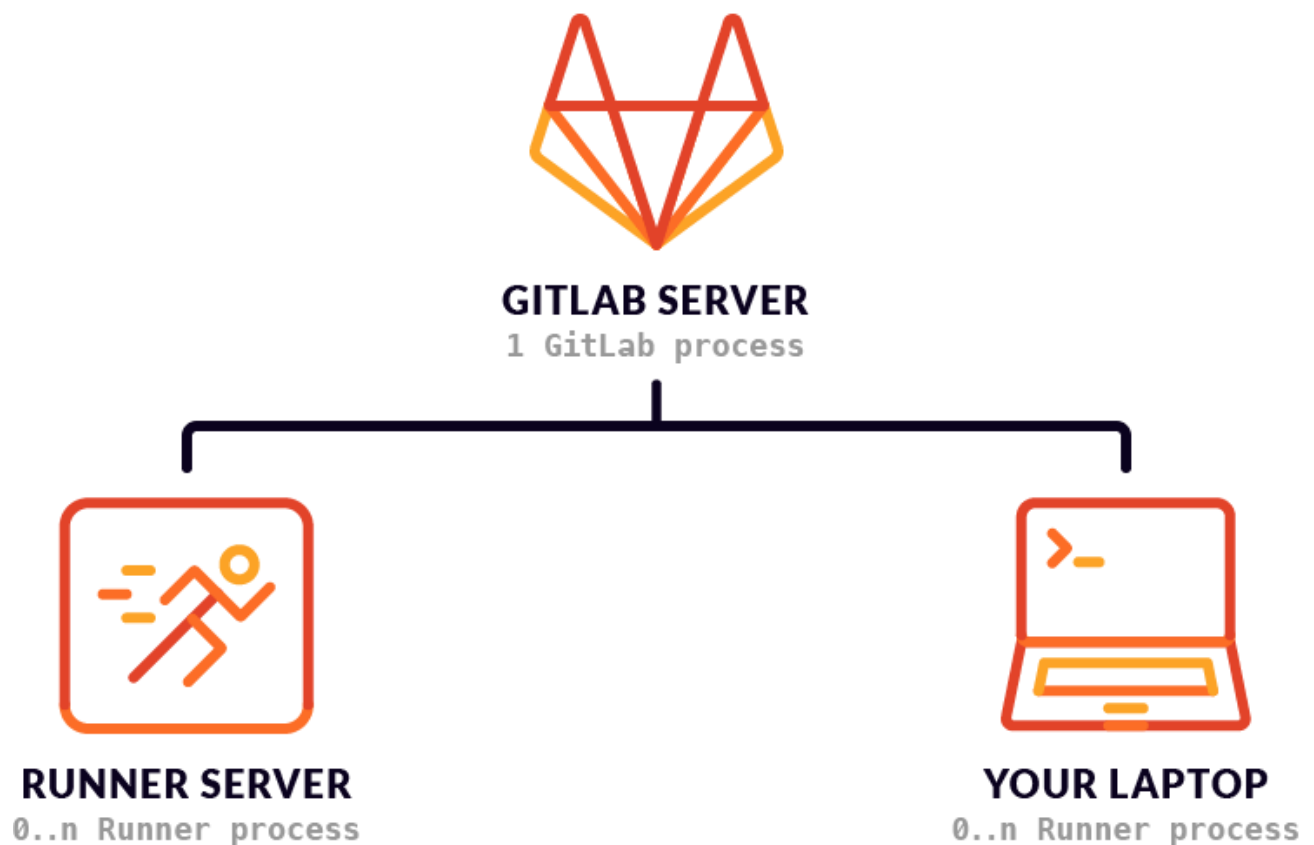
Скрипты внутри `gitlab_ci.yml` группируются в job'ы. Job'ы группируются в pipeline. Подробнее о них расскажем далее.

## Архитектура GitLab

GitLab условно можно разделить на GitLab server и GitLab runner.

Когда в репозитории есть файл `gitlab_ci.yml`, job'ы, описанные в нём, выполняются на GitLab runner'ах. Результат работы возвращается на GitLab server.

GitLab runner'ы могут быть установлены как на локальном компьютере, так и на отдельно выделенных серверах.



GitLab server состоит из множества компонентов. Мы не будем рассматривать их все, остановимся только на тех, которые могут быть нам полезны. Полное описание архитектуры GitLab можно посмотреть на [сайте с официальной документацией](#).

## Gitly

Сервис, предоставляющий доступ к git-репозиториям. Без него компоненты GitLab не смогут читать или записывать данные в git.

## GitLab Pages

Компонент, позволяющий публиковать статические сайты напрямую из репозитория GitLab.

## GitLab Shell

Предоставляет доступ к git-репозиториям через SSH.

## PostgreSQL

База данных. Вообще изначально поддерживались два вида БД: PostgreSQL и MySQL/MariaDB, но в последних версиях интеграция и оптимизация сосредоточены на PostgreSQL.

## Registry

Хранилище собственных образов Docker.

## Pipeline

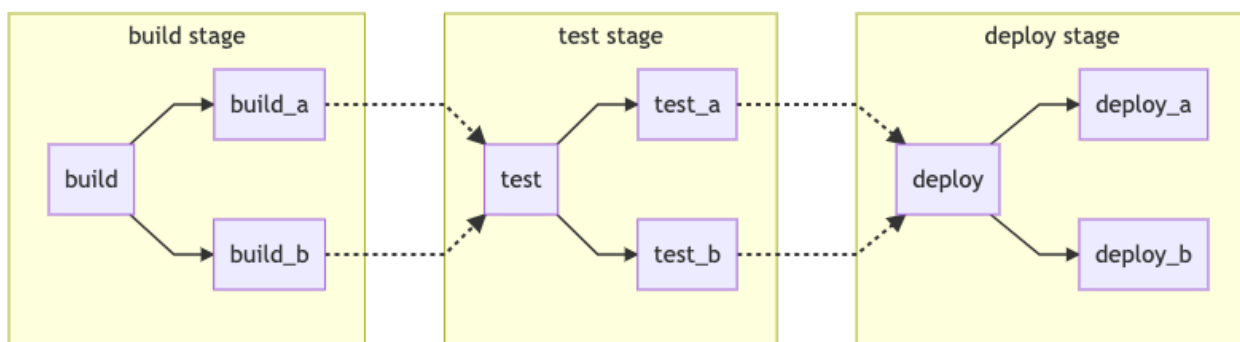
Pipeline (пайплайн) — это высокоуровневый компонент CI/CD.

Весь CI/CD GitLab'a описывается в файле `gitlab_ci.yml`, который находится в корне репозитория.

Этот файл создаёт pipeline, который запускается при изменении кода в репозитории.

Pipeline состоит из:

1. **Jobs**, которые определяют, что нужно сделать. Например, скомпилировать или протестировать код.
2. **Stages**, которые определяют, когда job'ы должны быть запущены. Например, stage, который запускает тесты после stage компиляции кода.



Здесь `build_a` и `build_b`, `test_a`, `test_b` и `deploy_a`, `deploy_b` — это job'ы.

Jobs выполняются на GitLab runner'ax.

Pipeline, то есть jobs и stages, как уже говорилось выше, описываются в файле `gitlab_ci.yml`, который находится в корне репозитория.

Для диаграммы выше `gitlab_ci.yml` выглядит следующим образом:

```
stages:
  - build
  - test
  - deploy

image: alpine

build_a:
  stage: build
```



```
script:
  - echo "This job builds something."

build_b:
  stage: build
  script:
    - echo "This job builds something else."

test_a:
  stage: test
  script:
    - echo "This job tests something. It will only run when all jobs in the"
    - echo "build stage are complete."

test_b:
  stage: test
  script:
    - echo "This job tests something else. It will only run when all jobs in
the"
    - echo "build stage are complete too. It will start at about the same time
as test_a."

deploy_a:
  stage: deploy
  script:
    - echo "This job deploys something. It will only run when all jobs in the"
    - echo "test stage complete."

deploy_b:
  stage: deploy
  script:
    - echo "This job deploys something else. It will only run when all jobs in
the"
    - echo "test stage complete. It will start at about the same time as
deploy_a."
```

Job'ы выполняются на GitLab runner'ax.

Если все job'ы внутри одного stage выполнились успешно, запускается следующий за ним stage.

Если хотя бы одна job в stage сфейлилась, pipeline прекращает свою работу и помечается как сфейлившийся (failed).

Обычно pipeline состоит из трёх stage:

- build, в котором происходит сборка артефактов;
- test — тесты;
- deploy — публикация изменений в stage-среду и production-среду.

# Переменные

В GitLab можно использовать предустановленные переменные, либо создавать переменные самостоятельно.

Создавать переменные самостоятельно можно двумя способами:

1. В самом файле `gitlab_ci.yml`:

```
variables:  
  TEST: "HELLO WORLD"
```

2. В Settings → CI/CD → Variables:

### Add variable ×

---

**Key**

**Value**

**Type**

Variable

**Flags**

☐ Protect variable [?](#)  
Export variable to pipelines running on protected branches and tags only.

☒ Mask variable [?](#)  
Variable will be masked in job logs. Requires values to meet regular expression requirements. [More information](#)

CancelAdd variable

Плюс этого способа в том, что вы можете:

1. Наследовать эту переменную в другие подпроекты.
2. Защищать переменную (protect variable). Переменные будут выполняться только в защищённых ветках и тегах.
3. Маскировать переменную (mask variable). Тогда в логе job значение самой переменной не будет отображаться в явном виде. Это может быть полезно, например, при использовании паролей.

```
$ echo $GITLAB_PASSWORD  
[MASKED]
```

## GitLab Pages

С помощью GitLab Pages можно публиковать статические сайты прямо из репозитория в GitLab.

Часто GitLab Pages используется для генерации документации в репозитории сервиса.

По умолчанию используется домен GitLab **\*.gitlab.io**, но также можно использовать свой собственный домен.

Если использовать дефолтный домен для GitLab Pages, сайт будет автоматически доступен по HTTPS. Если используется собственный домен, можно настроить использование SSL/TLS-сертификатов.




## Pipeline для GitLab Pages

Чтобы собрать и опубликовать GitLab Pages, необходимо:

1. GitLab CI/CD: в `gitlab_ci.yml` создать job с именем **pages**.
2. Создать в репозитории директорию **public**, в которой будут содержаться все файлы для публикации: HTML, CSS и т. п.

Пример, GitLab Pages:

1. Создадим директорию `public` в корне репозитория:

Name	Last commit	Last update
 <code>public</code>	Delete .gitkeep	4 minutes ago
 <code>.gitlab-ci.yml</code>	add gitlab_ci.yml	10 minutes ago
 <code>README.md</code>	Add README.md	3 days ago

2. Добавим в `public` файлы `index.html` и `style.css`:

master
dream-house / public /
+

History
Find file
Web IDE
256db685
Clone

Delete .gitkeep  
vpetukhova authored 5 minutes ago

Name	Last commit	Last update
..		
index.html	add index.html	6 minutes ago
style.css	Add style.css	5 minutes ago

index.html:

```
<meta charset="utf-8">
  <meta name="generator" content="GitLab Pages">
  <title>Plain HTML site using GitLab Pages</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="navbar">
    <a
href="https://gitlab.com/Geekbrains-CI-CD/dream-house/-/blob/master/.gitlab-ci.yml">S
chema</a>
    <a href="https://unsplash.com/photos/QVGwkHh10XQ">Picture</a>
  </div>

  <h1>We built a dream house</h1>

</body>
</html>
```

style.css:

```
body {
  font-family: sans-serif;
  margin: auto;
  max-width: 1280px;
}

.navbar {
  background-color: #313236;
  border-radius: 2px;
  max-width: 800px;
}

.navbar a {
  color: #aaa;
  display: inline-block;
```

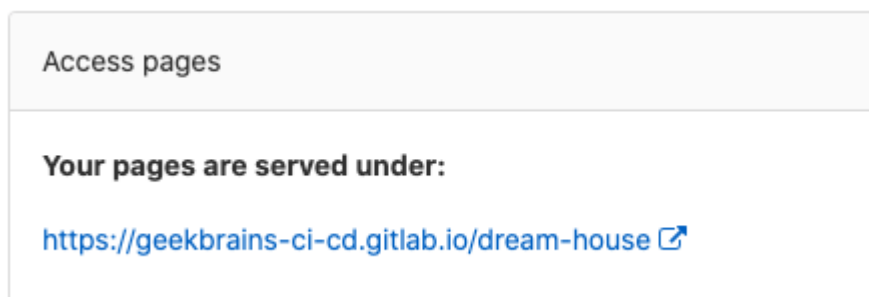
```
font-size: 15px;
padding: 10px;
text-decoration: none;
}

.navbar a:hover {
  color: #ffffff;
}
```

1. Добавим в `gitlab_ci.yml` job pages для сборки сайта только в мастер-ветке:

```
pages:
  stage: pages
  script:
    - echo 'dream house'
  artifacts:
    paths:
      - public
  only:
    - master
```

2. Теперь, когда pipeline успешно отработал, пойдём в Settings → Pages. Появилась ссылка на собранный сайт:



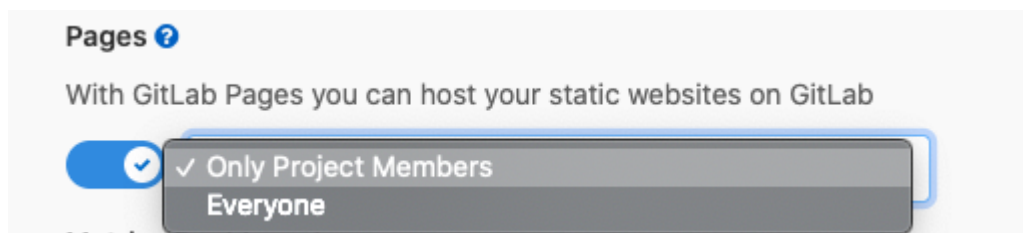
По дефолту URL выглядит следующим образом:

`https://название-репозитория.gitlab.io/название-проекта`

## Доступ к GitLab Pages

По умолчанию собранный сайт становится публично доступным в интернете. Это не всегда бывает нужно. Например, если вы собираете внутреннюю документацию сервиса, она должна быть доступна разработчикам сервиса, но недоступна всему остальному миру. В таком случае можно настроить доступ к GitLab Pages через GitLab Pages Access Control.

Для этого в Settings → General → Visibility нужно выбрать Only Project Members.



## Страницы с ошибками

В GitLab Pages можно также описывать страницы с ошибками. Например, для 403-й и [404-й ошибки](#) нужно создать в public страницы 403.html и 404.html. соответственно.

## Практическое задание

См. в уроке

## Глоссарий

**Управление выпуском новых версий или релизами (release engineering)** — это относительно новое и быстро развивающееся направление в IT, которое может быть кратко определено как разработка и доставка программного обеспечения.

**Релиз-инженеры или SRE (Site Reliability Engineer)** — это ребята, которые определяют все шаги, необходимые для выпуска ПО: от способа хранения исходных текстов в репозитории до написания скриптов (правил) компиляции и сборки, методов выполнения тестирования, формирования пакетов и развёртывания.

**CI/CD** — концепция непрерывной интеграции и доставки кода на продакшн сервера.

**Continuous Integration (CI) или непрерывная интеграция** — концепция, при которой разработчик может так часто, как это ему требуется, мержить код в мастер. Конечно, после прохождения тестирования.

**Continuous Delivery** — непрерывная доставка. Концепция, при которой код, успешно прошедший pipeline: тестирование, мерж в мастер, сборку образа и т. п., может быть выкачен в продакшн с минимальными усилиями. В идеале — нажатием одной кнопки.

**Continuous Deployment** — непрерывное развёртывание. Концепция, при которой код, успешно прошедший pipeline, сразу попадает в продакшн.

**Pipeline (пайплайн)** — это высокоуровневый компонент CI/CD. В GitLab pipeline состоит из job и stage.

## Дополнительные материалы

1. [Официальная документация по GitLab CI.](#)
2. [Вводное описание GitLab CI/CD.](#)
3. [Архитектура GitLab.](#)
4. [Документация по GitLab Pages.](#)

## Используемые источники

1. Бетси Бейер, Крис Джоунс, Дженнифер Петофф, Нейл Ричард Мёрфи «Site Reliability Engineering. Надёжность и безотказность, как в Google», 2018.
2. [Официальная документация по GitLab CI.](#)
3. [Вводное описание GitLab CI/CD.](#)
4. [CI/CD pipelines.](#)
5. [Документация по GitLab Runner.](#)
6. [Переменные.](#)
7. [Предустановленные переменные.](#)
8. [Документация по GitLab Pages.](#)