Документация GitFlic

Введение

Данная документация принадлежит сервису GitFlic - первому в России сервису хранения исходного кода.

Узнайте, как начать создавать, делиться и поддерживать программное обеспечение с помощью GitFlic. Изучите наш функционал, создайте учетную запись и присоединяйтесь к сообществу разработчиков.

В книге описан основной функционал сервиса и некоторые подсказки, применяемые при работе с GitFlic.

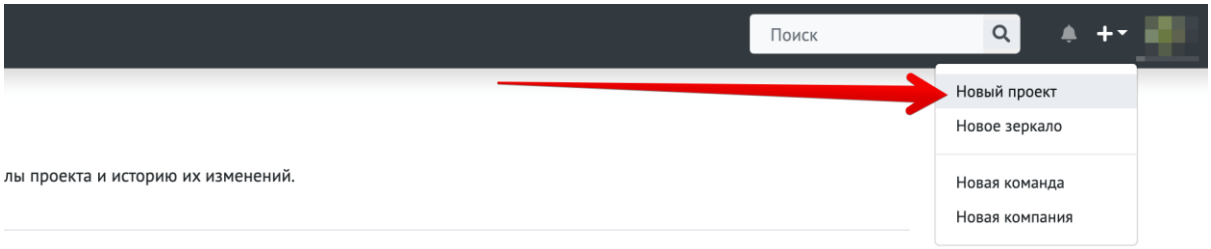
Перейти к книге можно по ссылке https://docs.gitflic.space

*Инструкция по установке OnPremise версии GitFlic находится в конце документа.

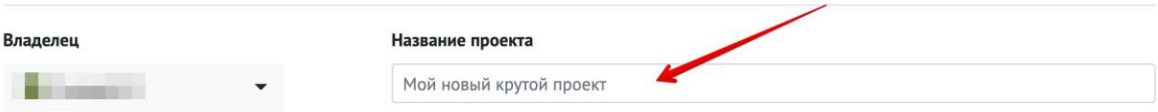
Проекты

Создание проекта

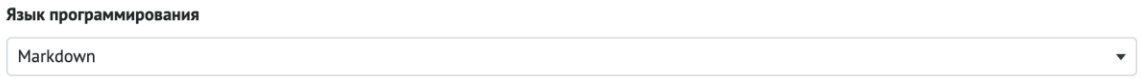
В правом верхнем углу любой страницы используйте раскрывающееся меню + и выберите Новый проект.



В форме создания проекта укажите короткое и понятное название для вашего репозитория.

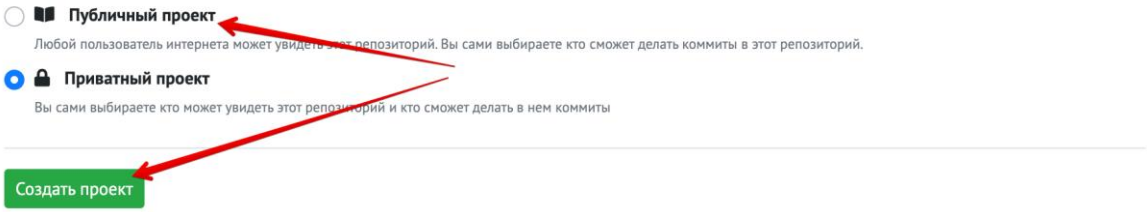


Укажите язык программирования, который используется или преобладает в вашем проекте.



При желании добавьте описание к вашему проекту. Можете оставить поле пустым и отредактировать его позже.

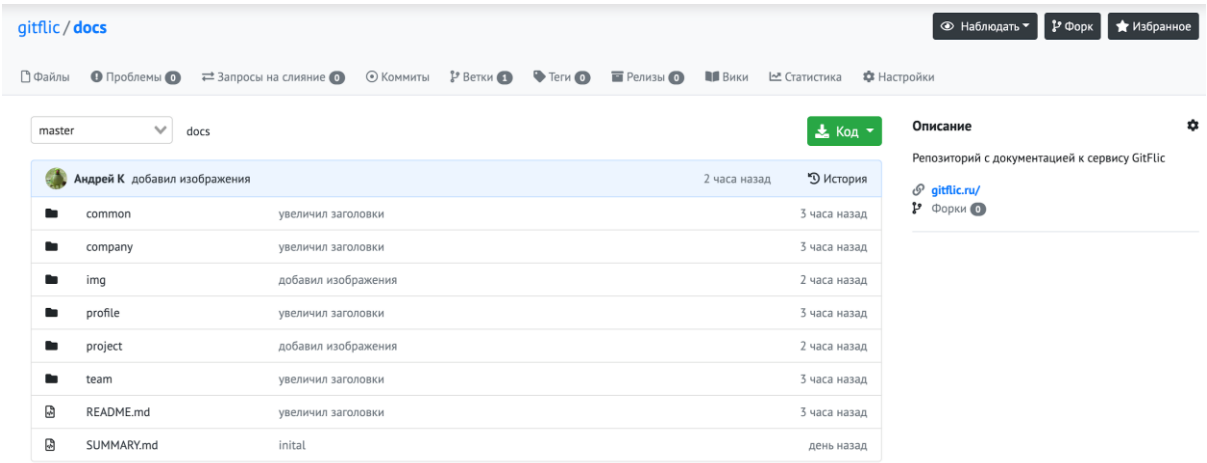
Выберите видимость репозитория, для завершения нажмите Создать проект.



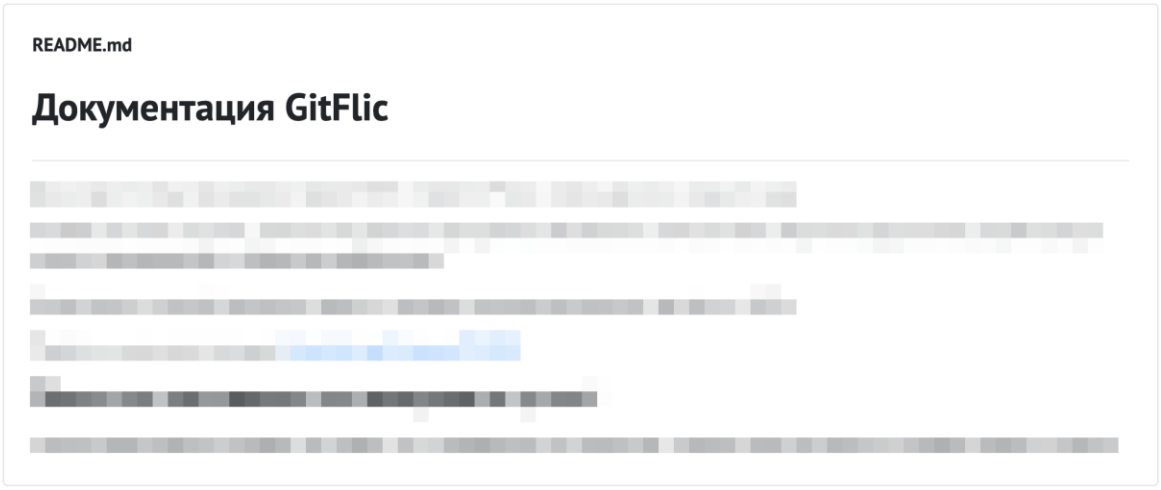
Для создания проекта на компанию или команду, в выпадающем меню выберете необходимый доступный вариант.

Просмотр проекта

На странице проекта (при условии соблюдения правил доступа) выводится список всех файлов проекта. В проекте могут содержаться папки и файлы. При прохождении во вложенные папки в панели навигации появляется меню “хлебные крошки”, по которому вы можете вернуться сразу на необходимый уровень каталога в проекте.

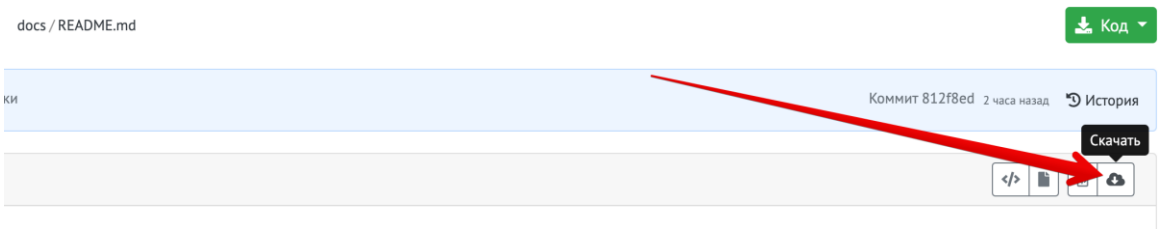


На главную страницу проекта добавьте README.md, чтобы отобразить для гостей проекта подробное описание. К такому описанию мы можете добавить изображения, таблицы или ссылки на сторонние ресурсы, если это необходимо.



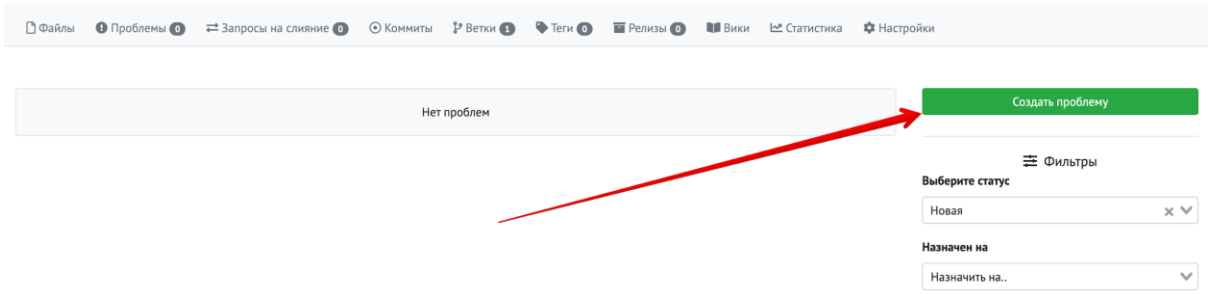
В каждой вложенной папке можно хранить файл README.md, который будет показываться аналогично главной странице проекта. Это может быть необходимо, если требуется подробнее описать содержимое папки или модуля проекта.

При работе с проектом есть возможность открыть каждый файл отдельно и посмотреть его исходный код. При необходимости мы можете загрузить его, нажав на кнопку “Скачать”

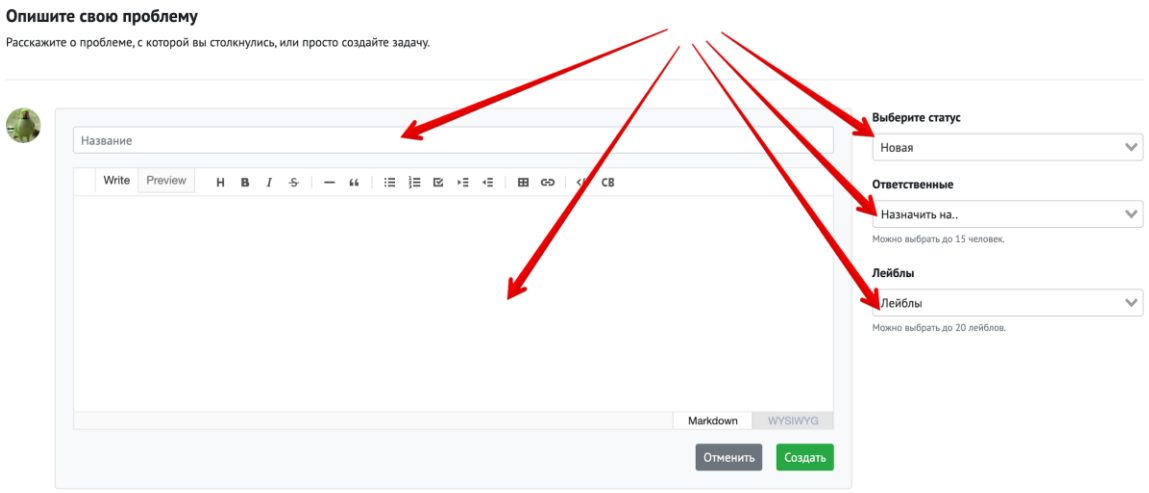


Проблемы

Для создания новой проблемы в проекте откройте раздел с проблемами и нажмите кнопку “Создать проблему”.

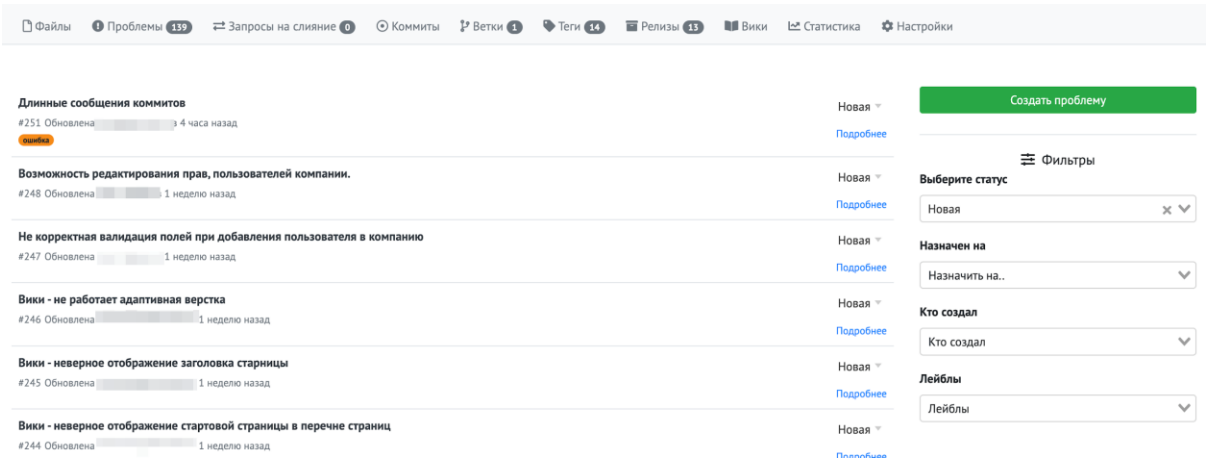


На странице создания укажите ёмкое название для проблемы, ограничение поля по длине 250 символов. Далее, если требуется, подробно раскройте суть вашей проблемы, можете прикрепить изображение и ссылки на необходимые ресурсы. Для проблемы можно указать ответственного, кто будет выполнять данную проблему, можно указать лейбл (ярлык) для проблемы. Ярлык обычно используется для визуального уточнения категории проблемы: релиз, срочный фикс, ошибка, фича.



Вы можете создавать собственные лейблы для вашего проекта, при помощи соответствующего метода API

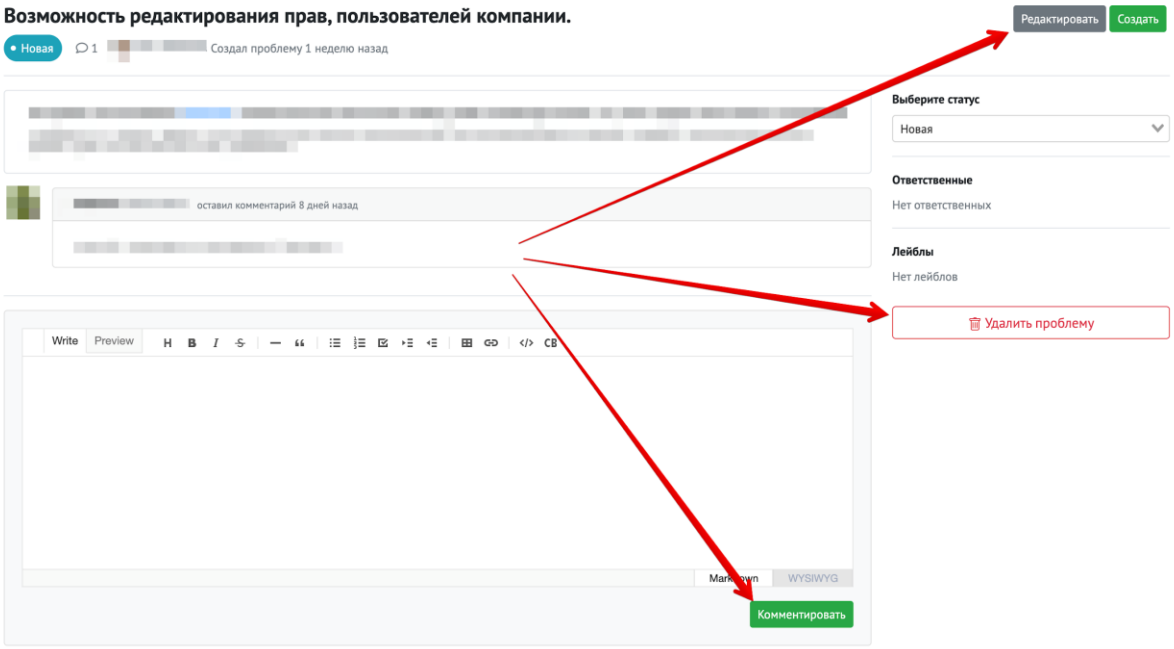
На странице с проблемами по умолчанию выводятся проблемы со статусом “Новая”. Чтобы посмотреть все проблемы, необходимо очистить фильтр в поле “Статус”. Также можно отфильтровать список проблем по доступным статусам. Стандартные статусы: новая, в работе, отменена, закрыта. Воспользуйтесь фильтрами, чтобы отыскать нужную проблему.



На странице просмотра проблемы можно оставить комментарий по теме и посмотреть ранее оставленные.

Если у вас есть права на редактирование проблем, вы можете исправить название и описание проблем, также можно изменить статус, ответственного и лейбл на текущей

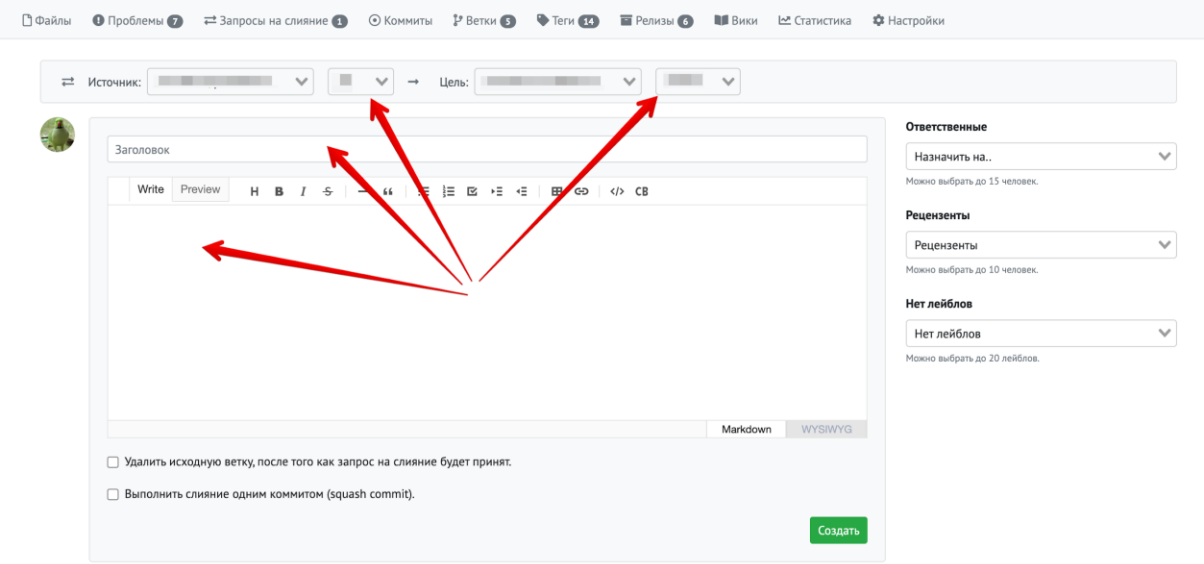
проблеме. Если проблема более не актуальна или была создана по ошибке, то проблему можно удалить, при наличии необходимых прав доступа.



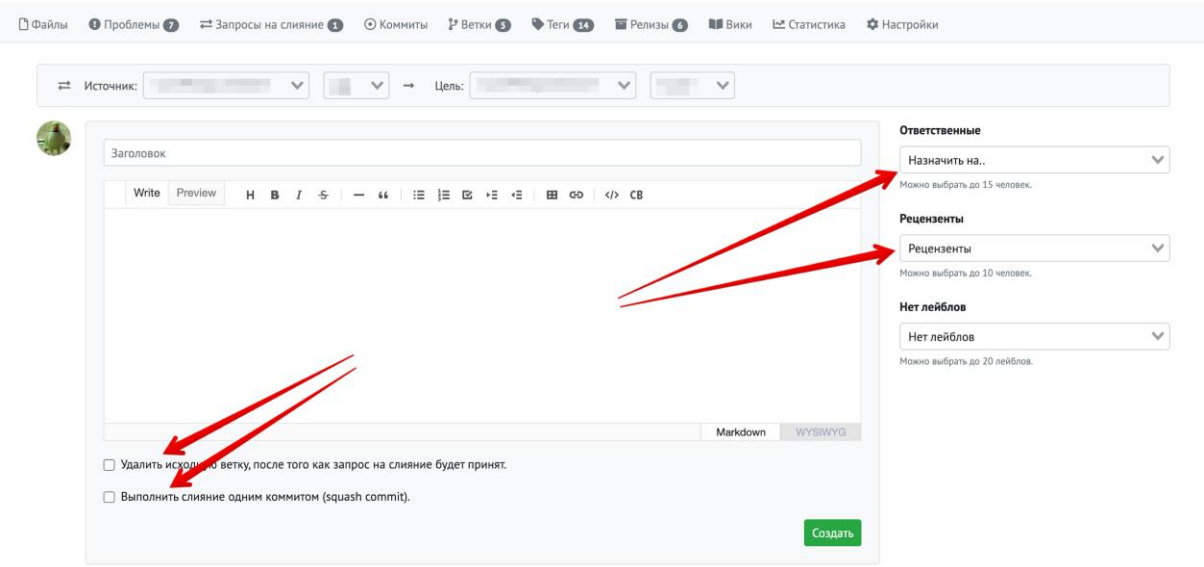
Запросы на слияние

Запросы на слияние - основной функционал на работы в вашем проекте. Он позволяет вести раздельную разработку всем участникам проекта и добавлять в основную ветку ваши доработки.

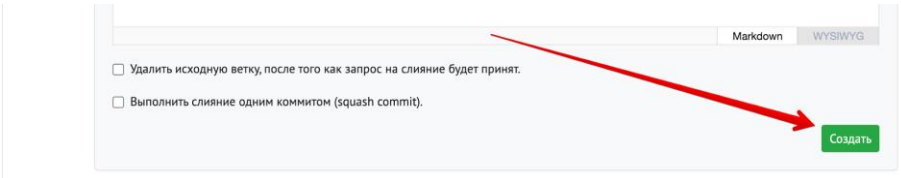
Для создания нового запроса на слияние укажите проект и ветку из которой вы хотите отправить изменения (проект указывается если вы планируете создать запрос от форка данного проекта). После перехода к странице создания запроса на слияние, необходимо указать целевую ветку, в которую будут слиты изменения. Укажите название для запроса и описание, чтобы проверяющие могли тезисно понять суть изменений или доработок.



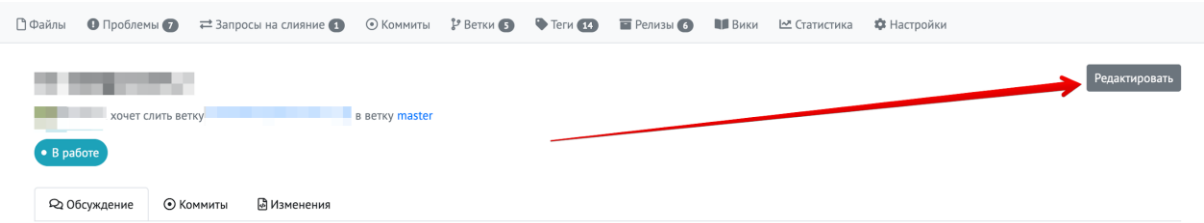
Дополнительно, но не обязательно, вы можете назначить ответственных и рецензентов для вашего запроса, так же добавить к запросу лейбл для вашего запроса. Если ветку использовать больше не планируете, поставьте галочку напротив функции удаления ветки после слития. Также используйте squash commit, если он необходим.



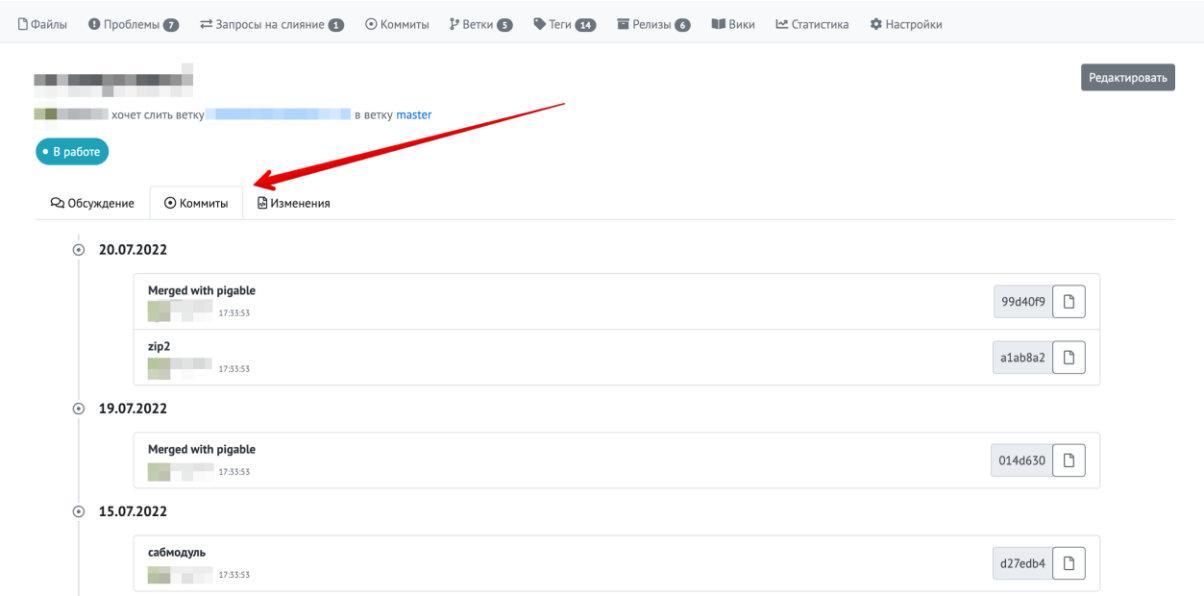
Для создания запроса нажмите кнопку “Создать”. Вас переведет на страницу созданного запроса.



Созданный запрос на слияние можно отредактировать, если вам это необходимо. Для редктирования доступны все поля, кроме исходной ветки.

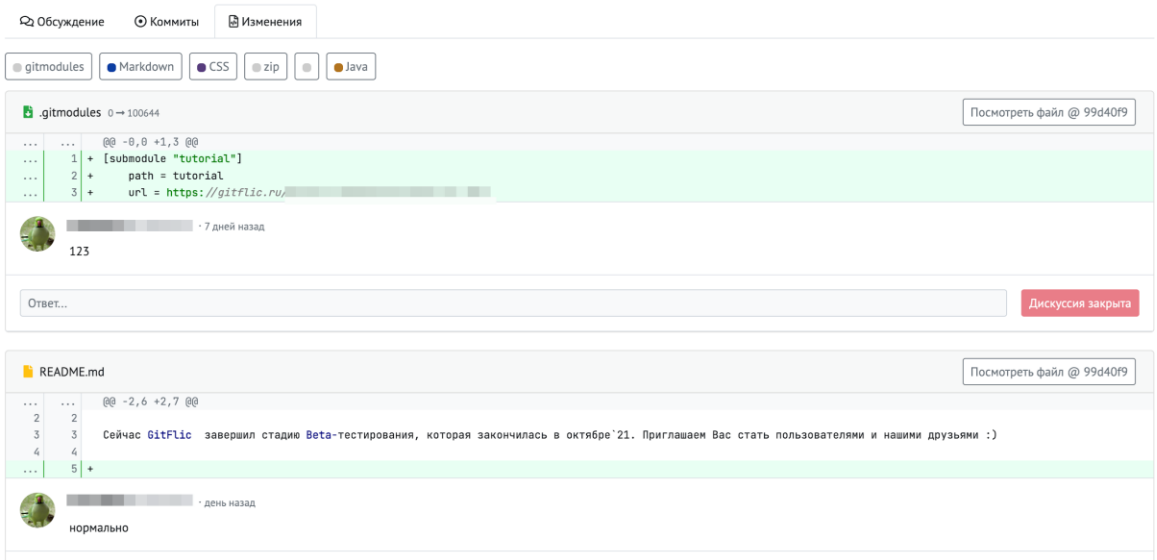


В активном запросе на слияние можно посмотреть коммиты, которые будут слиты при принятии запроса.



На странице с изменениями отображены файлы и изменения в них, либо отображено что файл был целиком добавлен, либо удален.

Также на странице с изменениями можно оставлять комментариии к конкретной строке кода. Все созданные дискуссии отображаются на обзорной странице запроса.



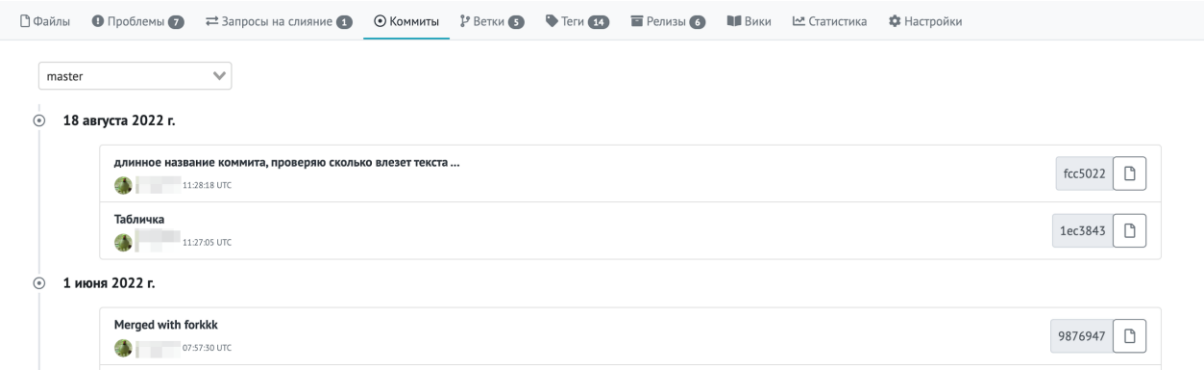
Обратите внимание, нельзя слить запрос на слияние, пока не закрыты все дискуссии!

У запросов на слияние есть три статуса: слит, закрыт и отменен. Закрытие и отмена имеют схожий функционал, они не сливают изменения в целевую ветку, данное различие в названии необходимо для упрощения работы в группе.

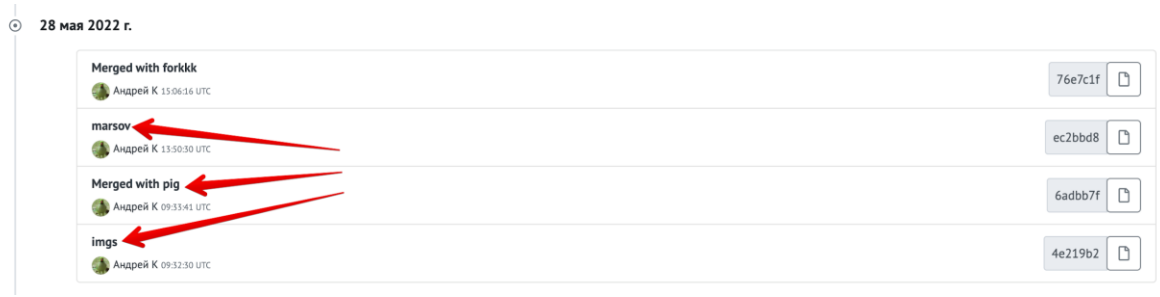
Вы можете использовать в работе правила для слития запросов. Указывайте без чьего подтверждения запрещено сливать изменения. Подробнее

Коммиты

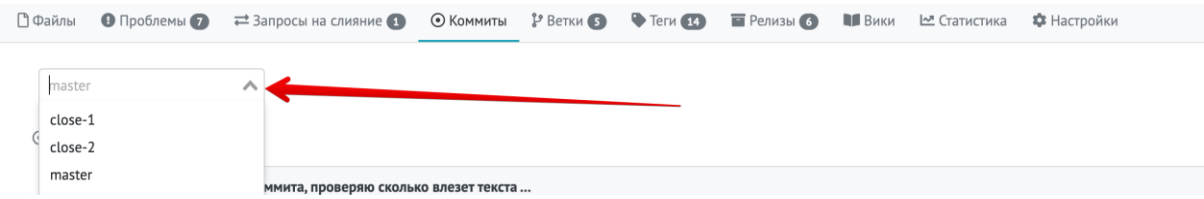
При работе с проектом можно посмотреть его историю изменений по коммитам и по отдельным веткам.



Перейдите по заголовку коммита, чтобы детально посмотреть внесенные изменения.

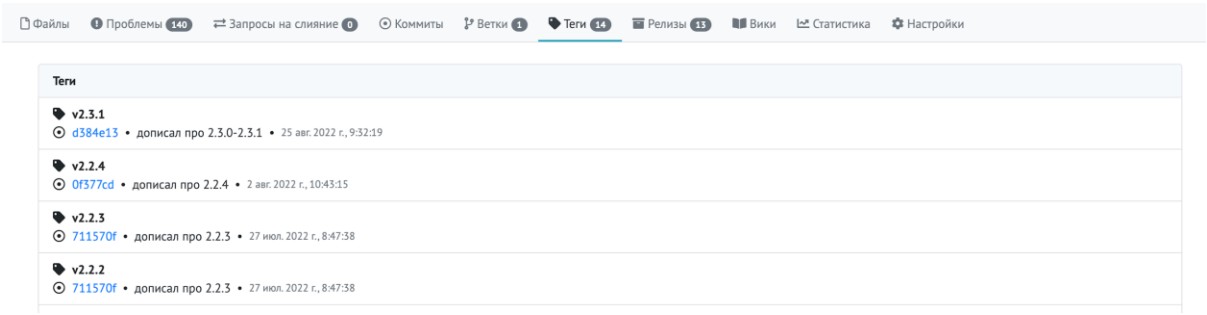


Переключайте ветки для просмотра полной истории изменений в проекте



Теги

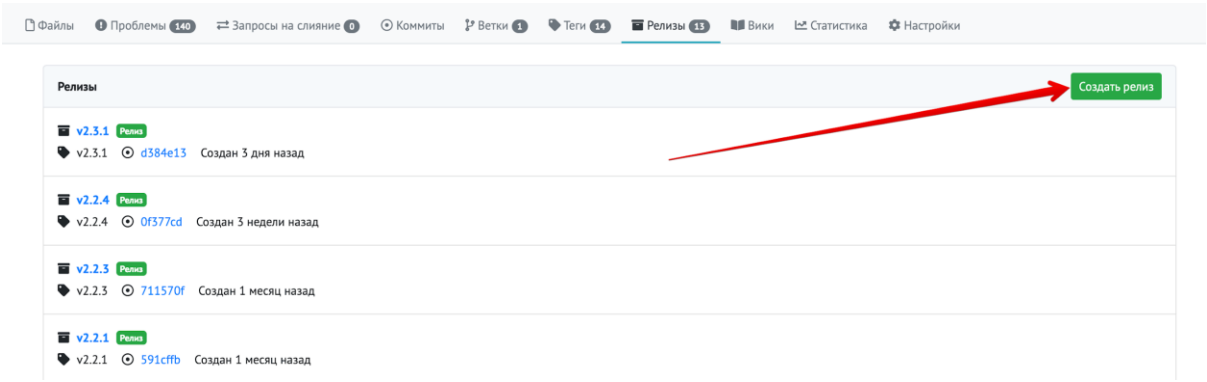
На странице тегов выведен список тегов и связанных с ним коммита. Пройдите по заголовку, чтобы получить информацию о теге. Теги используются для оформления релиза проекта.



Релизы

На странице создания релиза вы можете опубликовать промежуточную сборку вашего проекта со ссылкой на необходимый тег.

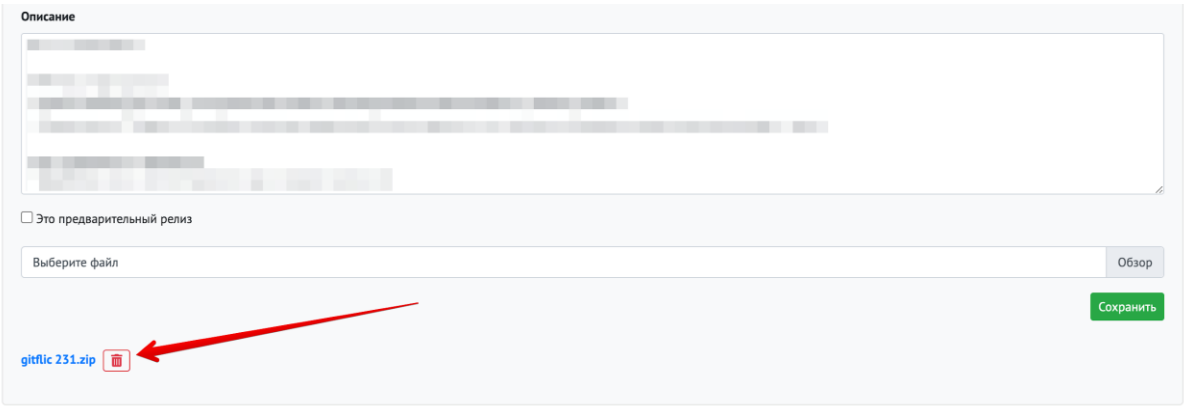
Для создания нового релиза нажмите кнопку “Создать релиз”.



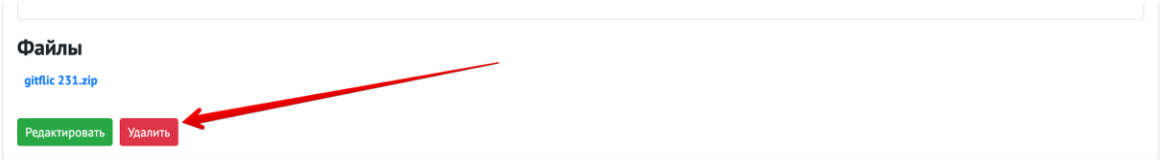
На странице создания выберете в селекторе нужный тег, укажите название для релиза, добавьте описание, если это необходимо. Далее выберете файл релиза через проводник с вашего компьютера и отметьте галочкой, является ли ваш релиз предварительным. После заполнения полей нажмите кнопку “Создать релиз”.

Обратите внимание, на один тег можно создать только один релиз

После создания вы можете редактировать ваши существующие релизы. Для этого в списке релизов перейдите в один из ваших релизов и нажмите кнопку “Редактировать”. Вам будут доступны для изменения те же поля, что и при создании. Вы можете удалить отдельные файлы со страницы релиза, если загрузили их случайно или решили их заменить.



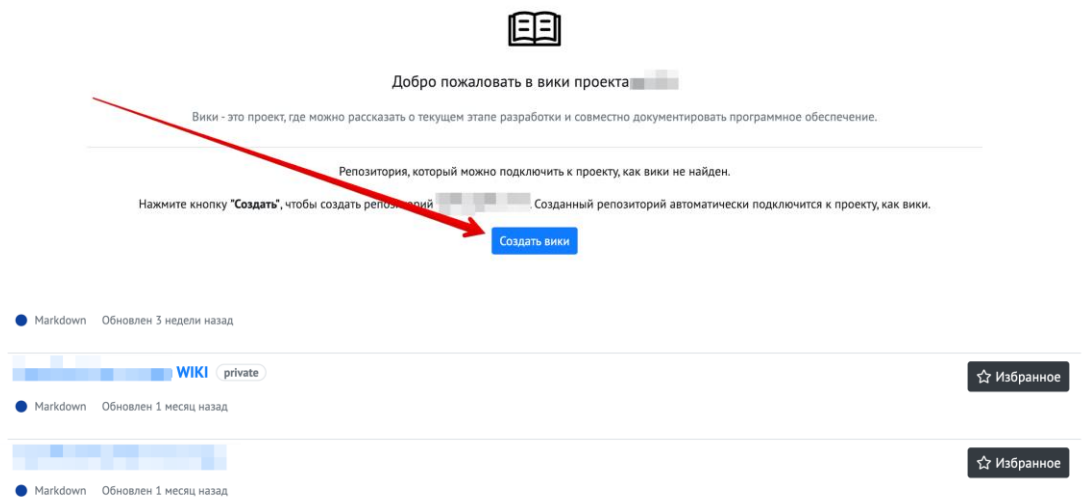
Если загруженный релиз более не актаульный или был создан ошибочно, вы можете его удалить. Для этого перейдите на страницу нужного релиза и нажмите “Удалить”.



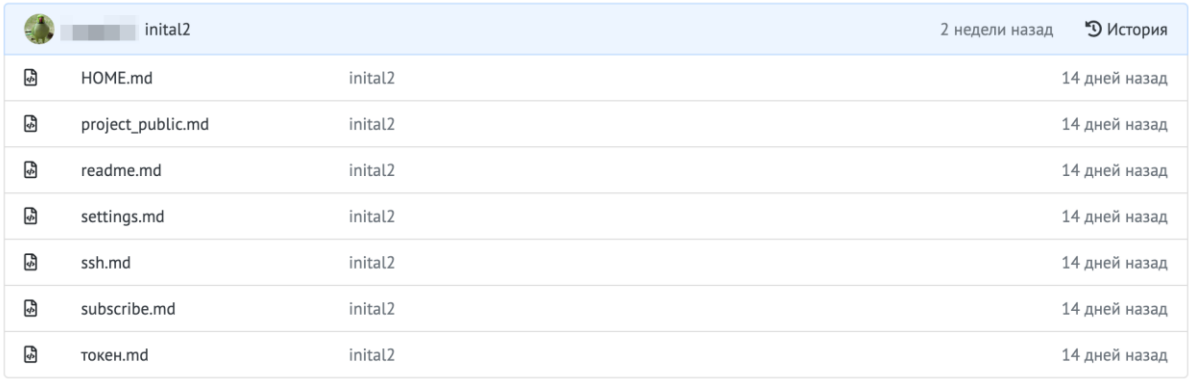
Вики

Настройка

Чтобы активировать документацию (вики) к проекту, необходимо открыть вкладку на странице проекта с названием “Вики”. На странице будет простой настройщик, в котором необходимо нажать всего одну кнопку “Создать вики”. В вашем профиле или профиле компании появится приватный проект с соответствующим названием.



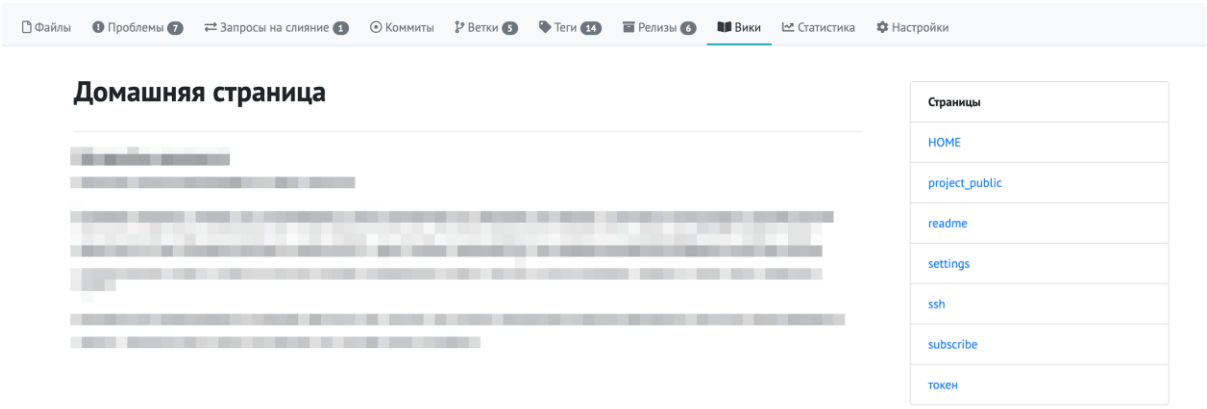
В новый Вики проект добавляйте текстовые файлы в разметке markdown, также можете собирать в папки группы файлов (обратите внимание, поддерживаются только папки первого уровня). Стартовый экран инициализируется через добавление в корне файла home.md.



Использование

После добавления достаточного количества страниц с описанием вашего проекта, вики будет доступна на вкладке “Вики” в вашем проекте. Вы можете добавлять новые

файлы и редактировать старые, изменения в документации будут опубликованы сразу после загрузки в вики-проект.



Вы можете сделать форк от вики-проекта, чтобы предложить свои улучшения к существующей документации.

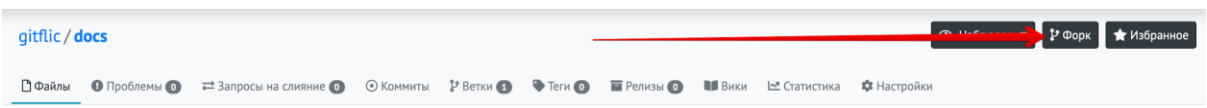
Чтобы отвязать от проекта вики-проект, необходимо в настройках проекта, в разделе Основное, спуститься к блоку “Опасная зона” и нажать “Отвязать вики-репозиторий”. При этом вики-проект не будет удален, он останется в списке проектов, будет удалена только связь между проектами.

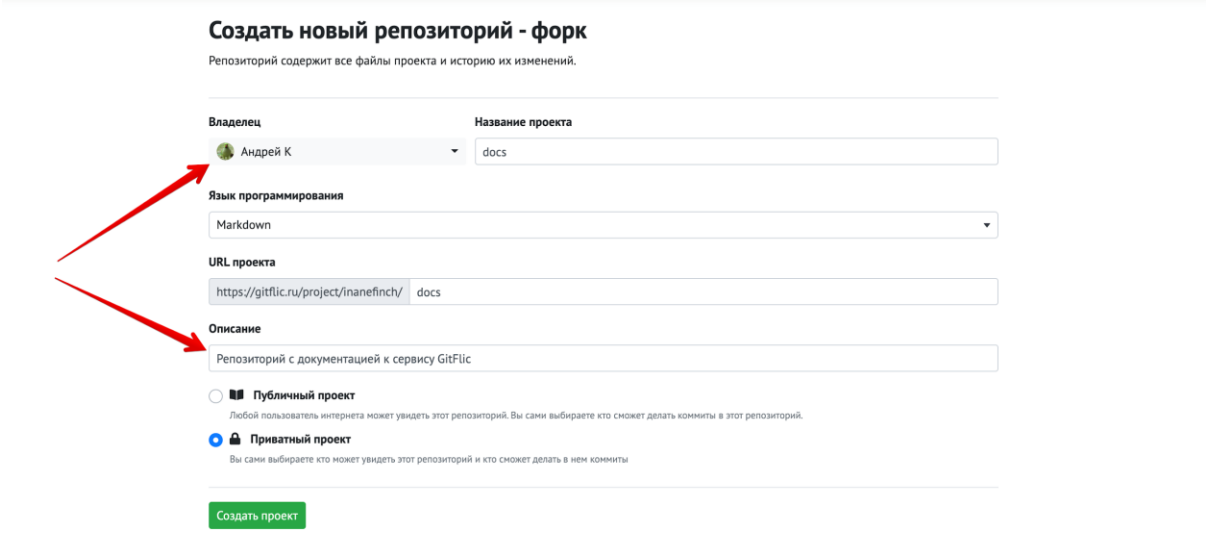
При передаче проекта с подключенной документацией, передается только сам проект, вики-проект необходимо передавать отдельно от основного.

Создание форка

Форк (от англ. вилка) проекта представляет из себя функцию создания клона проекта, аналогично зеркалу. Отличие форка от зеркала в том, что вы можете продолжить работать с проектом и вносить в него изменения. Также вы можете предложить свои изменения в начальный проект.

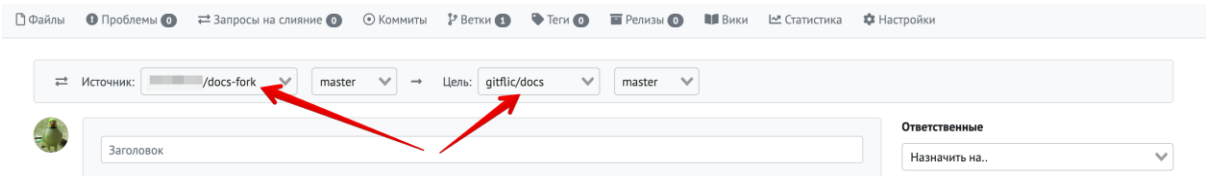
Создать форк от проекта можно на странице проекта, в верхнем правом углу нажмите кнопку “Форк” для перехода к странице создания. На странице создания автоматически перенесены параметры исходного проекта, вы можете назначить нового владельца, отредактировать название нового проекта, язык программирования и приватность.





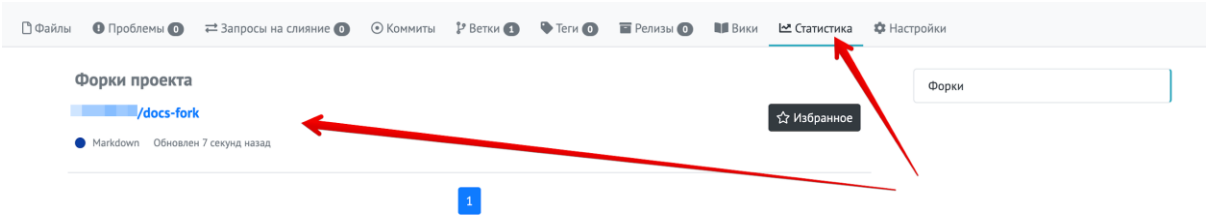
Обратите внимание, невозможно сделать публичный форк приватного репозитория.

Для создания запроса на слияние из форка в исходный проект, перейдите на страницу запросов на слияние, в селекторе выберете проект и ветку, из которой собираетесь сделать запрос на слияние. После того, как перейдете на страницу оформления запроса, укажите целевым проектом исходный проект и создайте запрос. Запрос на слияние будет аналогичен запросу, который был создан в рамках одного проекта.



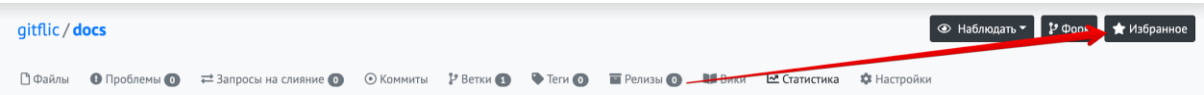
Статистика форков

На вкладке “Статистика” отображены все публичные форки текущего проекта. Вы можете перейти в один из них по заголовку и изучить его как обычный проект.

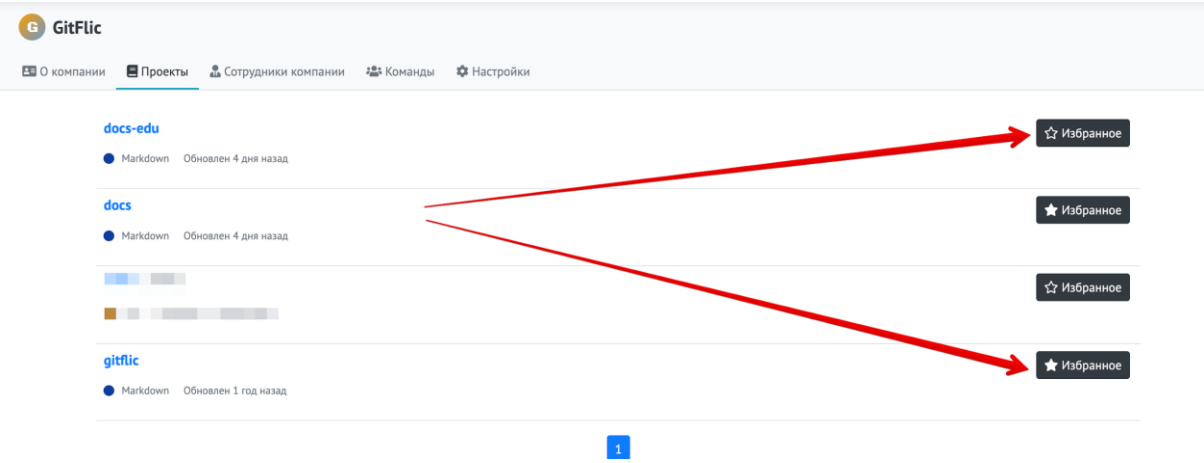


Добавление в избранные

Для сохранения проекта в избранное, нажмите в верхнем правом углу кнопку “Избранное”.



Также функция добавления в избранное доступна на странице списка проектов.



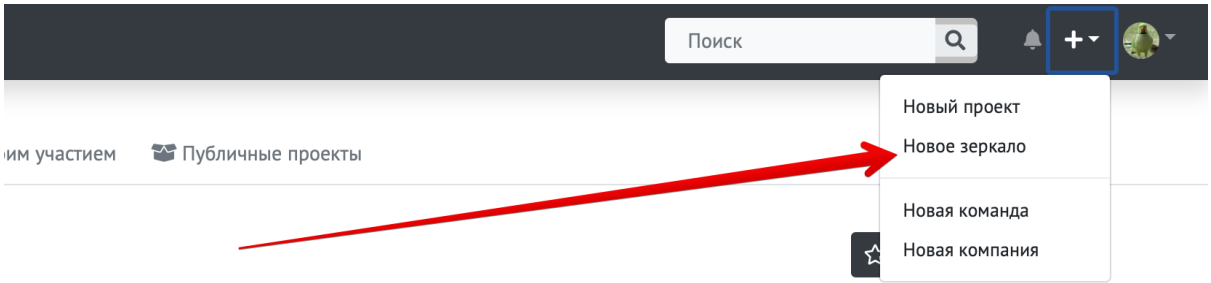
Все избранные проекты отображаются на странице проектов, на вкладке Избранное.

Зеркалирование проектов

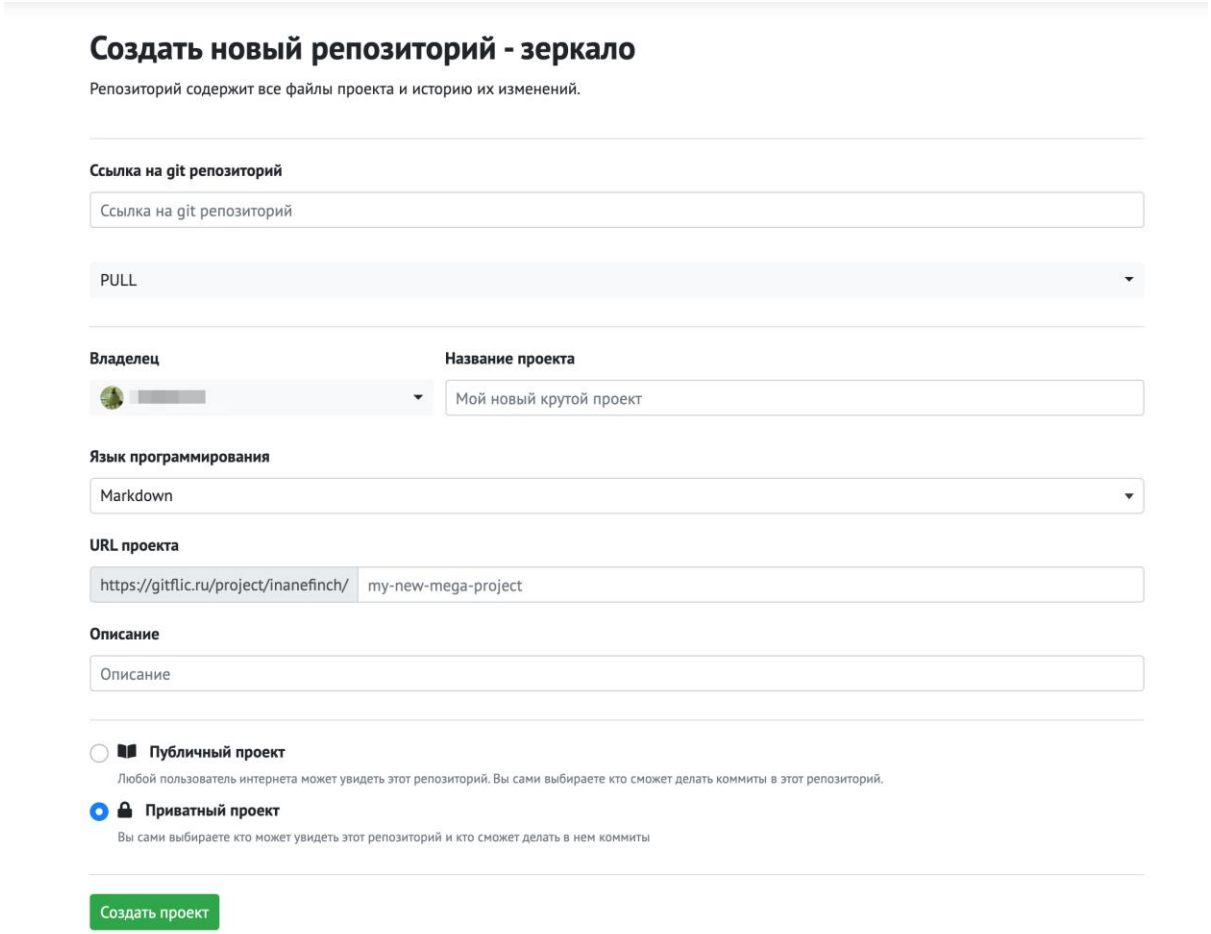
Для использования данного функционала необходимо иметь права пользователя для создания зеркал.

Оформление

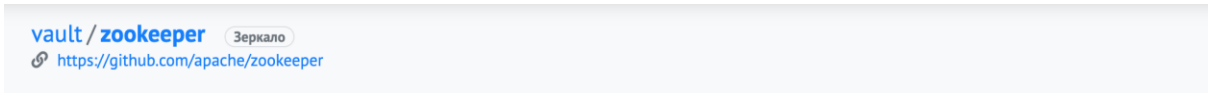
Для создания нового зеркала проекта нажмите на выпадающее меню в верхнем меню с иконкой +. Выберете вариант “Новое зеркало”.



На странице создания вставьте ссылку на публичный репозиторий. Ссылка на репозиторий должна оканчиваться на .git. Метод укажите PULL, все остальные поля заполните как для обычного проекта и нажмите “Создать”. Ваш проект встанет в очередь на зеркалирование и автоматически обновится через некоторое время (обычно в пределах 10 минут, без учета очереди).



После завершения процесса зеркалирования ваш проект будет иметь ярлык “Зеркало” и содержать в шапке ссылку на исходный проект.



Настройка проекта

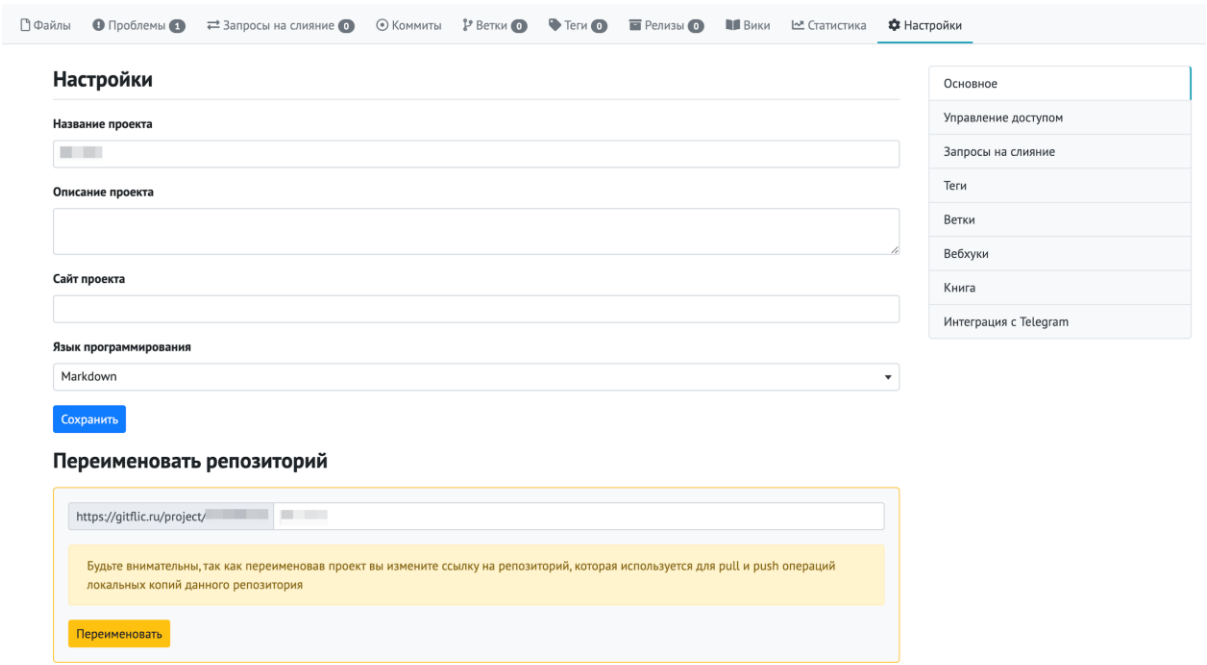
Раздел с описанием функционала настроек проекта и сопутствующих функций

Основное

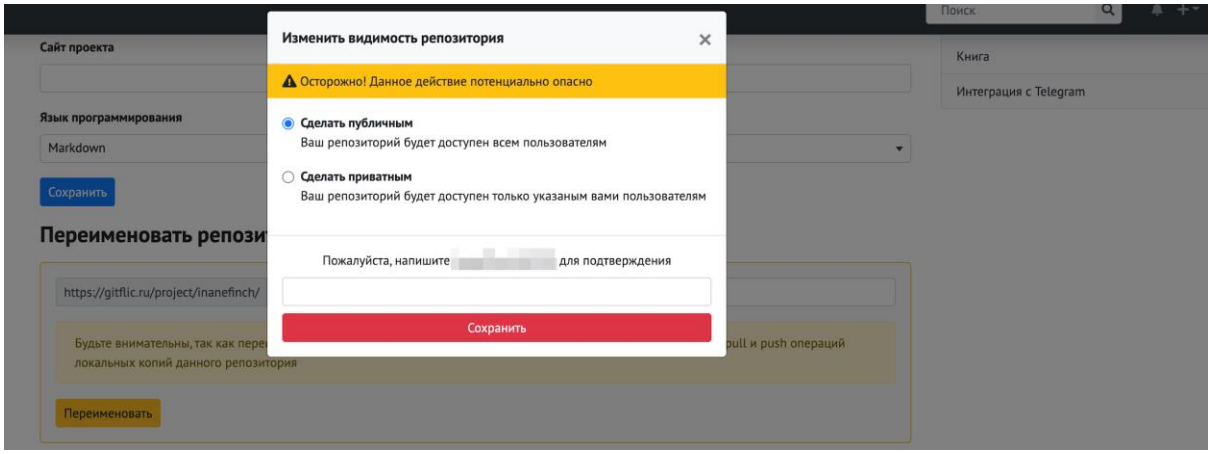
В настройках можно изменить все параметры, которые были указаны при создании проекта. Доступ к настройкам проекта имеет только владелец, либо назначенный администратор.

Также имеются дополнительные поля, которые можно заполнить, например, вы можете указать связанный с проектом сайт.

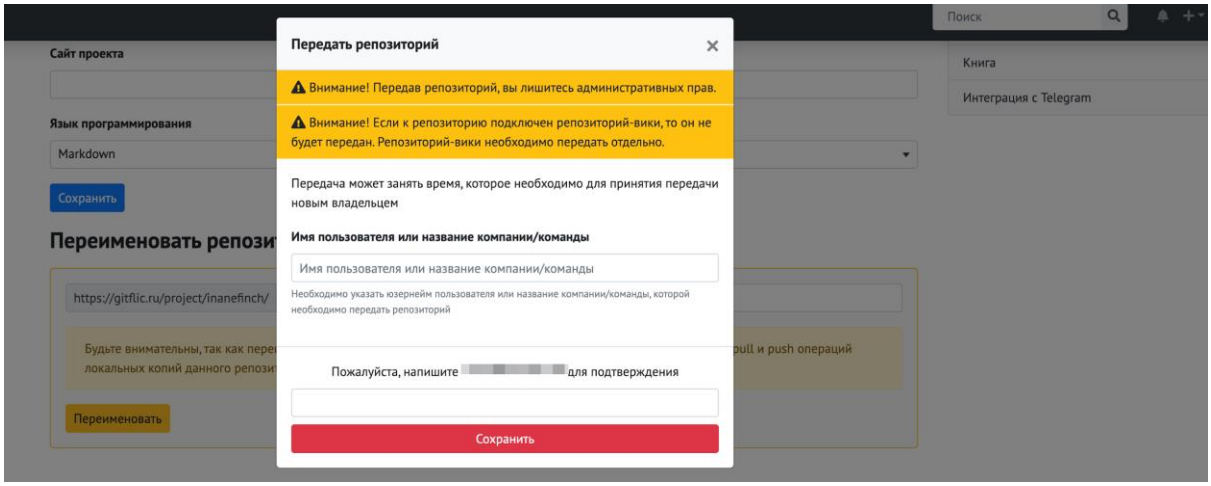
Вы можете изменить Url вашего проекта, однако старый адрес после сохранения перестанет работать. Не изменяйте адрес проекта, если не уверены в последствиях.



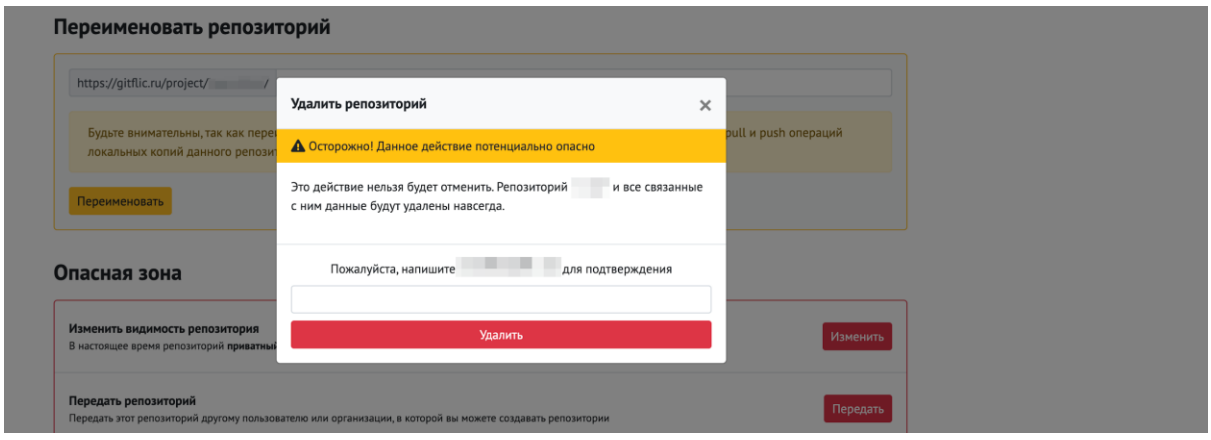
Чтобы изменить видимость проекта, нажмите на указанную кнопку. В открывшемся окне измените параметр видимости проекта и введите контрольную строку для подтверждения действия (ее можно скопировать и вставить).



Передача проекта происходит аналогично процессу смены видимости и требует ввода фразы для подтверждения действия, однако, для передачи проекта новый владелец должен быть участником проекта с правами владельца.



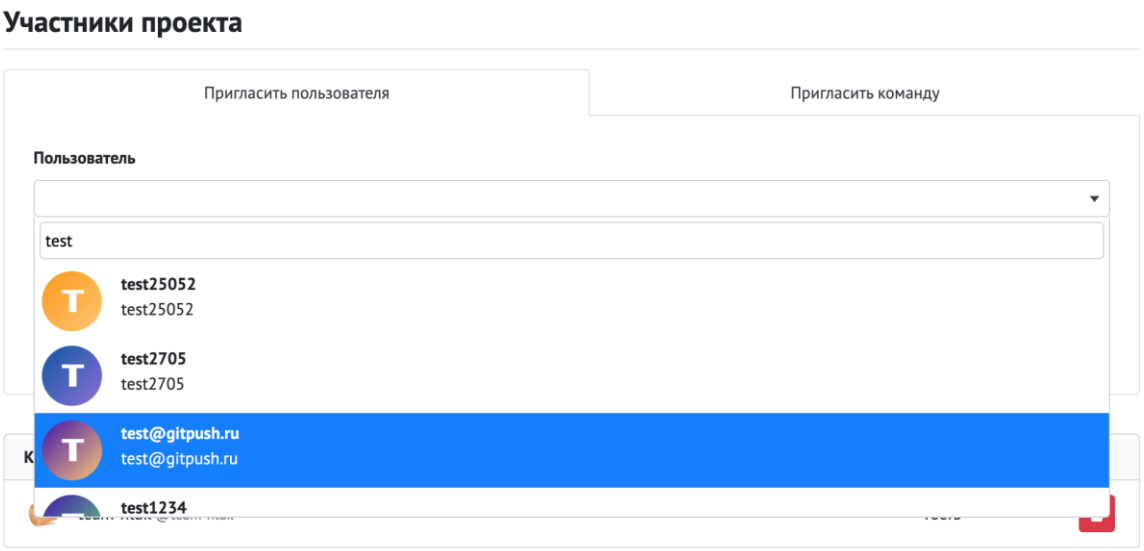
Для удаления проекта воспользуйтесь функцией удаления. Подтвердите контрольной строкой ваше действие.



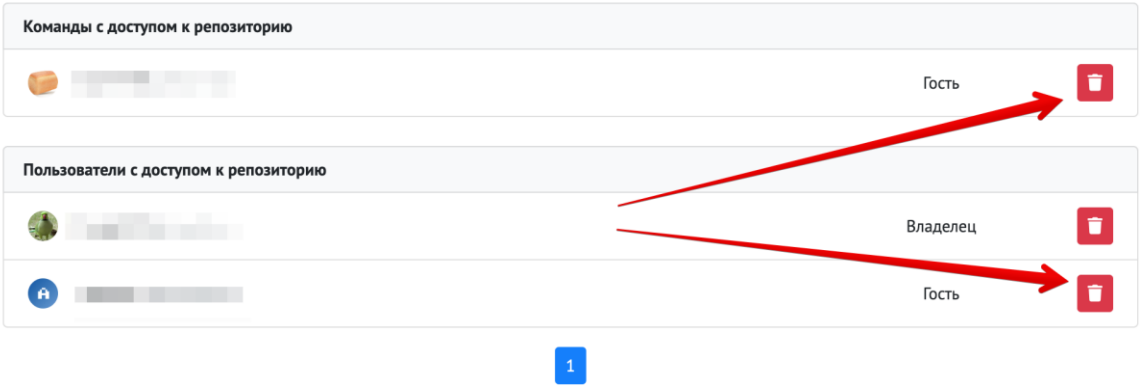
Действие удаления проекта - необратимо, не используйте эту функцию, если не осознаете последствий или не уверены.

Управление доступом

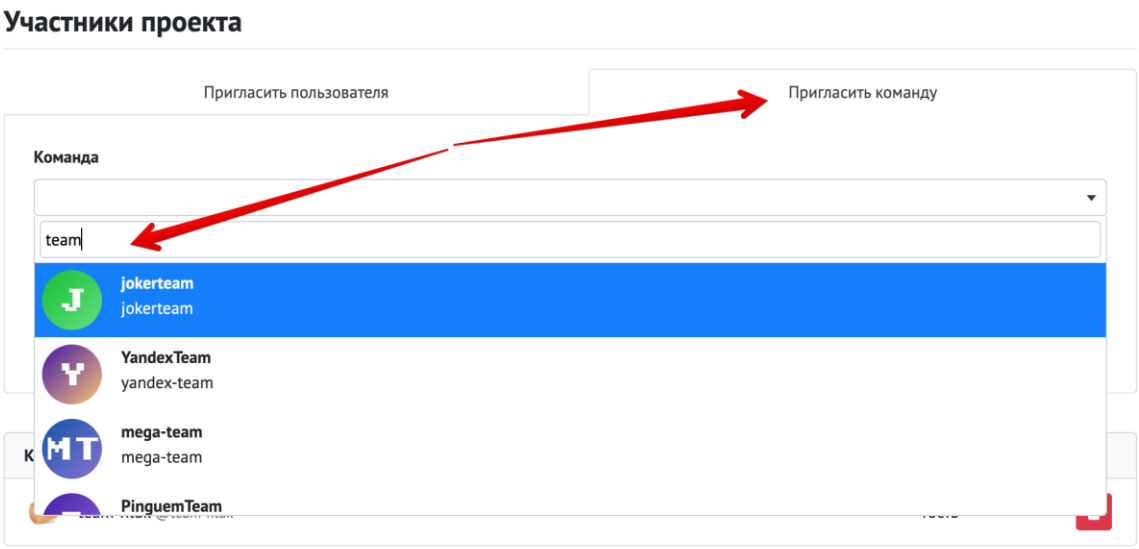
Для добавления новых участников проекта откройте вкладку управления доступом. Для поиска пользователя, которого вы собираетесь добавить, нажмите на поле “Пользователь”, откроется выпадающее меню с поиском. Начните вводить имя пользователя или имя профиля, поиск начнется после ввода не менее 3 символов. Далее укажите роль для нового участника вашего проекта и сохраните изменения. Приглашение будет отправлено выбранному пользователю на почту и на страницу уведомлений в его профиль.



Для удаления пользователей из списка участников, напротив имени пользователя нажмите на кнопку удаления.



Также вы можете добавить публичную команду к вашему проект, для это переключите вкладку на “Пригласить команду”, в поле “Команда” начните вводить название команды, вам будут выводиться все подходящие под запрос варианты. Указаная роль будет распространена на всех участников команды и будут приоритетной в проекте относительно ролей участников внутри команды.



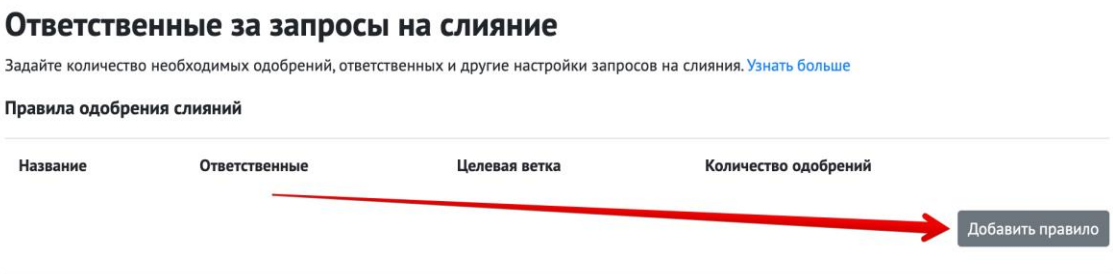
Если ваш проект публичный, то вы можете добавлять неограниченное количество участников. Но если вы решите сделать проект приватным, то на него будут распространяться условия тарификации GitFlic.

Запросы на слияние

Настройка

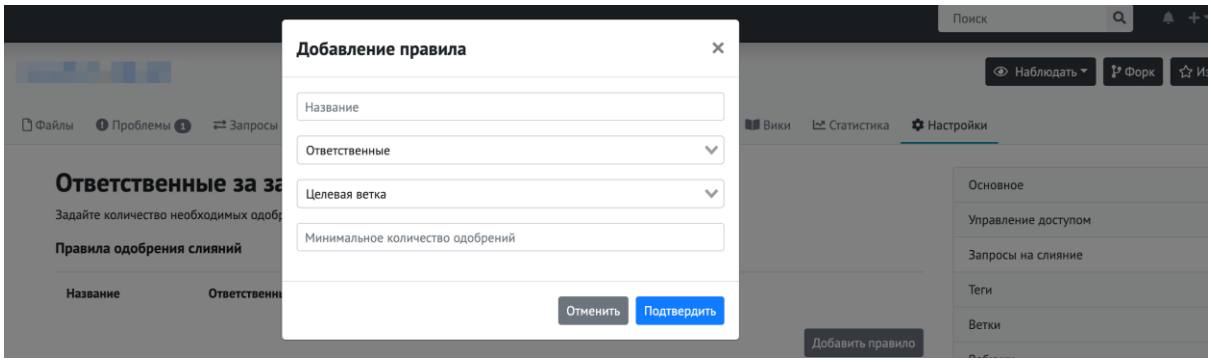
- Данная функция доступна только в Enterprise версии Для активации функции перейдите в настройки проекта, откройте вкладку Запросы на слияние

Для добавления нового правила для одобрения запросов нажмите “Добавить правило”.

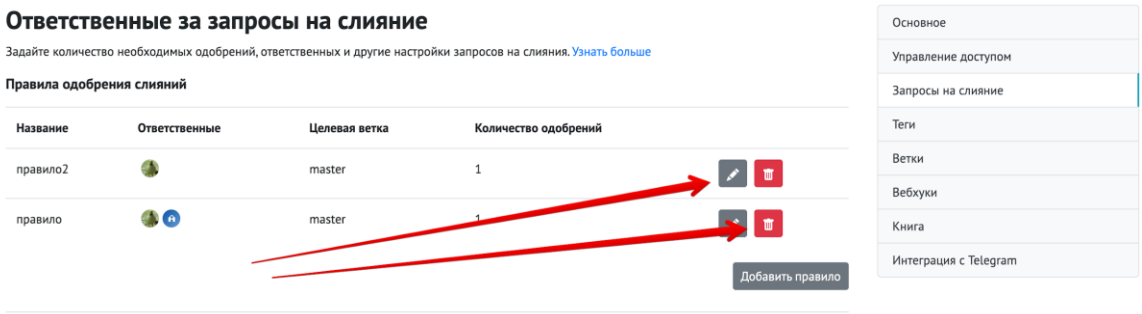


Укажите название для правила, далее выберете ответственных, от которых необходимо подтверждение для слития запроса на слияние. В селекторе “Целевая ветка” необходимо выбрать ветку проекта, для которой будет работать данное правило, например, если выбрать ветку master, то при слитии в данную ветку будет работать данное правило. В поле “Минимальное количество одобрений” введите число одобрений, достаточных для того, чтобы считать запрос на слияние доступным для слияния. Если нет необходимости ожидать одобрения от всех ответственных

разработчиков, то можно указать число меньшее, чем количество ответственных в правиле. Нажмите “Добавить правило” для сохранения изменений.



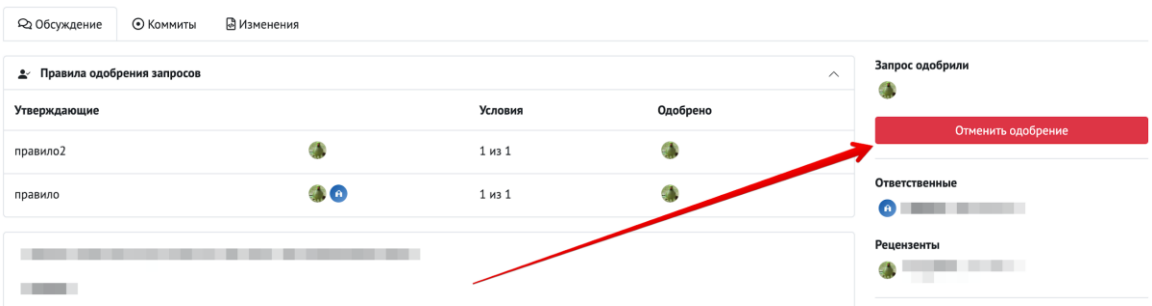
Вы можете внести изменения или удалить существующее правило соответствующими кнопками редактирования и удаления. Если вы укажите несколько правил на одну ветку, то необходимо выполнения каждого из этих правил.



Можно также указать дополнительные настройки для правил слияния веток.

Использование

При активном правиле работает ограничение к запросам на слияние, более невозможно слить изменения в ветку, пока необходимое количество ответственных не нажмут на подтверждение запроса, тем самым выразят одобрение к внесенным изменениям. После накопления необходимого количества отметок, разблокируется доступ для слития изменений.

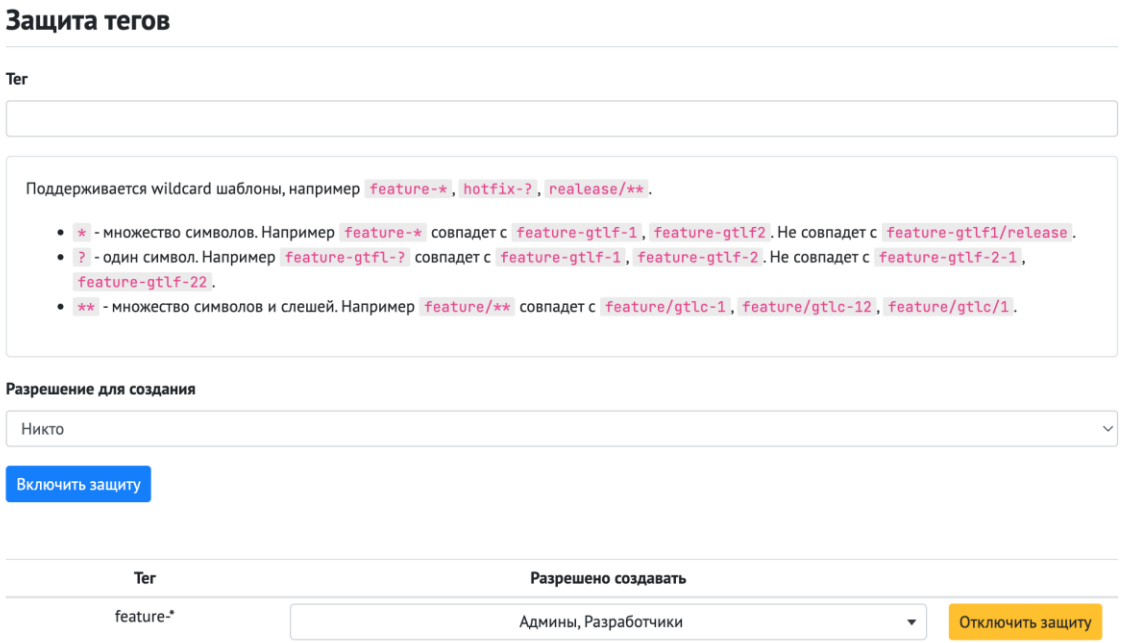


Защита тегов

Когда вы добавляете правило защиты тегов, все теги, соответствующие предоставленному шаблону, будут защищены. Только пользователи, чья роль указана в правиле, смогут создавать или удалять защищенные теги.

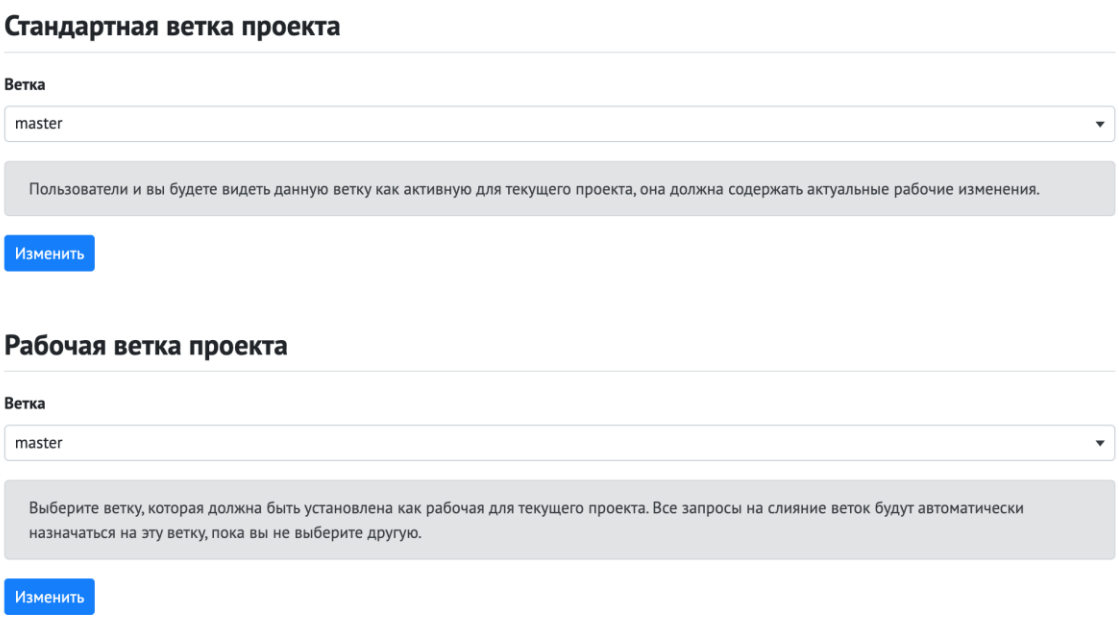
Для создания правила используйте шаблон (подсказки представлены на странице создания). Шаблон необходимо записать в поле “Тег”, далее выберете кто будет иметь доступ к работе с данными тегами. Нажмите “Включить защиту”, чтобы применить правило к проекту.

Созданные правила отображены в таблице на странице защиты тегов. Вы можете отредактировать существующие правила, либо убрать их по кнопке “Отключить защиту”.



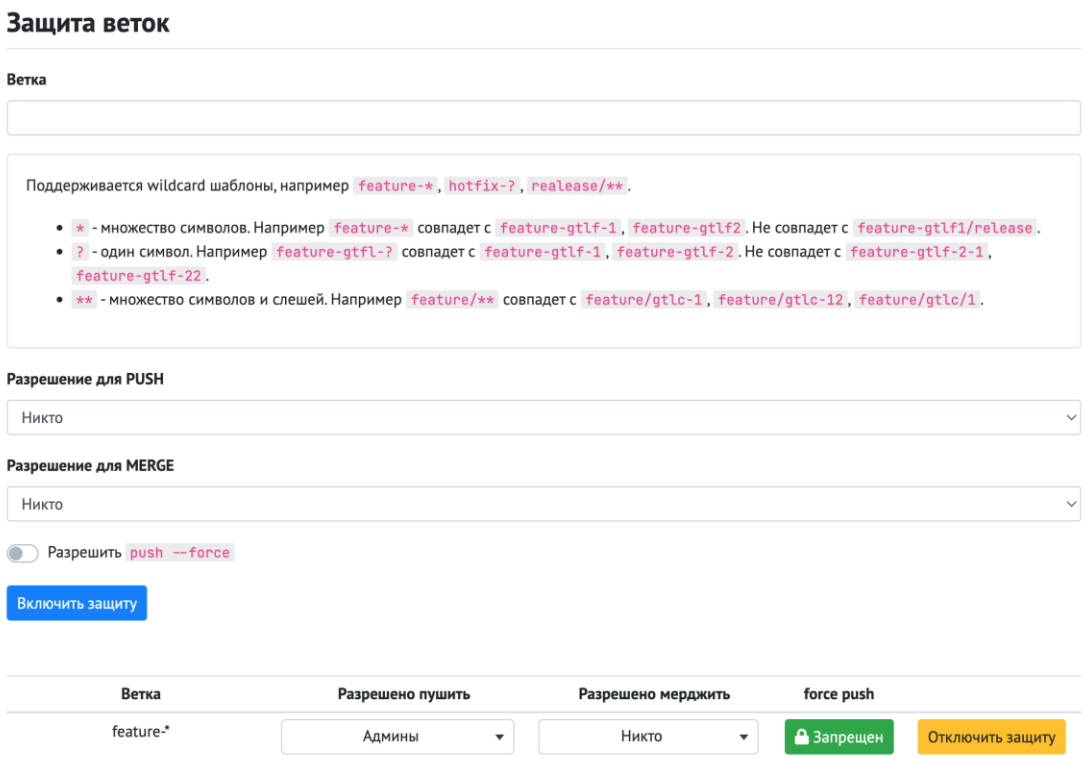
Настройка веток

На странице настроек веток можно указать стандартную ветку, которая будет отображаться на странице просмотра проекта. Рабочая ветка - это ветка на которую будут ссылаться GitFlic при создании запрос на слияние.



Вы можете защитить важные ветки, установив правила защиты веток, которые определяют, могут ли участники проекта удалять или принудительно отправлять правки в ветку. Для этого установите шаблон для правила защиты ветки, для этого, согласно подсказке, далее укажите необходимые права пользователей для команд push и merge для указанного правила. Для активации правила нажмите кнопку “Включить защиту”.

В списке созданных правил вы можете изменить права доступа или удалить правило.

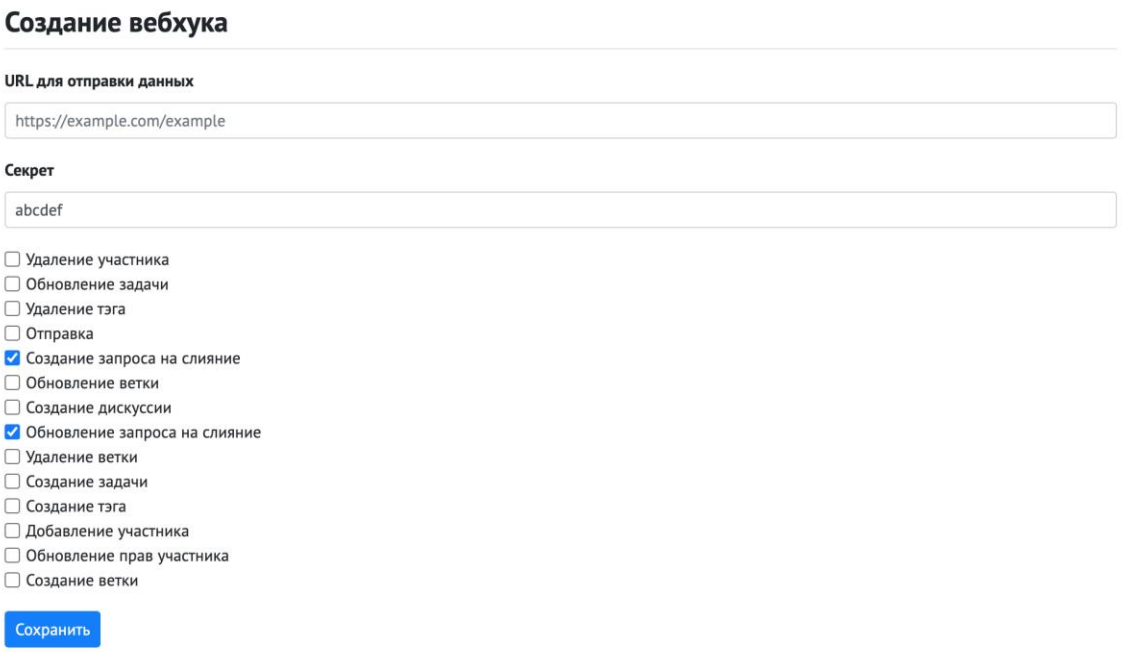


Вебхуки

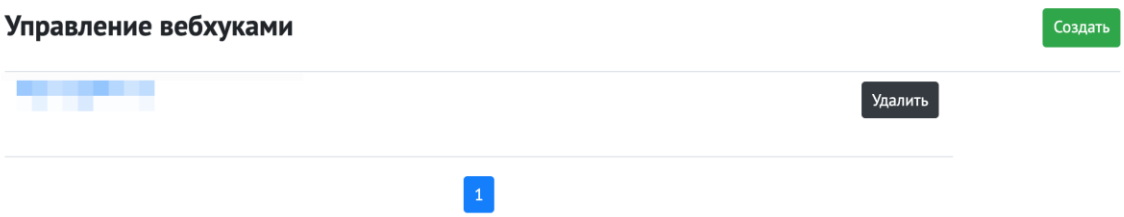
Вебхуки позволяют создавать или настраивать интеграции для приложения OAuth, которые подписываются на определенные события на GitFlic.ru. Когда происходит одно из обозначенных событий, отправляются данные на настроенный URL-адрес вебхука. Вебхуки можно использовать для обновления внешнего сервиса, обновления резервного зеркала или даже для развертывания на рабочем сервере. Вы можете использовать вебхуки любым способом по своему усмотрению.

Для создания нового вебхука нажмите кнопку Создать.

При создании вебхука заполните все поля. Для вебхуков доступны следующие события: Удаление участника, Обновление задачи, Удаление тэга, Отправка, Создание запроса на слияние, Обновление ветки, Создание дискуссии, Обновление запроса на слияние, Удаление ветки, Создание задачи, Создание тэга, Добавление участника, Обновление прав участника, Создание ветки.

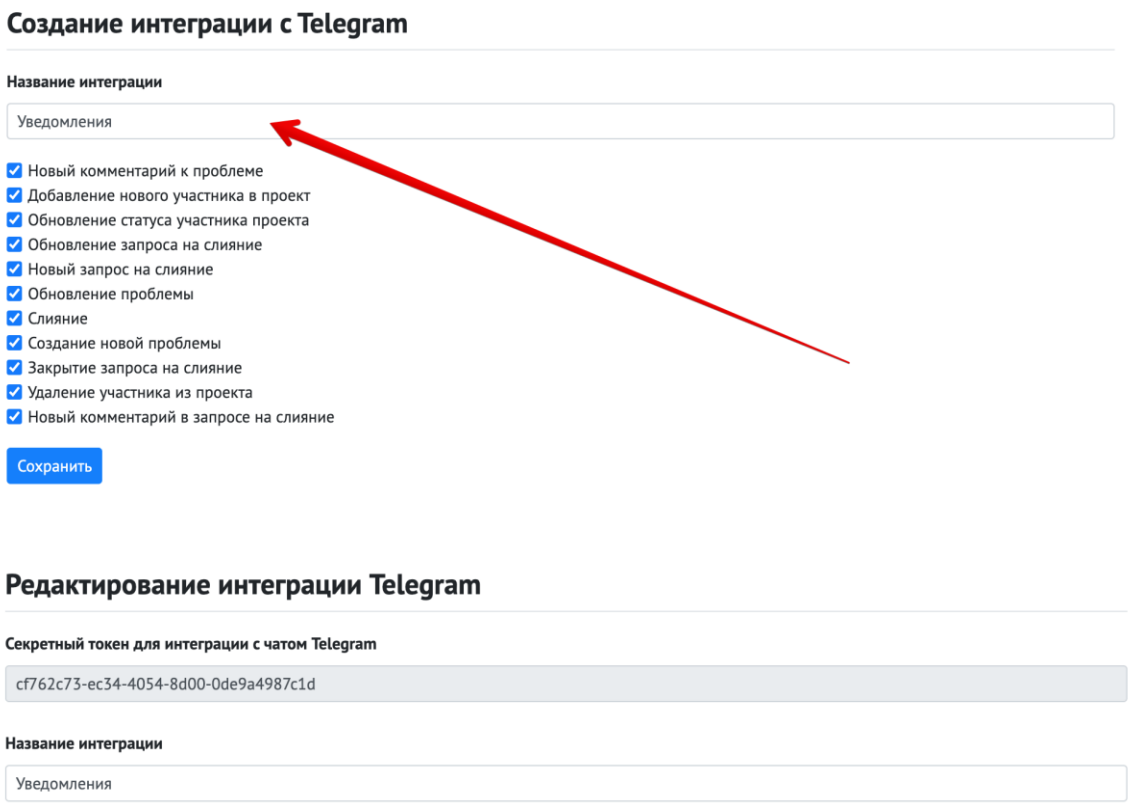


После создания вебхука в проекте, его можно отредактировать, чтобы изменить набор событий или URL для отправки данных.



Интеграция с Telegram

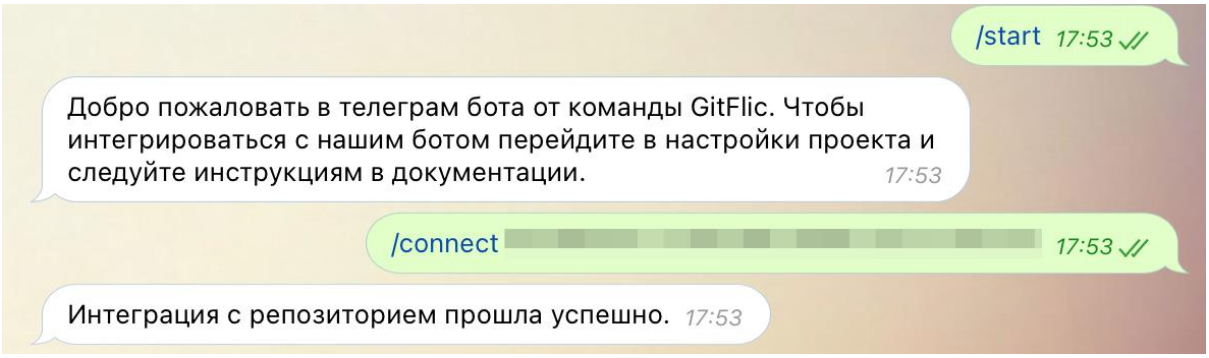
Для создания новой интеграции с Telegram нажмите кнопку “Создать”. На странице создания интеграции выберете необходимый набор событий, который вам требуется для получения уведомлений в чате или канале, укажите название для новой интеграции и нажмите “Сохранить”. После успешного сохранения вам будет доступен секретный токен для интеграции с чатом Telegram.



Для настройки связи GitFlic-Telegram вам необходим GitFlic бот.

Рекомендуем создать отдельный чат для уведомлений о проектах, чтобы важные уведомления не затерялись в переписке.

Далее вам нужно добавить бота в ваш канал или групповой чат вашей команды разработки, для бота достаточно прав только для отправки новых сообщений. Далее необходимо отправить в чат команду бота /start, вам придет обратное сообщение, что бот готов к работе. Чтобы подключить интеграцию с проектом, необходимо скопировать секретный токен из настроект проекта и с командой /connect через пробел отправить в чат. Например, команда для подключения бота может выглядить /connect a1b2c3d4-abcd-1234-5678-05a592e5578f. После создания успешной связи вы получите обратное сообщение, что ваш проект был подключен. А на странице управления интеграцией появится информация о подключенном чате.



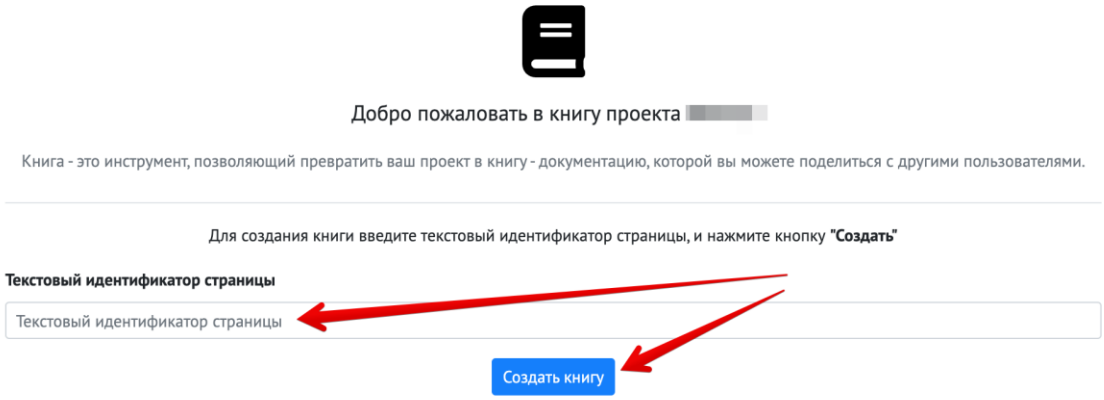
Если добавляете бота в групповой чат, необходимо добавить его в администраторы. При добавлении бота в канал, права администратора будут предложены автоматически. Боту достаточно прав для отправки сообщений.

Внимание! Максимальное количество интеграций 5 т.е один проект можно подписать на 5 telegram-чатов.

Книга проекта

Книга проекта позволяет создать подробную документацию из вашего проекта. Данную документацию можно создать как в виде отдельного репозитория, так и в самом проекте.

Для создания книги перейдите в настройки проекта и откройте вкладку “Книга”. Добавьте текстовый идентификатор для вашей книги и нажмите “Создать”. Вам будет доступна ссылка на вашу книгу, можете добавлять ее к описанию проекта, чтобы пользователи имели прямой доступ к вашей книге.



Для инициализации страницы с книгой в проект необходимо добавить 2 файла: README.md для главной страницы SUMMARY.md для создания оглавления.

Оформление SUMMARY.md

SUMMARY.md наполняется следующим образом: в файле могут присутствовать заголовки и обязательно ссылки на файлы, указанные разметкой Markdown. пример оформления оглавления ниже

```
# Документация

* [Введение](README.md)

## Раздел 1
* [Статья 1](file1.md)
* [Статья 2](file2.md)
* [Подраздел]()
        * [Статья 3](folder/file3.md)
        * [Статья 4](folder/file4.md)
```

Файлы с документацией можно расположить как в корне, так и папке. Путь в ссылке можно указать относительно файла с заголовками, как указано в примере.

Git LFS

Git Large File Storage (LFS) заменяет большие файлы, такие как аудио, видео, наборы данных и графики, текстовыми указателями внутри Git, сохраняя при этом содержимое файла на удаленном сервере.

Для установки можете использовать следующие команды:

- Homebrew: brew install git-lfs
- MacPorts: port install git-lfs
- Для Windows перейдите по ссылке

Загрузите и установите расширение для Git. После загрузки и установки настройте Git LFS для своей учетной записи пользователя, выполнив:

git lfs install

В каждом репозитории, где вы хотите использовать Git LFS, выберите типы файлов, которыми вы хотели бы управлять с помощью Git LFS (или непосредственно отредактируйте .gitattributes). Вы можете настроить дополнительные расширения файлов в любое время.

```
git lfs track "*.psd"
git add .gitattributes
```

Обратите внимание, что указание типов файлов, которые должен отслеживать Git LFS, само по себе не преобразует какие-либо ранее существовавшие файлы в Git LFS, такие как файлы в других ветвях или в вашей предыдущей истории загрузки. Для этого используйте команду git lfs migrate, которая имеет ряд опций, разработанных в соответствии с различными потенциальными вариантами использования.

Для просмотра указанных расширений для Git LFS используйте команду

git lfs track

Далее просто зафиксируйте и отправьте изменения на GitFlic, например, если ваша текущая ветка называется master:

```
git add file.psd
git commit -m "Add design file"
git push origin main
```

Действующие ограничения на репозиторий

- Стандартный размер репозитория 4Гб
- Размер инициирующего коммита 1Гб
- Стандартный размер коммита 100Мб
- Нельзя загрузить файл больше текущего размера репозитория (если проект 100Мб, то нельзя загрузить файл более 100Мб)
- Размер файла не может превышать 2Гб

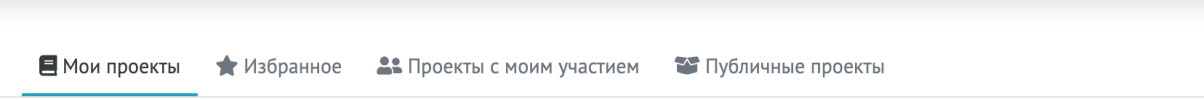
В данный момент тарификация для LFS находится в разработке, если у вас возникли трудности с ограничениями, обратитесь в поддержку.

Профиль

Список проектов

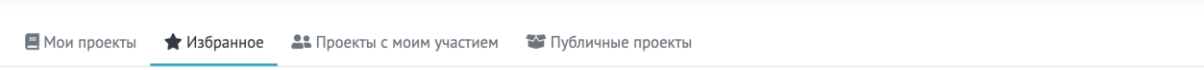
Список проектов пользователя

В данном списке отображаются все проекты, созданные пользователем, в которых он является непосредственным владельцем



Избранные проекты

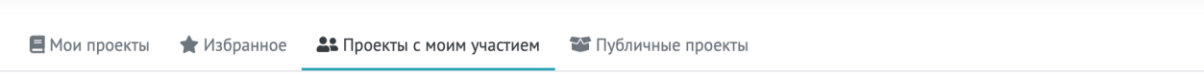
Здесь проекты, которые отметил пользователь. Для добавления в избранное нажмите на иконку звезды рядом с названием проекта.



Подробнее об избранном по ссылке

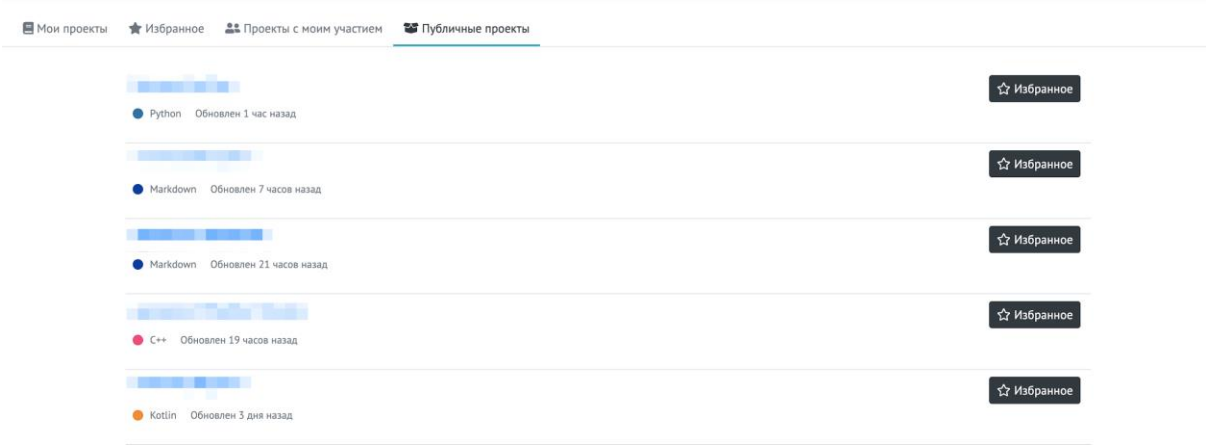
Проекты с доступом пользователю

В данном списке отображаются проекты, в которые был приглашен пользователь. Здесь также все проекты, которые создал пользователь на компанию или на команду, в которых состоит пользователь



Список публичных проектов

На этой вкладке отображены публичные проекты всех пользователей сервиса. Для поиска по проектам воспользуйтесь поиском по сервису.



Страница профиля

Readme профиля

О README для профиля

Вы можете поделиться информацией о себе, создав README для вашего профиля. README будет показан в верхней части страницы вашего профиля.

Вы сами решаете, какую информацию включать в свой профиль README, поэтому у вас есть полный контроль над тем, как вы представляетесь на GitFlic. Вот несколько примеров информации, которую посетители могут найти интересной, забавной или полезной в вашем профиле README.

- Раздел “Обо мне”, в котором описывается ваша работа и интересы
- Проекты, которыми вы гордитесь
- Какая-то другая информация, которая может быть интересна остальным пользователям

Требования

Мы сможем отобразить README вашего профиля на странице вашего профиля, только если все следующие условия верны.

Вы создали репозиторий с именем, совпадающим с вашим именем пользователя Gitflic. Репозиторий публичный. Репозиторий содержит файл с именем README.md в корне. Файл README.md содержит любой контент. Создание README для профиля В правом верхнем углу любой страницы нажмите “+” и создайте новый проект.

Создание проекта

В разделе «Название проекта» введите имя проекта, соответствующее вашему имени пользователя GitFlic. Например, если ваше имя пользователя «gitflicuser», имя репозитория должно быть «gitflicuser».

По желанию, добавьте описание вашего проекта. Ниже выберете чекбокс “Публичный проект” и нажмите “Создать проект”.

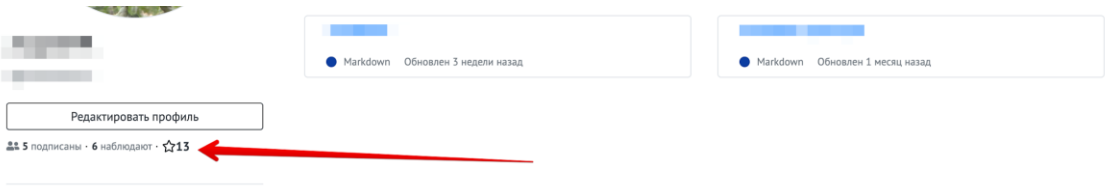
Далее склонируйте репозиторий или создайте подключение к удаленному репозиторию. Создайте в корневой папке файл README.md и заполните по своим предпочтениям, используя Markdown. Сделайте новый коммит и запушьте изменения в удаленный проект.

Если все шаги выполнены верно, README будет отображен в вашем профиле на вкладке “Обзор”.

Подписки и подписчики

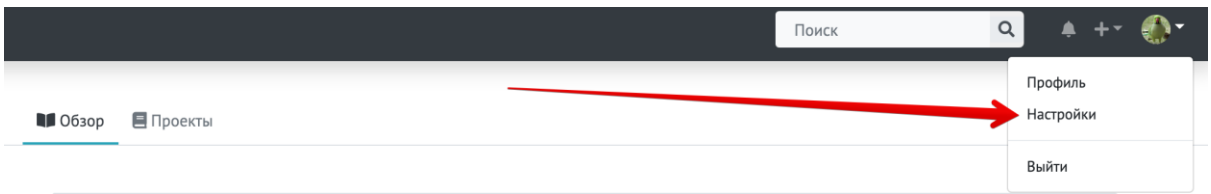
В GitFlic Вы можете добавить себе в подписки других пользователей, чей профиль стал вам интересен, либо это может быть ваш коллега или друг. Аналогично, на вас могут подписываться другие пользователи сервиса.

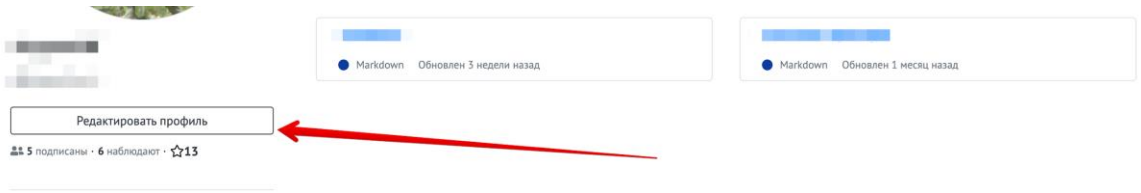
Чтобы посмотреть подписки или подписчиков, на странице профиля на вкладке “Обзор” под аватаром находятся блок с подписчиками.



Настройки профиля

В настройки профиля можно попасть 2-мя путями. Либо через выпадающее меню в верхнем правом углу нажать “Настройки”, либо перейдите на страницу профиля и под аватаром нажмите кнопку “Редактировать”

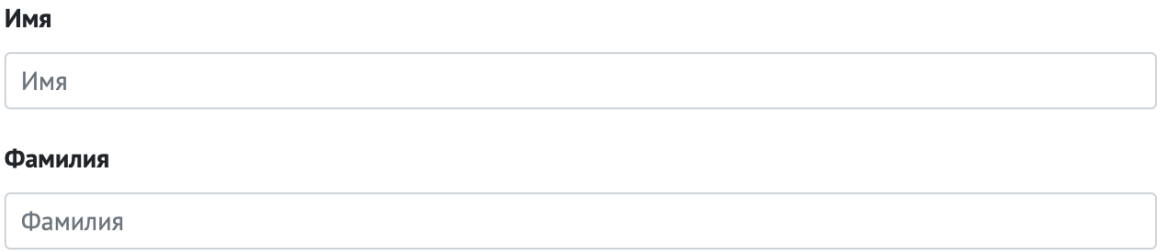




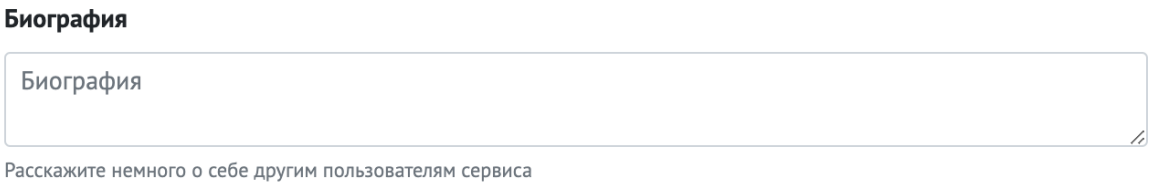
Данные профиля

Вы сами решаете, какую информацию включать в свой профиль, чтобы гости имели представление о вас на GitFlic.

Смена имени и фамилии находится на главной странице настроек профиля.



Биография - это описание вашего профиля. Приветствуется описание вашего опыта, в каких компаниях вы работали, чем занимались, что освоили.



Если у вас есть персональный сайт, можете указать URL к нему, либо оставить ссылку на ваши соц сети.

Можете указать название компании, в которой работаете или которую представляете.

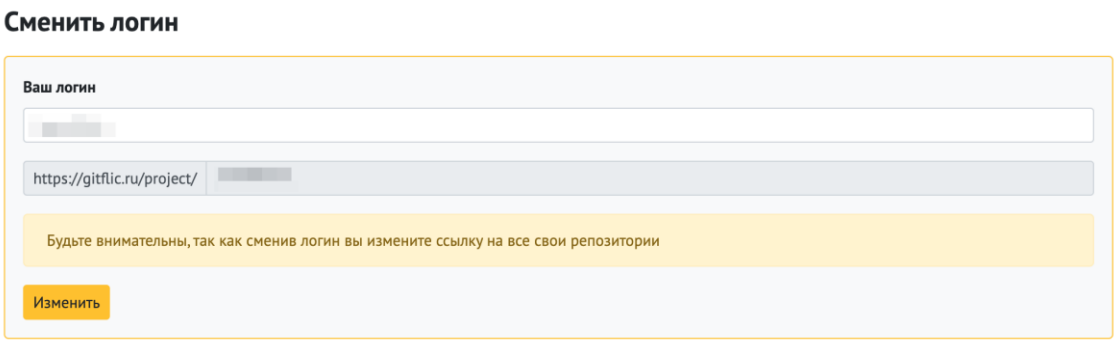
Незаменимым атрибутом профиля является аватар. Добавьте изображение, чтобы придать уникальности. Приветствуются как личные фото, так и отдельно взятые изображения.

Также вы можете сделать README для вашего профиля, ознакомьтесь со статьей о README профиля

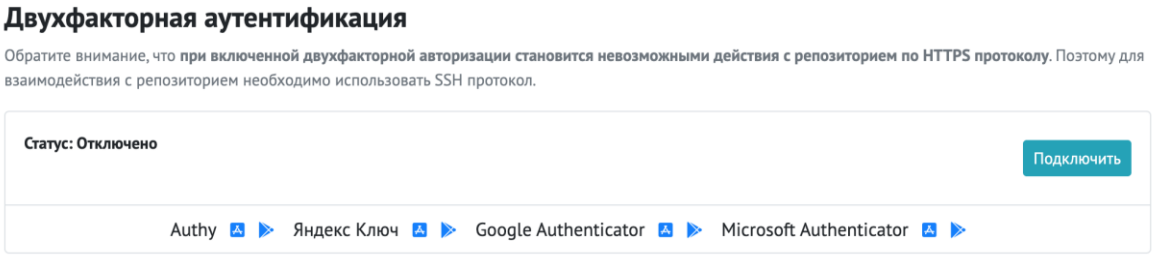
Аккаунт

Настройка аккаунта производится в меню настроек.

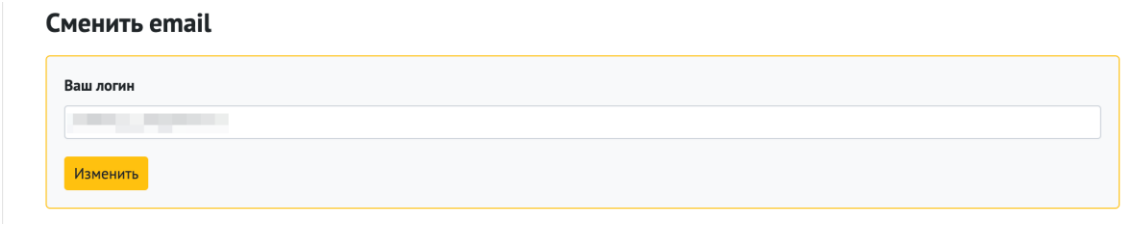
Вы можете заменить текущее имя пользователя на любое свободное, если желаемое имя занято, то вы можете использовать альтернативное написание, например, с использованием чисел. Следует учитывать, что после сохранения нового логина, предыдущий URL профиля будет недоступен, так как он будет изменен в соответствии с новым именем профиля. Если вы являетесь владельцем товарного знака для имени пользователя, вы можете запросить дополнительную информацию о подаче жалобы на использование товарного знака через поддержку сервиса.



Двухфакторная аутентификация позволяет защитить ваш аккаунт при помощи приложений генерации временных секретных кодов. Пройдите процесс подключения согласно инструкции при помощи вашего телефона.



Смена почты профиля производится на данной странице в 2 этапа: укажите новую почту для вашего профиля, перейдите по ссылке из письма для подтверждения новой почты профиля.



SSH-ключи

SSH ключи позволят установить защищенное соединение между вашим компьютером и сервисом.

В GitFlic есть возможность подключить аутентификацию по публичному ssh-ключу. Вставьте ваш публичный SSH ключ, который обычно содержится в файле ‘~/.ssh/id_ed25519.pub’ или ‘~/.ssh/id_rsa.pub’ и обычно начинается с ‘ssh-ed25519’ или ‘ssh-rsa’. Не вставляйте свой приватный SSH ключ, так как это может скомпрометировать вашу личность.

Если у вас нет ssh-ключа, создайте ключ следующей командой:

ssh-keygen -t ed25519 -C "your_email@gitflic.ru".

На консоль будет выведен следующий диалог (или аналогичный):

Enter file in which to save the key (/home/user/.ssh/id_ed25519):

Нажмите на клавишу Enter.

Система предложит ввести кодовую фразу для дополнительной защиты SSH-подключения. (Данный шаг можно пропустить.) Enter passphrase (empty for no passphrase):

После этого ключ будет создан и помещён в директорию /home/user/.ssh/

1. Зайдите в папку /home/user/.ssh и скопируйте содержимое файла id_ed25519.pub.
2. Перейдите в GitFlic, далее в настройки профиля. На вкладке Ключи в поле Ключ вставьте скопированный SSH-ключ.
3. Дайте название ключу и выберете дату окончания действия ключа (оставьте поле пустым, чтобы ключ не имел срока годности).
4. Нажмите кнопку добавить.

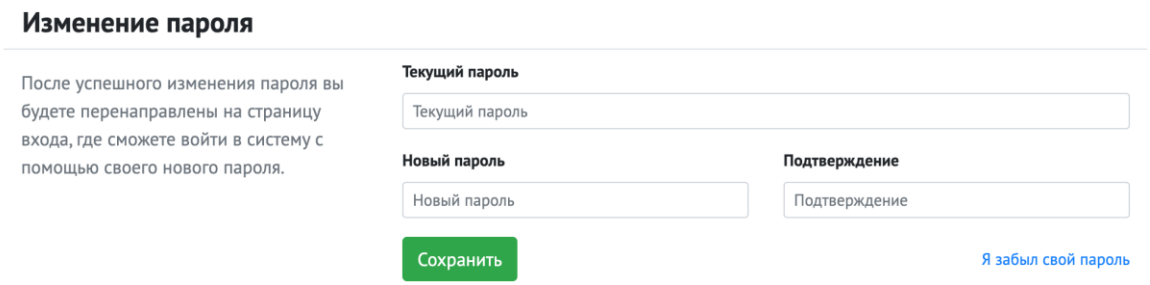
Запись о сохраненных ключах будет отображена на текущей странице.

Для получения отпечатка вашего публичного ключа используйте команду ssh-keygen -E sha256 -l -f id_rsa_3.pub.

Смена пароля профиля

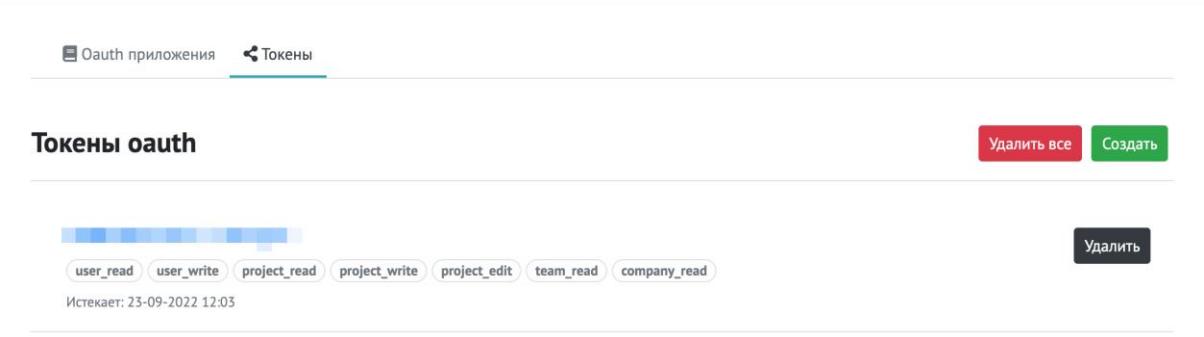
Вы всегда можете заменить ваш пароль на новый. Для этого укажите старый пароль, введите новый и повторите новый пароль. Либо можете заказать письмо на восстановление пароля, нажав кнопку “Я забыл свой пароль”.

Рекомендуем не использовать ваши шаблонные пароли, добавьте несколько чисел или ассоциацию с сайтом, это повысит его надежность.



Создание токена oauth

Вы можете создать Oauth токен для приложения с необходимыми правами, либо создать токен пользователя для отладки вашего приложения. Отличие токенов заключается в сроке действия токена: токен пользователя активен только 1 месяц, а токен для приложения не имеет срока годности.



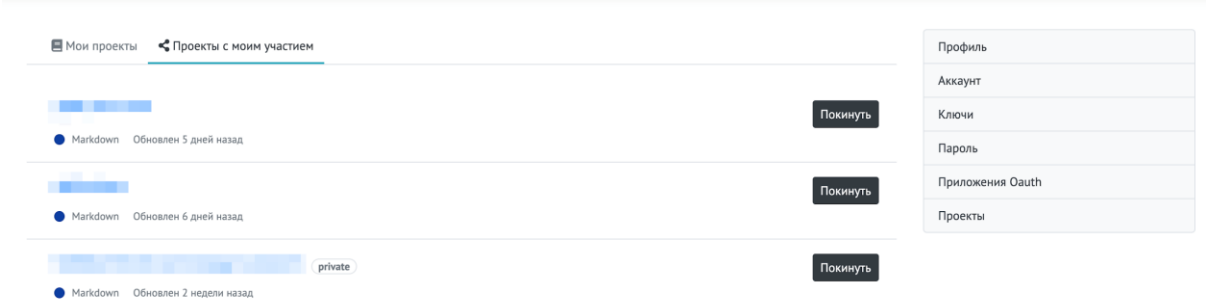
Подробную информацию как получить токен для приложения вы можете получить по ссылке

Управления проектами

На вкладке с проектами выведен список ваших проектов, которые вы можете удалить, быстрее чем если вы будете удалять их по отдельности, переходя в настройки каждого из этих проектов.

На соседней вкладке “Проекты с моим участием” отображены проекты, в которых вы принимаете участие. Вы можете выйти из числа участников проектов, нажав на кнопку “Покинуть”.

Обратите внимание, удаление и выход - необратимые действия. Если вы выйдете из проекта, вам потребуется попросить администратора проекта включить вас обратно в число участников.

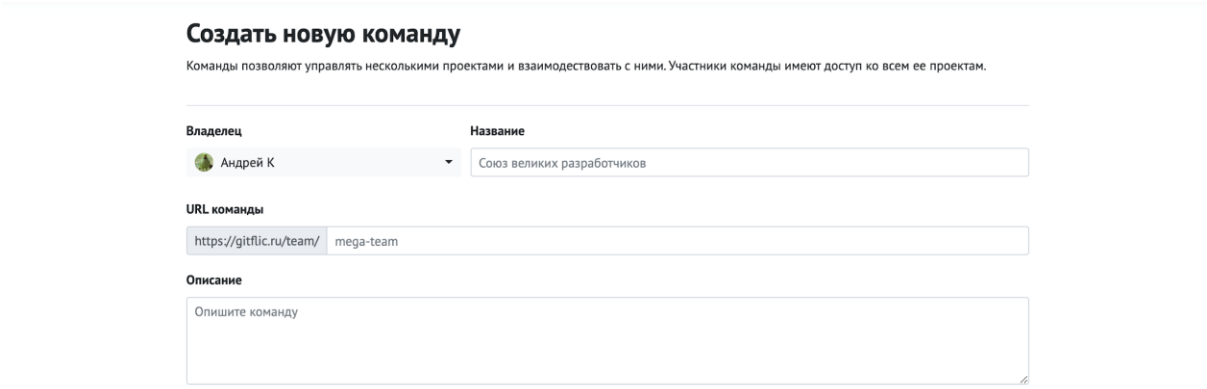


Команды

Создание команды

Команды - это группа пользователей, которые объединены общими целями и/или задачами по совместным проектам.

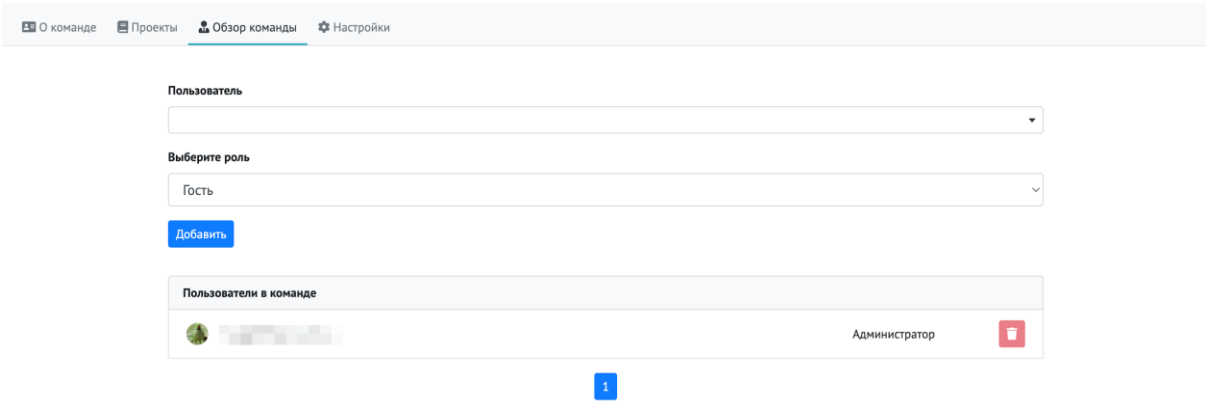
Для создания новой команды откройте меню создания в верхнем правом углу и перейдите к созданию команды. Выберете для вашей новой команды уникальное название и укажите приватность новой команды. Владельца вы можете указать себя, либо компанию.



Обзор команды

Пользователи, не состоящие среди участников публичной команды могут посмотреть публичные кпроекты компании и ознакомиться с readme команды, также посмотреть состав участников.

Если вы обладаете правами владельца или администратора, вы можете добавлять новых участников в команду. Для этого перейдите на вкладку “Обзор команды”. При помощи поля поиска найдите пользователя в сервисе и укажите его роль в команде, определяющую определенные права доступа. После приглашения нового пользователя в команду, ему на почту придет приглашение от команды, пользователь должен подтвердить свое согласие на вступление переходом по ссылке-приглашению.



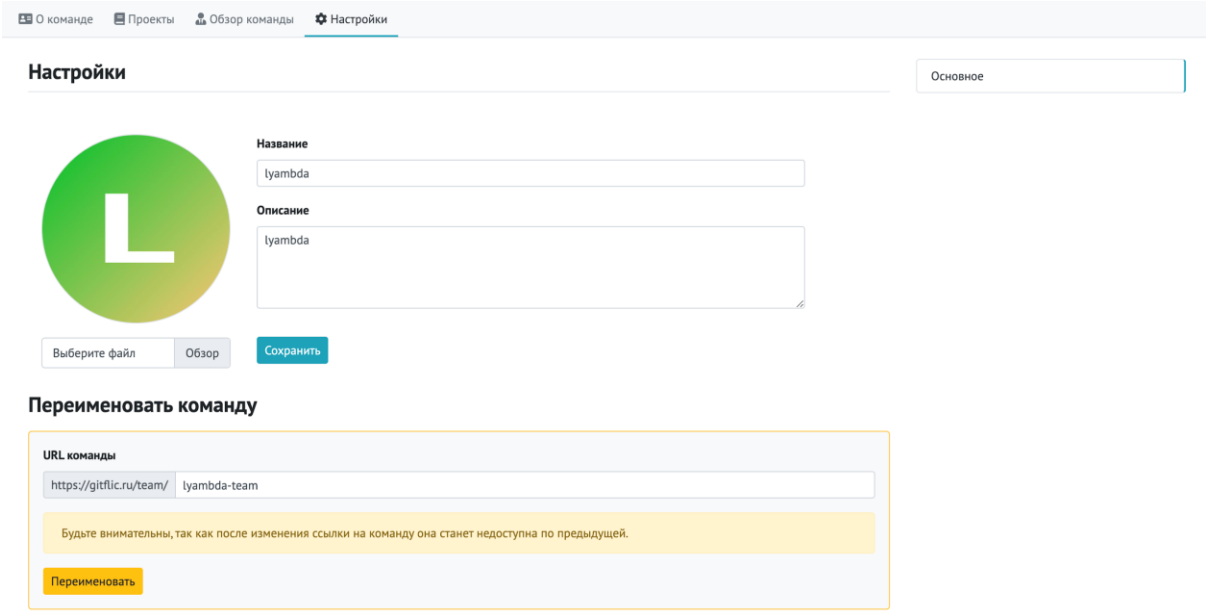
Удаление членов команды осуществлется на вкладке с участниками команды. Напротив пользователя расположена красная кнопка удаления из команды.

Для SaaS - в сервисе существует ограничение на количество участников внутри команды. Бесплатно можно пользоваться сервисом, пока не превышено ограничение 5 пользователей в рамках одной команды/проекта, после превышения лимита предлагается создать компанию и перейти на платный тариф. см. соглашение и тарифы

Настройки команды

При наличии прав администратора команды, вам доступен раздел с настройками в команде.

Сотрудник компании с правами администратора может вносить изменения в профиль компании. Для этого откройте настройки, вам доступны изменение названия, описания, также изменить контактный телефон, адрес электронной почты и ссылку на сайт вашей компании. Также есть возможность изменить URL, держите во внимании, что после смены URL старый адрес будет недоступен, также он будет свободен для всех остальных пользователей GitFlic. Администратор компании может изменить видимость компании.



Для создания проектов от имени команды, необходимы права администратора команды.

Readme команды

Вы можете поделиться информацией о том, как взаимодействовать с вашей командой, создав README команды. Gitflic показывает README профиля вашей организации на вкладке «Обзор».

Вы сами выбираете, какую информацию включать в README. Вот несколько примеров информации, которая может оказаться полезной.

- Раздел «Об организации», описывающий вашу организацию.
- Руководство для получения помощи в организации.

Создание README организации

- Создайте публичный проект, который будет иметь название, совпадающее с названием вашей организации в Gitflic. Подробнее о создании проектов можно прочитать здесь.
- Склонируйте или создайте удаленное подключение к созданному проекту.
- В корне проекта создайте файл README.md и заполните его по своему усмотрению, используя Markdown.
- Сделайте новый коммит и запушьте изменения в проект.

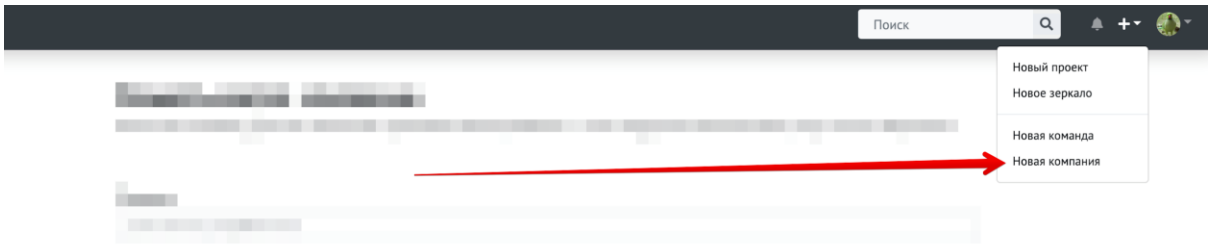
Если все шаги выполнены верно, README вашей организации будет отображен на вкладке “Обзор компании”.

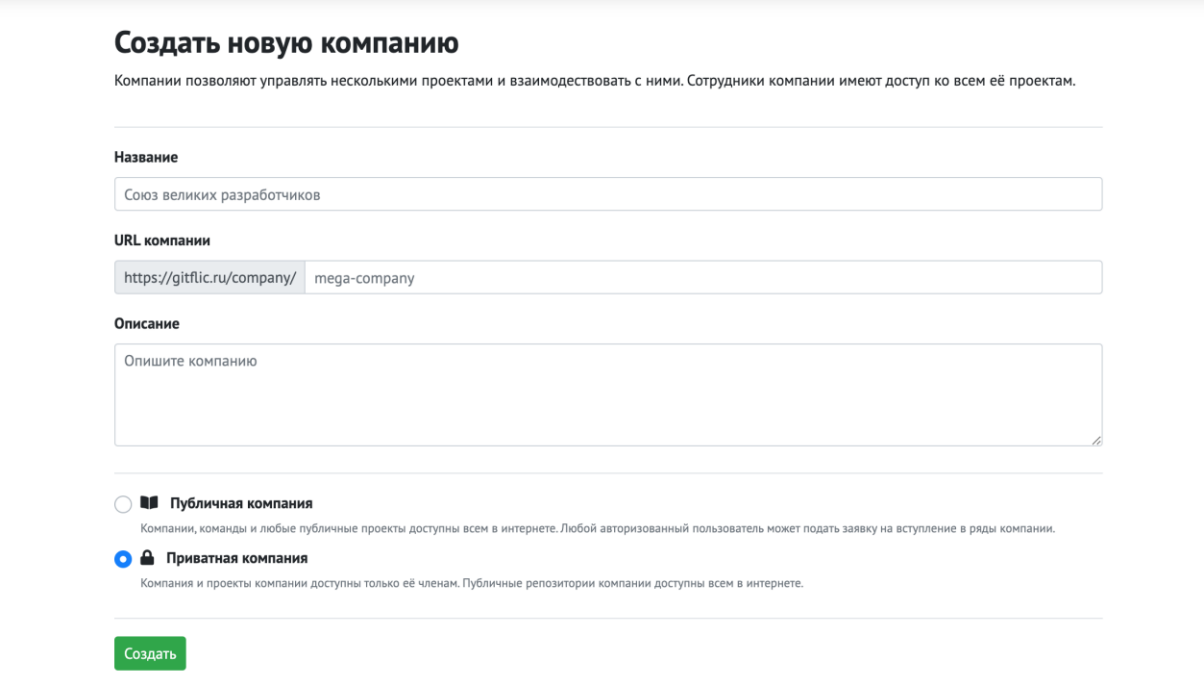
Компании

Создание компании

Компания - это обобщенная учетная запись, в которой собраны пользователи и команды, ведущие совместную работу над проектами. Администраторы и владелец (пользователь GitFlic) могут управлять доступом участников компании к проектам компании.

Для создания новой команды откройте меню создания и выберете пункт создания новой команды. На странице создания требуется указать базовую информацию: название, описание и тип приватности. URL компании заполняется автоматически из названия, но его, при желании, можно указать самостоятельно.



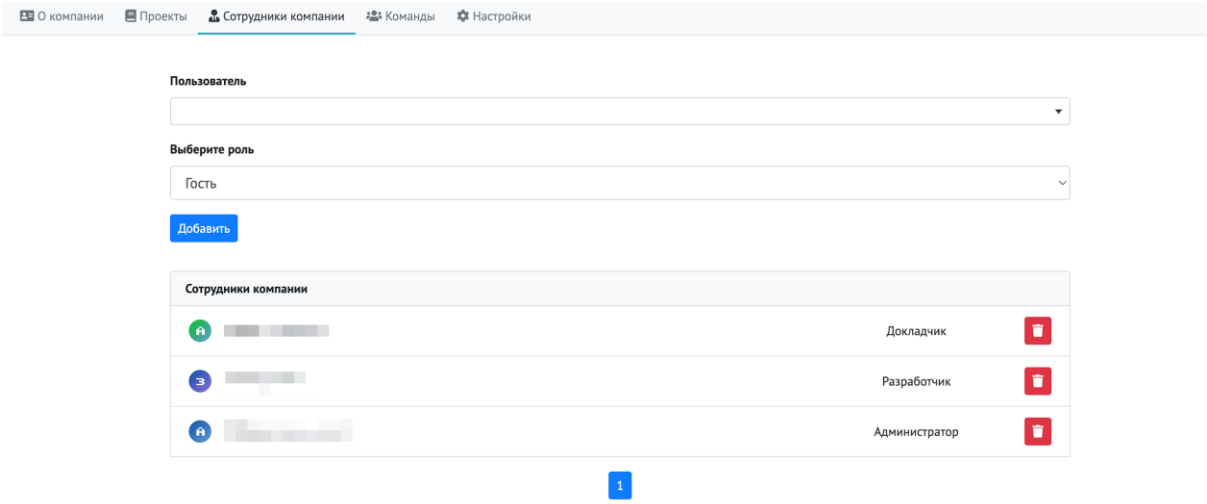


Обзор компании

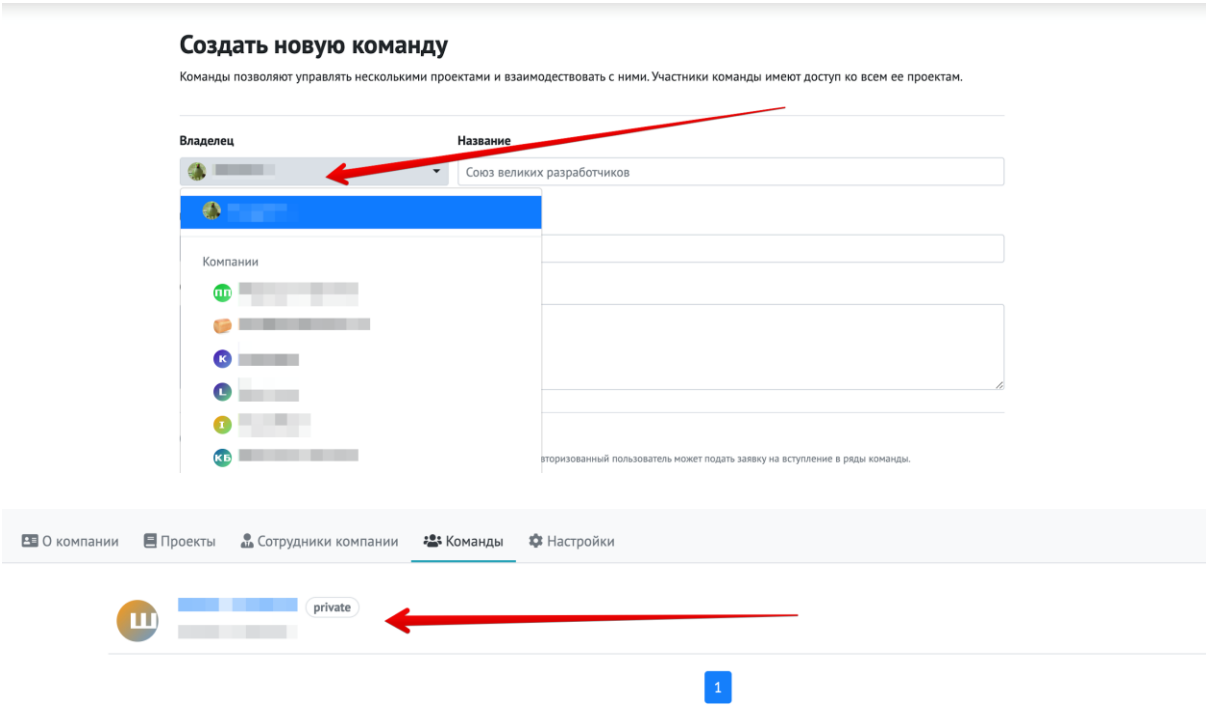
При просмотре компании, пользователь может просматривать сотрудников и команды компании, публичные проекты компании, а также readme компании, если он имеется.

Добавление сотрудников

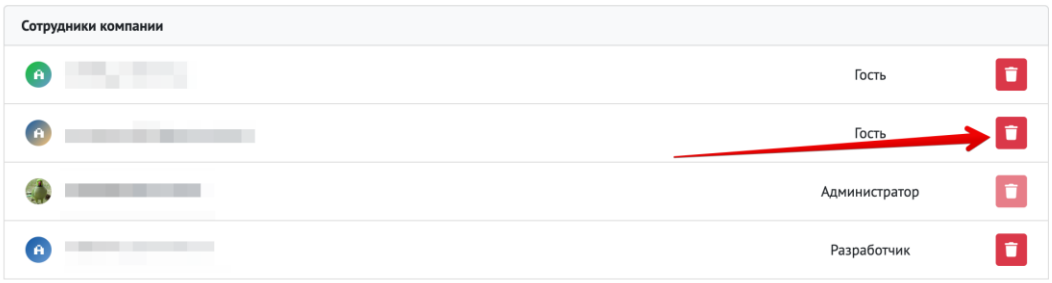
Приглашайте пользователей в компанию напрямую или через команды. Для добавления пользователей пройдите на вкладку Сотрудники компании. Выберете пользователя GitFlic через поле поиска и назначьте роль, далее нажмите “Добавить”. Новый сотрудник появится среди остальных сотрудников (для данного действия требуются права администратора).



Для добавления команды к компании перейдите к созданию новой команды и владельцем укажите вашу компанию. После создания команды добавьте в число участников пользователей GitFlic.



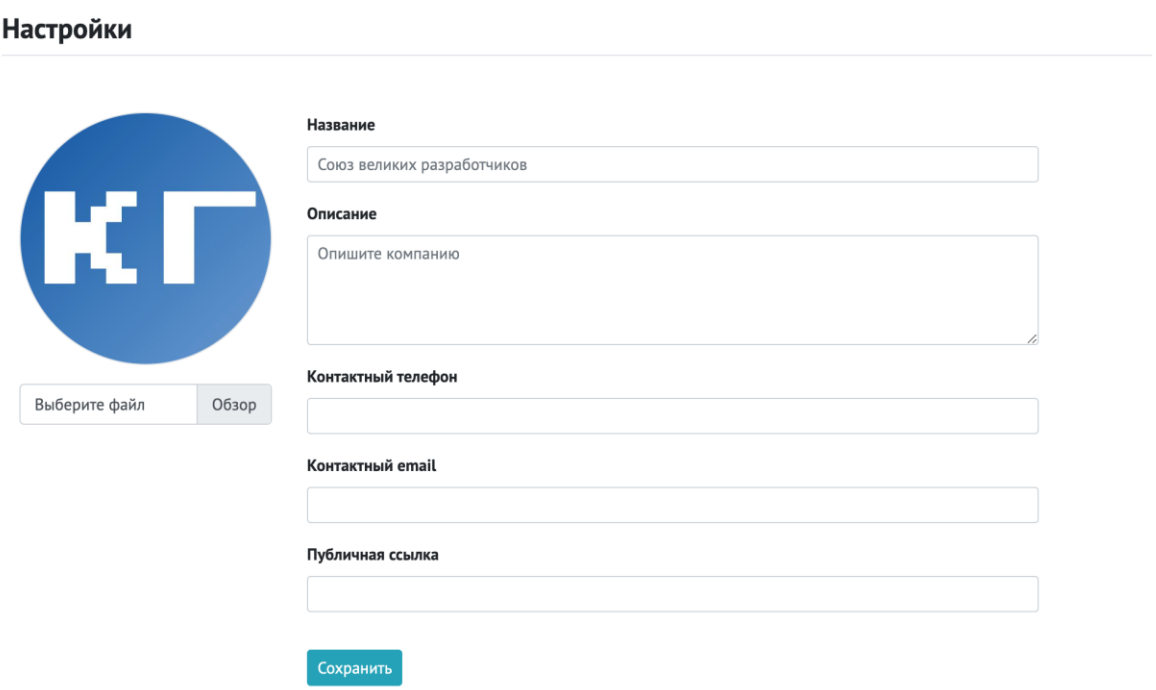
Удаление сотрудников из компании или команды внутри компании доступно только с правами администратора компании. Для этого откройте список участников компании или команды и напротив имени пользователя нажмите кнопку удалить. Стоит учитывать, что удаленный пользователь потеряет доступ к проектам компании, но у него могут остаться локальные копии проектов, с которыми он работал.



Настройки компании

При наличии прав администратора компании, вам доступен раздел с настройками в компании.

Вы можете изменить аватар в компании, изменить название и описание. Вам доступна смена адреса компании, а также изменение видимости и возможность удалить компанию.



Для создания проектов от имени компании, необходимы права администратора компании.

Readme компании

Вы можете поделиться информацией о том, как взаимодействовать с вашей организацией, создав README профиля организации. Gitflic показывает README профиля вашей организации на вкладке «Обзор».

Вы сами выбираете, какую информацию включать в README. Вот несколько примеров информации, которая может оказаться полезной.

- Раздел «Об организации», описывающий вашу организацию.
- Руководство для получения помощи в организации.
- Некоторые публичные контакты, для связи с сотрудниками организации

Создание README организации

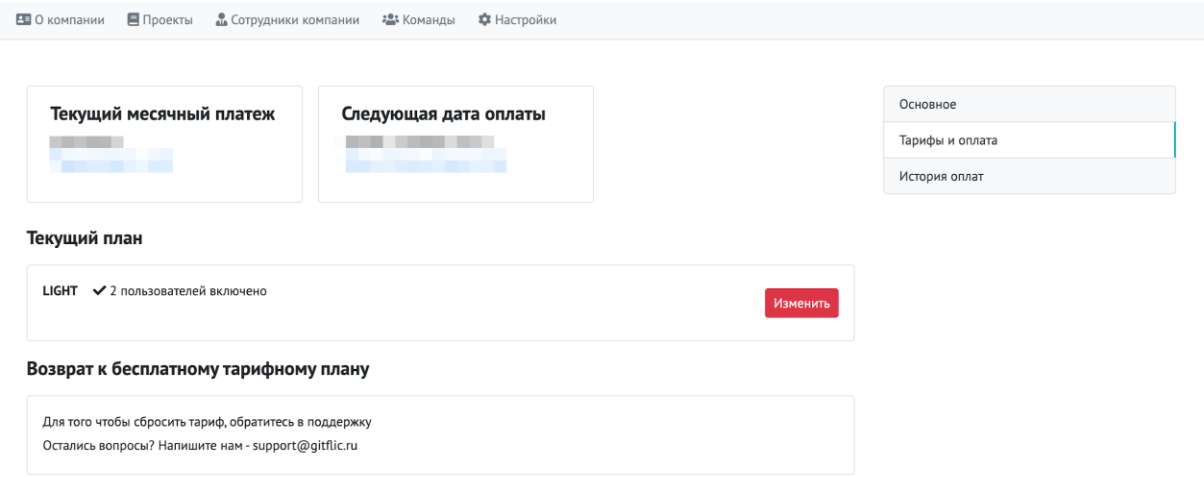
- Создайте публичный проект, который будет иметь название, совпадающее с названием вашей организации в Gitflic. Подробнее о создании проектов можно прочитать здесь.
- Склонируйте или создайте удаленное подключение к созданному проекту.
- В корне проекта создайте файл README.md и заполните его по своему усмотрению, используя Markdown.
- Сделайте новый коммит и загрузите изменения в проект.

Если все шаги выполнены верно, README вашей организации будет отображен на вкладке “Обзор компании”.

Страница тарифов и оплаты (Для SaaS)

Вы можете управлять тарифом вашей компании. По умолчанию в компании действует бесплатный тариф, который включает в себя доступ до 5 участников (включая владельца).

На странице выбора тарифов выберете подходящий, укажите платежные данные и нажмите “Подключить оплату”, вас переведет на страницу оплаты банковской картой. После подключения оплаты, на странице тарифов будет отображена информация о следующей дате оплаты, о количестве оплаченных мест для участников компании и возможность отредактировать количество мест для сотрудников компании.



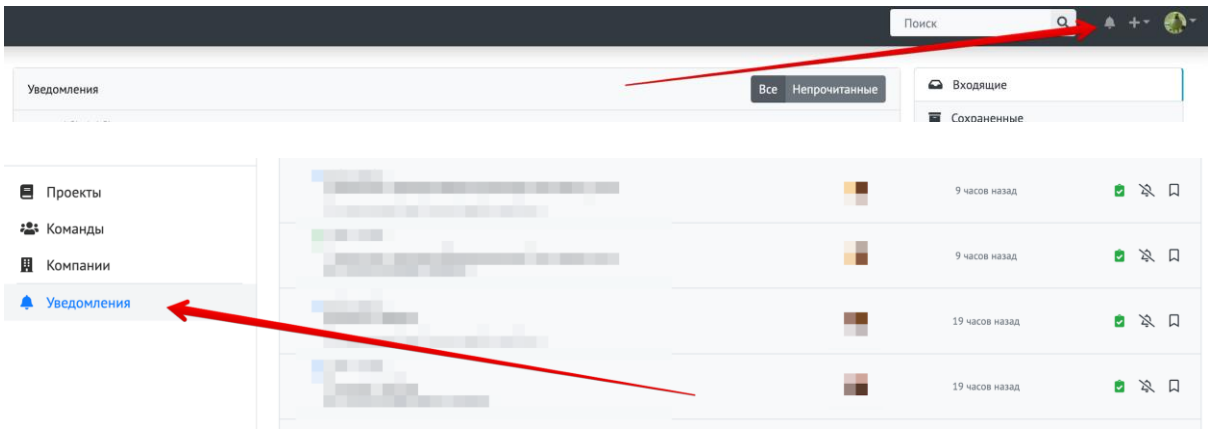
Для администратора компании доступна страница “Истории оплат”, где расположена информация о совершенных платежах.

Если у вас остались вопросы по оплате и тарифам, напишите нам на support@gitflic.ru

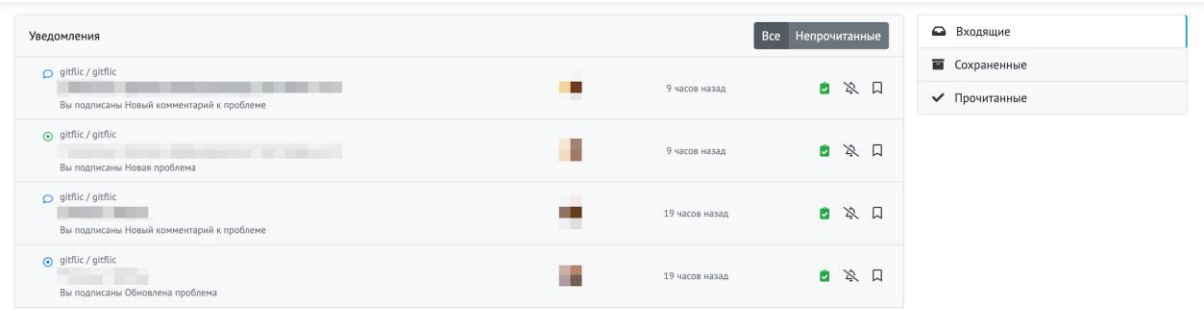
Уведомления

Уведомления

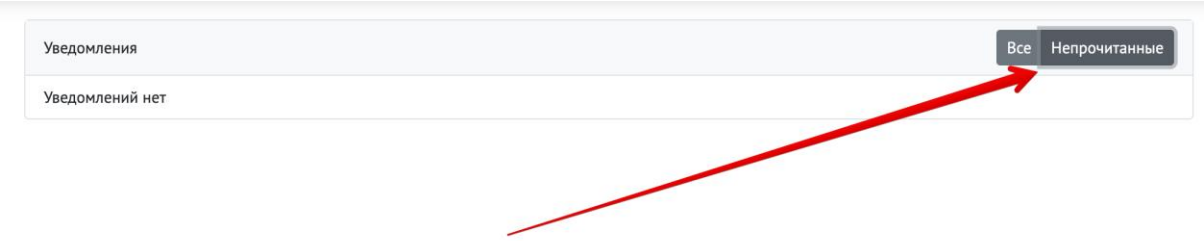
В GitFlic реализована система внутренних уведомлений для пользователей об изменениях в проектах, приглашениях и прочих действиях внутри сервиса. На страницу с уведомлениями можно попасть нажав на кнопку с колокольчиком в левом меню, либо на кнопку в верхней панели левее кнопки профиля.



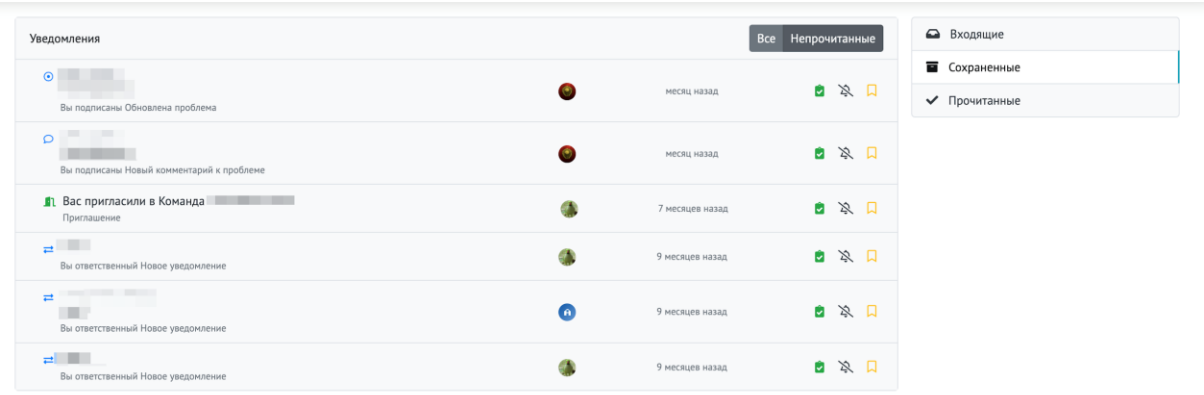
На основной странице уведомления отображены все уведомления, которые направлены пользователю, они расположены в хронологическом порядке от новых к старым. Уведомления имеют различные типы: приглашения, оповещения о действиях в проекте.



На вкладке “Прочитанные” отображены уведомления, которые пользователь прочитал ранее. Также имеется фильтр для отображения только непрочитанных уведомлений.



Пользователь имеет возможность сохранить для себя важные уведомления, нажав на соответствующую кнопку напротив уведомления.



Меню управления подпиской на проект находится на странице с проектом. Нажмите на кнопку “Наблюдать”, перед вами откроется меню, в котором вы можете выбрать нужный тип уведомлений - все уведомления по действиям в проекте, либо действия, которые прямо связаны с вашим профилем (упоминания и назначения), третьей опцией вы можете отключить все уведомления по проекту.

CI/CD

Общая информация

GitFlic CD/CD - это инструмент для организации процессов непрерывной интеграции и доставки кода. С помощью данного инструментария можно на ранних этапах обнаруживать ошибки, тестировать функционал в рабочем окружении, производить сборку вашего кода в готовые приложения.

Концепции

GitFlic CD/CD реализует несколько концепций для описания и выполнения сборки, а именно:

Концепция	Описание
Конвейер (Pipeline)	Структурирование задач через определение стадий и последовательности
Задача (Job)	Фундаментальная структурная единица конфигурации CI/CD
Артефакты (Artifacts)	Файлы, которые являются результатом выполнения задания, подлежащие длительному хранению
Кэш (Cache)	Хранение файлов, например зависимостей сборки, в рамках рабочего окружения агента
Агент (Runner)	Приложение, которое выполняет задачи.

Задача

Введение

Конфигурация конвейеров начинается с Задачи (Job). Задача является фундаментальной частью конфигурации gitflic-ci.yaml файла.

Задачи имеют следующие базовые свойства:

- ограничений для запуска задачи;
- задача должна содержать как минимум один элемент с именем script;
- количество задач в рамках конвейера не ограничено.

Пример:

```
job 1:
  script: echo "Привет всем"

job 2:
  script: echo "И хорошего настроения"
```

В данном примере приведены самая простая конфигурация конвеера с двумя задачами, которые называются job 1 и job 2 соответственно. Конечно, вы можете выполнять любые команды, которые можете выполнить в терминале операционной системы, например yarn install, ./test.sh и так далее.

Задачи запрашиваются агентом и выполняется в окружении конкретного агента, который взял ее в работу. Так же важно отметить, что каждая задача выполняется независимо от всех остальных задач конвейера.

Ограничения для названий задач

Задачи не могут быть названы следующими словами, в силу того, что данные слова являются ключевыми для конфигурации gitflic-ci.yaml файла.

- image
- stage
- stages
- after_script
- before_script
- job
- scripts
- artifacts
- needs
- tags
- only
- except
- allow_failure

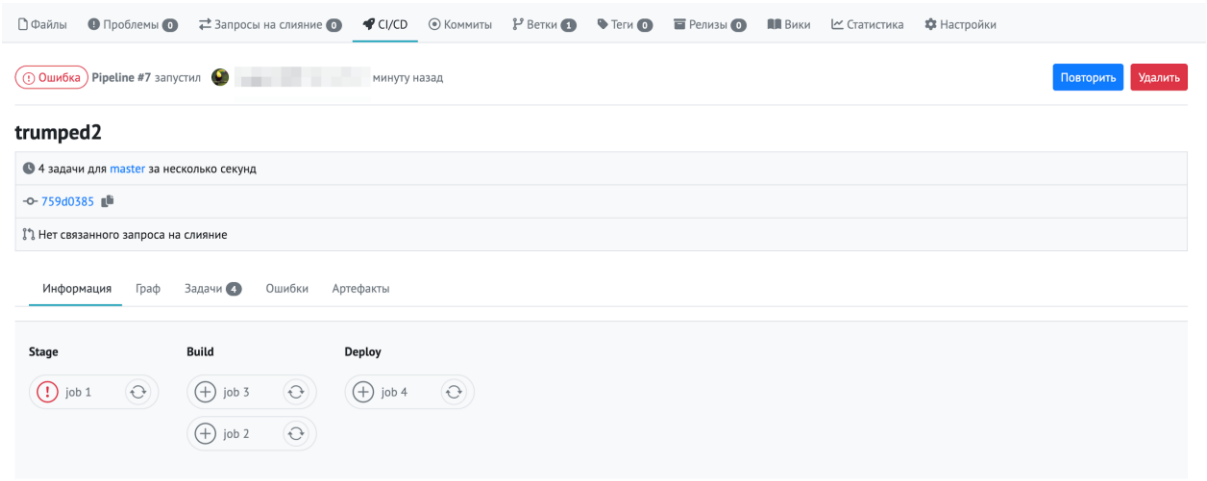
Используйте уникальные названия задач, потому что если в конвейере будет несколько задач с одинаковым именем, то в сам конвейер будет добавлена только одна задача.

Просмотр задач

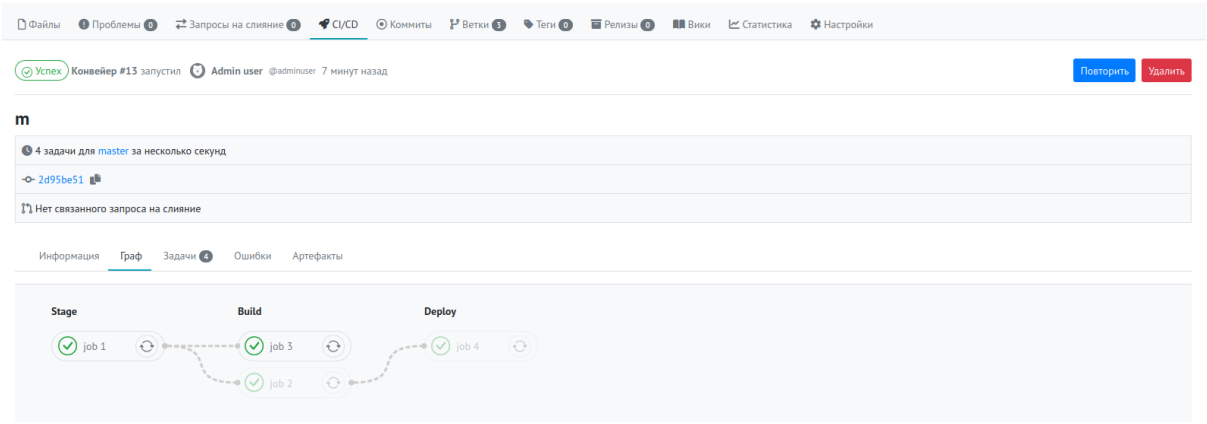
Чтобы перейти к просмотру сведений о конвейере, нажмите на номер конвейера. Вас переведет на страницу задач этого конвейера.

В шапке на странице сведений указана ветка, по которой работает конвейер, время выполнения, ссылка на связанный коммит и информация о запросе на слияние. На

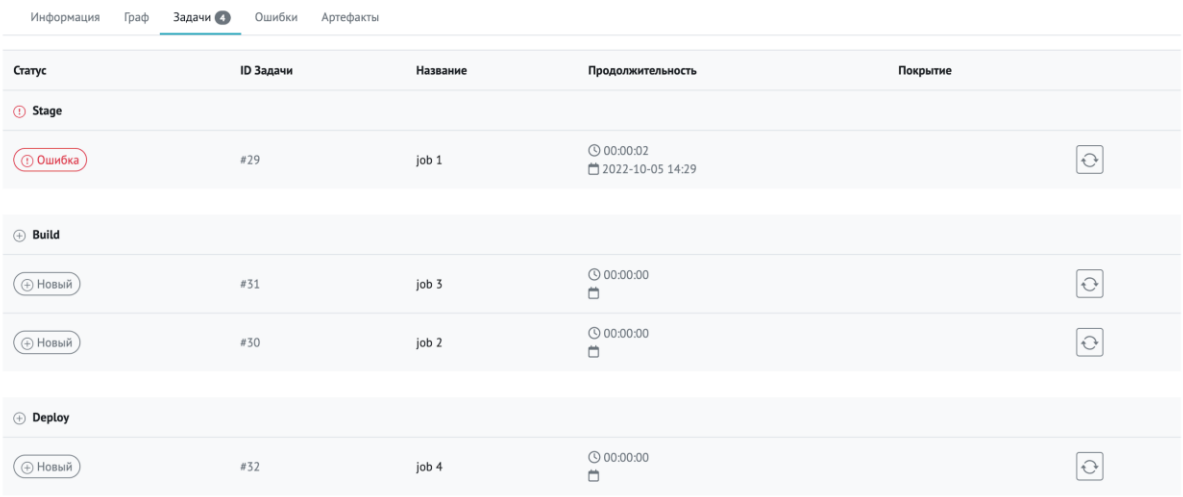
вкладке с информацией отображены все задачи, которые должны быть выполнены в рамках конвейера. Рядом с каждой задачей отображен её текущий статус.



На вкладке с графом изображены последовательные зависимости между задачами. Другими словами, линиями указаны задачи, выполнение которых необходимо для начала выполнения следующей. Задачи отображены аналогично вкладке с информацией: рядом с названием отображен текущий статус и кнопка перезапуска задачи.

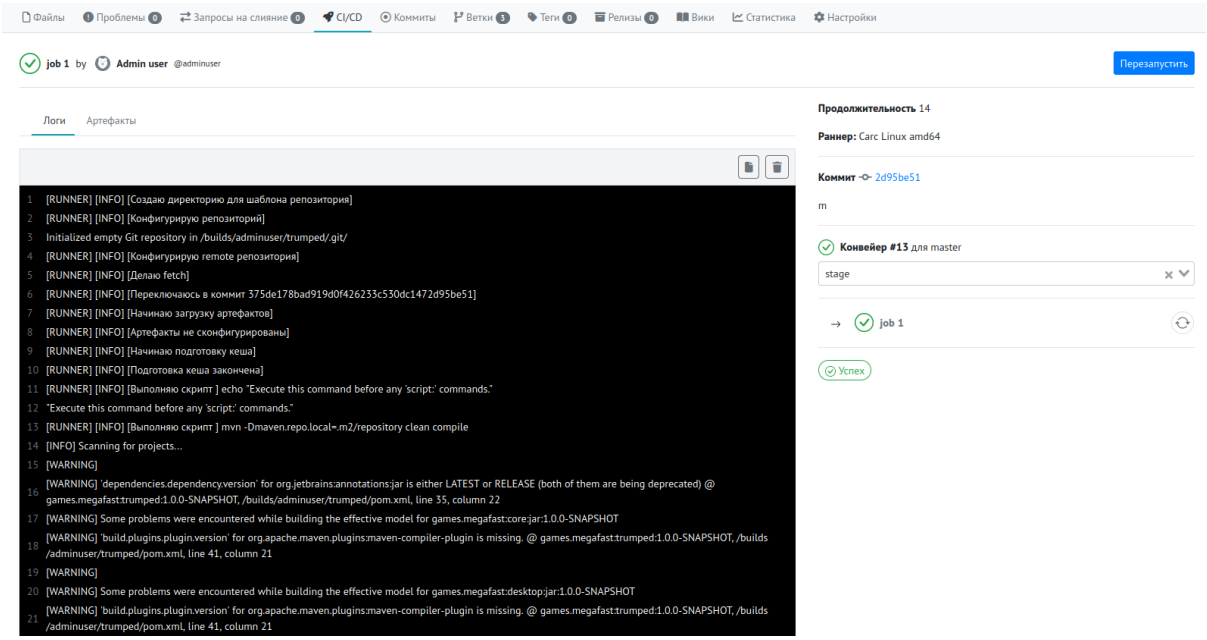


На вкладке с задачами отображен список всех задач. Представлена информация о времени выполнения задачи и времени начала выполнения. Также на вкладке представлена информация о статусе, стадии и имени задач.

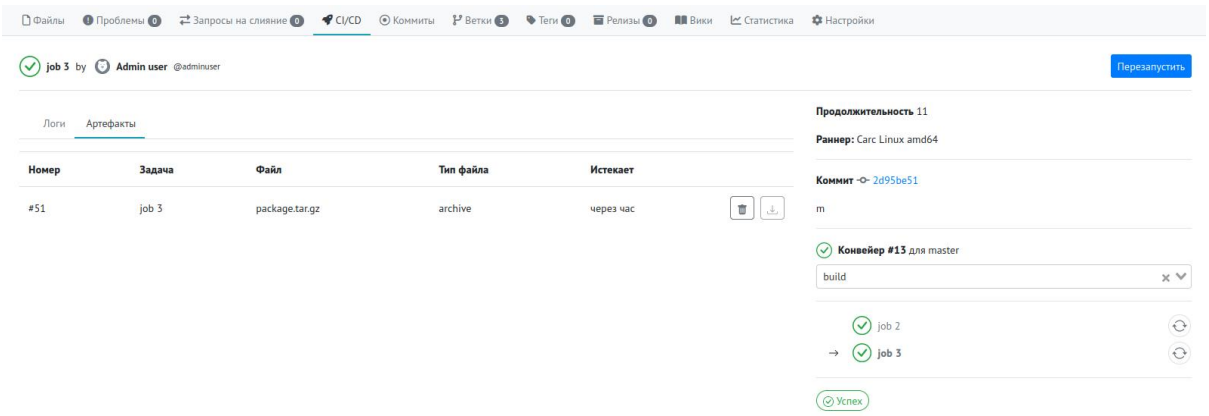


На вкладке с ошибками собирается информация об ошибках задач. Для получения подробных логов необходимо нажать на заголовок задачи, вас переведет на новую страницу. Страница задачи конвейера расположена по адресу виду project-name/ci-cd/job/*/, где * - номер выполняемой задачи.

На вкладке с логами отображены все логи по выполнению задачи в конвейере. Есть кнопки управления логами: просмотр raw файла логов и удаление логов выбранной задачи. Используйте правую панель для навигации между задачами конвейера.



На вкладке с артифактами расположены результаты выполнения задачи. Артифакты создаются в соответствии с настройками в вашем gitflic-ci.yaml файле. Артифакты можно загрузить, либо удалить. У всех артифактов имеется срок жизни и они будут удалены по истечении этого срока.



Конвейер

Конвейер (pipeline) - это верхнеуровневый компонент процесса непрерывной интеграции и доставки.

Конвейер состоит из следующих элементов:

- Задачи (jobs), которые определяют что кокнретно что-то нужно сделать;
- Этапы (stages), которые определяют когда конкретно нужно выполнить задачу. Например, этап build обычно следует за этапом test

Задачи этапов выполняются агентами. Различные задачи этапа выполняются параллельно. Каждая задача этапа выполняется отдельно от остальных задач этапа. количество, выполняемых параллельно задач, ограничено доступным количеством агентов.

Конфигурация

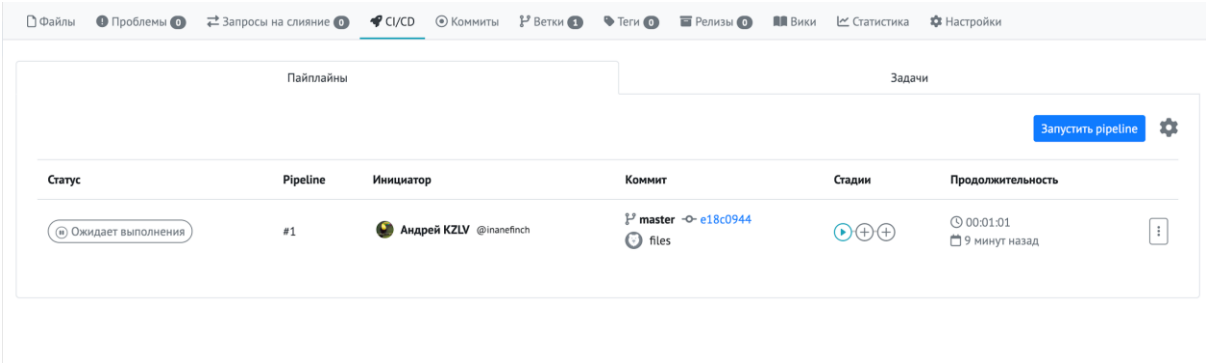
Конвейер и его компоненты (задачи и этапы), описываются в специальном конфигурационном файле gitflic-ci.yaml для каждого проекта отдельно.

- Задача - это базовый конфигурационный компонент
- Этапы описываются с помощью ключевого слова stages

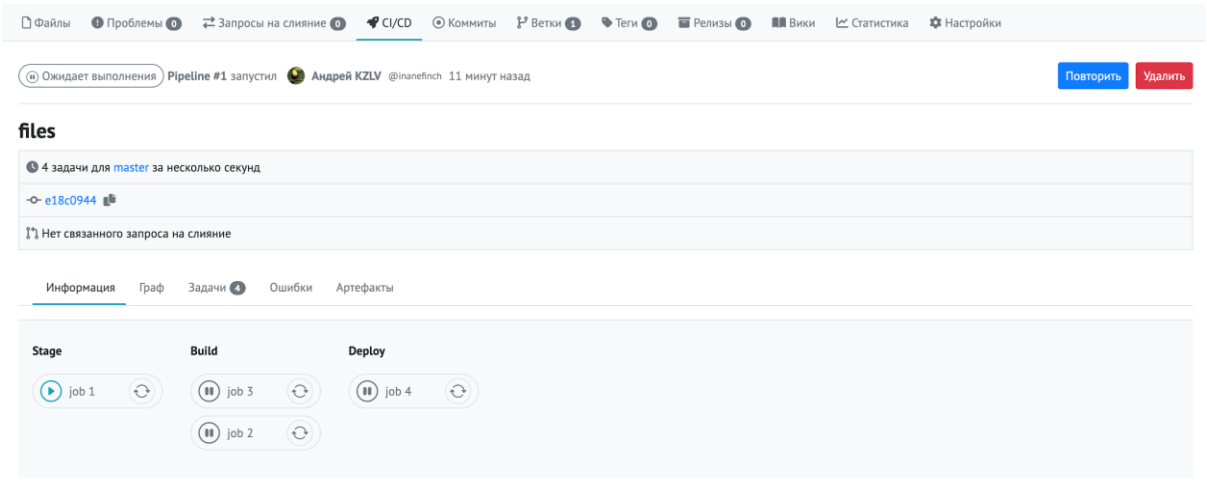
Список опций конфигурации можно посмотреть на странице справочника Ci/CD конфигурации.

Запуск конвейеров

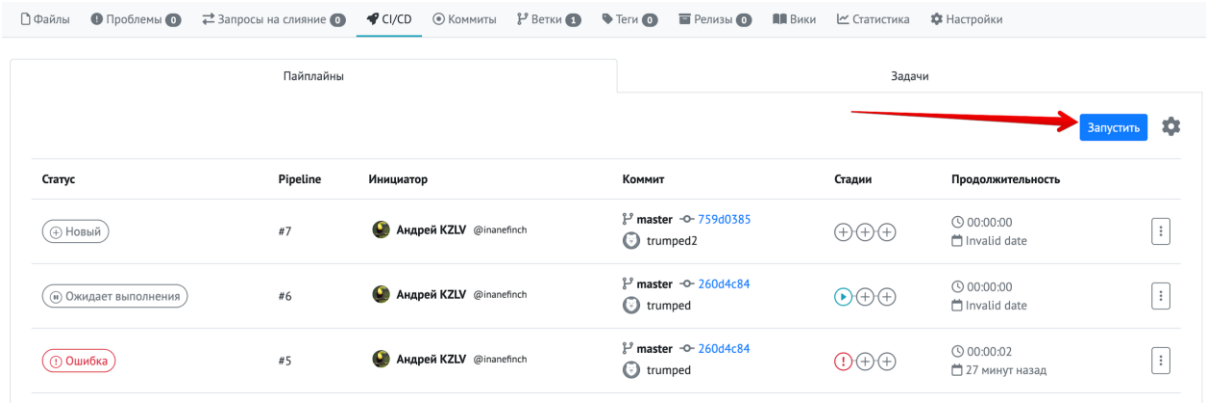
Вы можете найти текущие и предыдущие запуски конвейера на странице вашего проекта CI/CD > Конвейеры.



Нажмите на заголовок конвейера, чтобы открыть страницу сведений о контейнере. Страница имеет адрес вида project-name/ci-cd/pipeline/*/, где * - номер созданного конвейера. На странице сведений отображена информация о выполнении заданий, кнопки упраления конвейером (перезапуск и удаление), логи выполения задач и артефакты проекта. Для удаления конвейера нажмите кнопку Удалить в правой верхней части экрана.

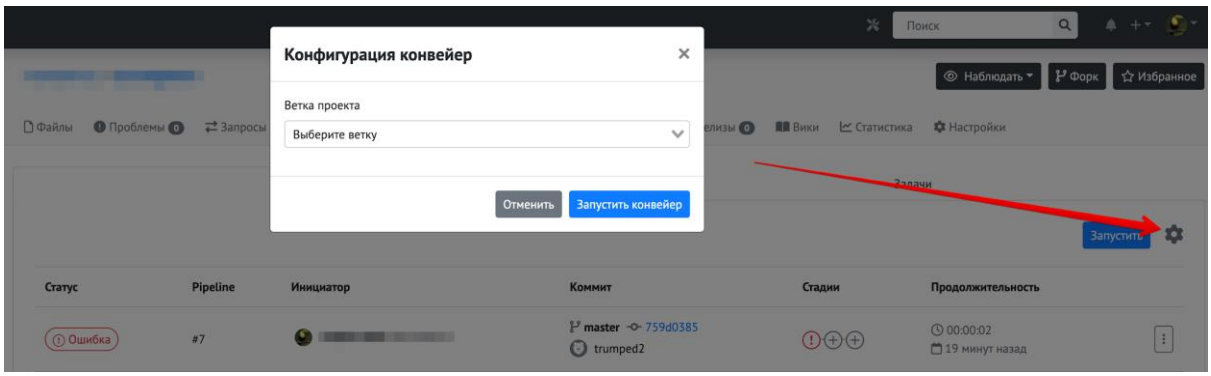


Для запуска конвейера нажмите кнопку “Запустить” на странице просмотра всех конвейеров. Также запусить можно со страницы просмотра отдельно выбранного конвейера по кнопке “Повторить”. После старта в списке конвейеров будет создана запись о новом конвейере и теперь конвейер выполняет задания в соответствии с настройками.

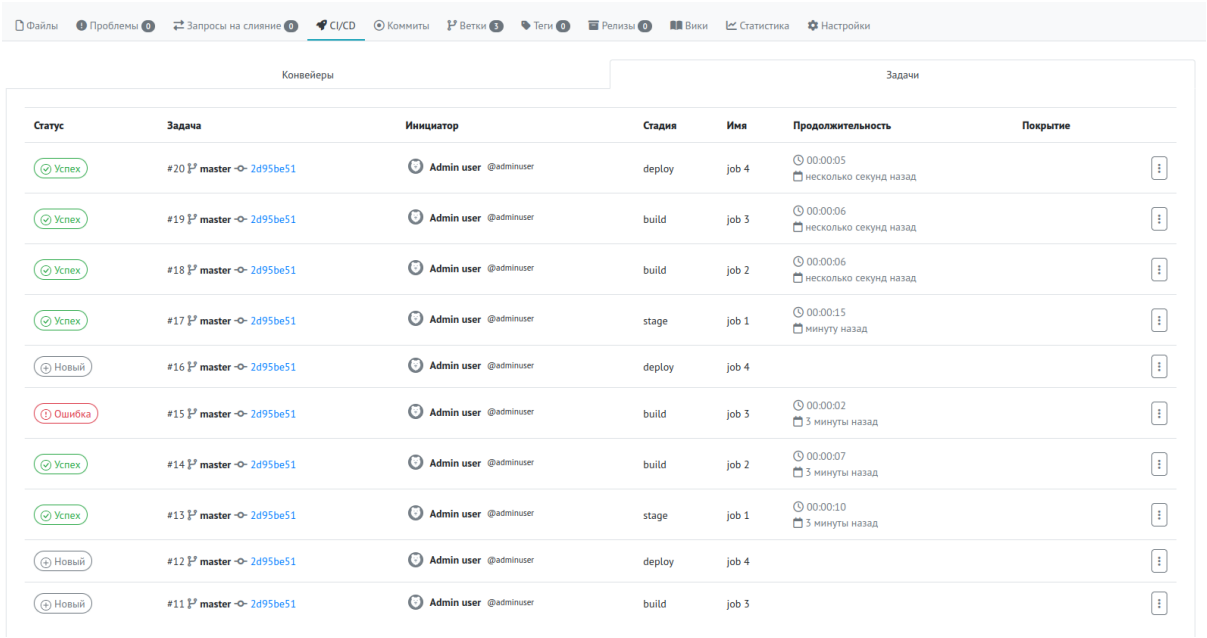


Для запуска конвейера на другой ветке проекта необходимо выполнить следующие действия:

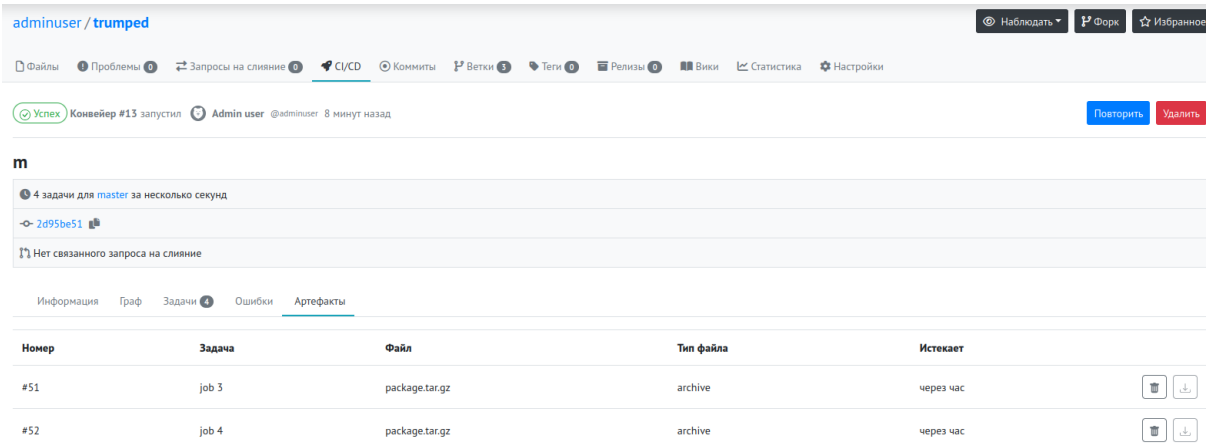
- Нажать на кнопку с шестеренкой;
- В открывшемся окне нажать на селектор веток;
- Указать нужную ветку проекта;
- Нажать “Запустить конвейер”.



На вкладке Задачи отображена информация о статусах и времени выполения всех задач со всех конвейеров. Для каждой задачи указан ее текущий статус, название стадии и задачи, время выполнения задачи, если она была запущена. В меню с многоточием можно перезапустить отдельно взятую задачу.



На вкладке Артефакты расположены все артефакты, которые собираются в рамках конвейера. Для перехода к этому экрану, перейдите к просмотру конвейера и откройте вкладку Артефакты. Чтобы инициировать загрузку артефакта проекта, нажмите кнопку загрузки напротив необходимого файла. По истечении указанного времени файлы будет удалена. Для ручного удаления артефакта нажмите на кнопку удаления.



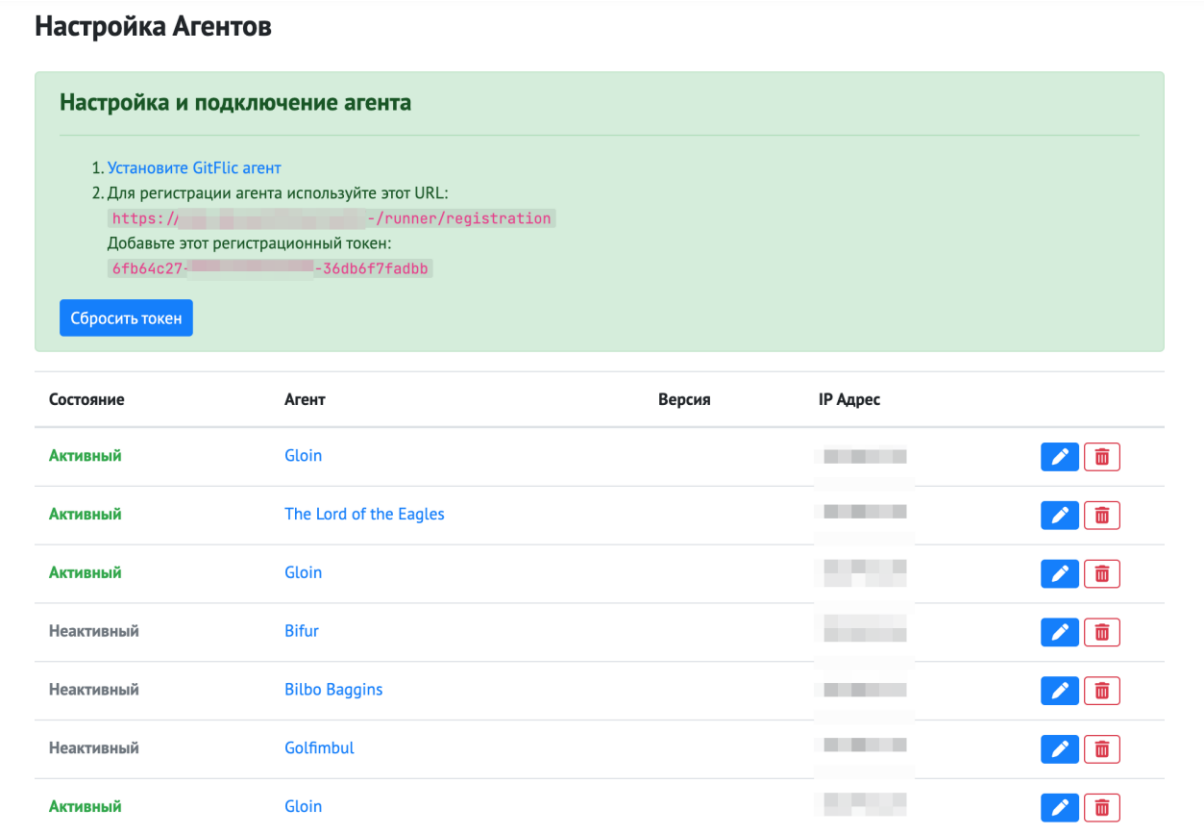
Агенты

Настройка агентов

Для настройки агентов CI/CD обратитесь к статье. Или к разделу Запуск агента (раннера)

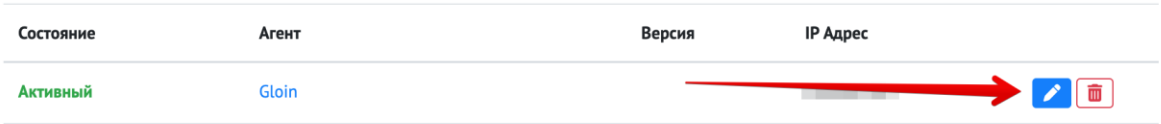
Наблюдение за активными агентами

В случае успешной регистрации агента он синхронизируется с приложением и передает информацию о себе в GitFlic. Информация об агенте собирается на панели администратора на странице “Агенты”. На этой странице отображены все зарегистрированные агенты.



Управление агентами

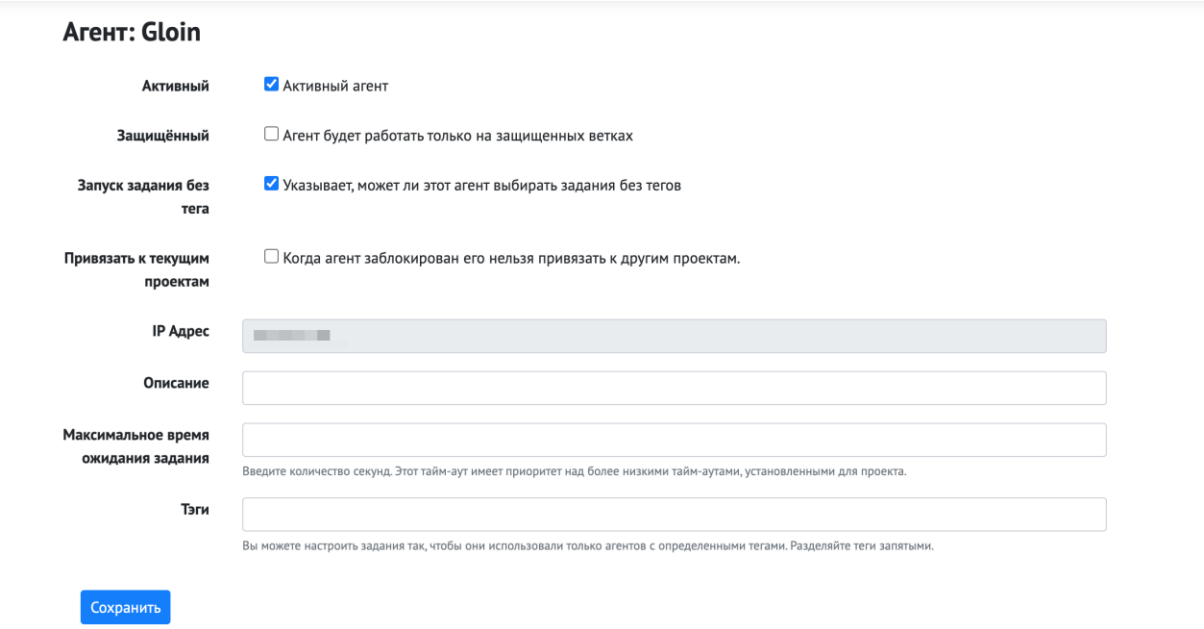
Для настройки агента нажмите на кнопку редактирования агента для выбора необходимых параметров работы агента. Для удаления агента нажмите на кнопку с иконкой корзины.



На странице редактирования агента можно управлять такими базовыми параметрами, как:

- Включение и выключение агента;
- Настройка работы агента на защищенных ветках (подробнее);
- Запуск агента только после тега или без него;
- Работа агента только на уже существующих проектах;
- Описание агента;
- Максимальное время ожидания задания в агенте. Указывается в секундах;

- Теги, по которым будет запускаться данный агент. Теги разделяются запятыми.



Кэш

Кэш предназначен для того, чтобы при выполнении задач агент мог переиспользовать различные файлы, которые обычно необходимо скачать со сторонних ресурсов.

Например, для сборки Java приложения с помощью maven или gradle скачиваются из репозитория артефактов необходимые библиотеки (mavencentral). Такие библиотеки можно сохранить на машине, где запущен агент и переиспользовать их в тех задачах, в которых они необходимы.

Как работает кэширование

Основой кеширования для текущей версии агента, который использует Docker для выполнения задач конвейера, является общий для проекта том (Docker Volume). Для каждого экземпляра агента (виртуальной машины, на которой запущен агент) создается свой том для конкретного проекта. Данный том монтируется к каждому контейнеру, который запускается для выполнения конкретной задачи проекта. В каждом томе может храниться множество архивов, которые получаются из конкретной конфигурации кэша задачи.

Архивы с кэшем различаются названием, которое является хешем от строки, состоящей из ключей кэша. Последовательность ключей в конфигурации не важна, потому что перед тем, как получить хеш строки с ключами, список ключей сортируется.

Если при конфигурировании кэша не указаны ключи для имени, то используется стандартное значение default.

Пример конфигурации кэша:

- без ключей кеширования. В данном случае для ключа кэширования применится значение default.

```
cache:
  paths:
    - .m2/repository/
```

- с ключами кеширования maven и build

```
cache:
  key:
    - maven
    - build
  paths:
    - .m2/repository/
```

Процесс работы задачи с кэшем

При запуске каждой задачи агентом, агент проверяет наличие доступного для работы кеша нужного для проекта. Если необходимый архив с кэшем найден, тогда этот архив распаковывается в рабочую директорию задачи. После завершения выполнения всех скриптов задачи агент ищет подходящие для кеширования файлы и папки, после этого эти файлы и папки упаковываются в архив, который отправляется на хранение в нужный для кэша том. Старый архив заменяется новым.

Справочник для .yaml файла

В этой документации перечислены ключевые слова для конфигурации вашего gitflic-ci.yaml файла

Глобальные ключевые слова

Ключевые слова	Описание
image	Docker Image
stages	Имена и порядок выполнения этапов конвеера

cache	Список файлов и каталогов для кэширования между задачами

Ключевые слова для задачи

Ключевые слова	Описание
stage	Определение стейджа для задачи
before_script	Список шелл-скриптов которые будут выполнены агентом перед задачей
scripts	Список шелл-скриптов которые будут выполнены раннером
after_script	Список шелл-скриптов которые будут выполнены агентом после задачи
artifacts	Список файлов и каталогов для прикрепления к задачи в случае успеха.
directory	
needs	Данное поле может содержать массив названий задач, которые необходимы для выполнения текущей задачи.
except	Название веток на которых задача не будет создана.
only	Название веток на которых будет создан.
tags	Теги задачи для раннера.
allow_failure	Используйте allow_failure, чтобы определить, должен ли конвейер продолжать работу в случае сбоя задачи.

Глобальные ключевые слова

image

Используйте image, чтобы указать образ Docker, в котором выполняется конвеер.

Пример:

```
image: maven:3.8.5-openjdk-11-slim
```

stages

Используйте stages, чтобы определить список этапов выполнения конвеера.

Если этапы не определены в gitflic-ci.yaml файле, то по умолчанию используются:

- .pre
- build
- test
- deploy
- .post

Порядок элементов этапов определяет порядок выполнения задач:

- задачи на одном этапе выполняются параллельно.
- задачи на следующем этапе запускаются после успешного завершения заданий на предыдущем этапе. Если конвеер содержит только задачи на этапах .pre или .post, он не запускается. На другом этапе должна быть хотя бы одна другая работа. .pre и .post этапы можно использовать в требуемой конфигурации конвейера для определения задач соответствия, которые должны выполняться до или после задач конвеера проекта.

Пример:

```
stages:
  - build
  - test
  - deploy
```

- Все этапы выполняются друг за другом. Если какой-либо этап будет провален, то не начнется следующий и конвеер завершится с ошибкой.
- Если все эти этапы будут успешно пройдены, то конвеер будет считаться успешно выполненным.

---

cache

Используйте cache, чтобы указать список файлов и каталогов для кэширования между задачами. Вы можете использовать только те пути, которые есть в локальной рабочей копии.

Кэширование распределяется между конвейерами и заданиями. Кэши восстанавливаются раньше артефактов.

cache:paths

Используйте cache:paths, чтобы выбрать файлы или каталоги для кэширования.

Пример:

```
cache:
  paths:
    - .m2/repository/
    - core/target/
    - desktop/target/
```

cache:key

Используйте cache:key, чтобы присвоить кэшу уникальный идентификационный ключ. Все задачи, использующие один и тот же ключ кэша, используют один и тот же кэш.

Если не задано, по умолчанию используется ключ default. Все задания с cache ключевым словом, но без cache:key совместного использования default кеша.

Должен использоваться с cache: path, иначе ничего не кэшируется.

cache:when

Используется cache:when для определения времени сохранения кэша в зависимости от состояния задачи.

Должен использоваться с cache: paths, иначе ничего не кэшируется.

Возможные значения:

- on_success(по умолчанию): Сохранять кеш только после успешного выполнения задачи.
- on_failure: Сохранять кеш только в случае сбоя задачи.
- always: всегда сохранять кеш.

Пример:

```
cache:
  paths:
    - .m2/repository/
    - core/target/
    - desktop/target/
```

Ключевые слова для задачи

stage

Используйте stage, для определения этапа для задачи.

Пример

```
stages:
  - build
  - deploy

job 0:
  stage: build
```

---

scripts

Используйте scripts, чтобы указать команды для выполнения раннером.

```
job1:
  script: apt-get update

job2:
  script:
    - apt-get -y install maven
    - apt-get -y install git
```

---

before_script

Используйте before_script для определения массива команд, которые должны выполняться перед командами сценария каждого задачи, после восстановления артефактов.

Тип ключевого слова: Ключевое слово для задачи. Вы можете использовать его только как часть задачи или в разделе по умолчанию.

Возможные значения: Массив строк

Пример

```
job1:
  script: apt-get update
  before_script:
    - apt-get -y install maven
    - apt-get -y install git
```

Дополнительные сведения:

- Сценарии, указанные в before_script, объединяются с любыми сценариями, указанными в script. Объединённые сценарии выполняются вместе в одной оболочке.

---

after_script

Используйте after_script для определения массива команд, которые должны выполняться после команд сценария каждой задачи, включая ошибочные задачи.

Тип ключевого слова: Ключевое слово для задачи. Вы можете использовать его только как часть задачи или по умолчанию во всем конвеерее.

Возможные значения: Массив строк:

Пример

```
job1:
  script: apt-get update
  after_script:
    - apt-get update
```

```
  - apt-get -y install maven
  - apt-get -y install git
```

Дополнительные сведения:

Если время ожидания задачи истекло или он был отменён, команды after_script не выполняются.

По умолчанию используются внешние before_script и after_script shell-скрипты, однако задача может иметь и свои собственные.

Если они не определены внутри задачи, то используются внешние.

Пример

```
image: test

stages:
 - test

job 0:
  stage: test
  scripts:
    - mvn test
  rules:
    - if: "predicate1 && predicate2"
      when: newer
```

---

artifacts

Используйте артефакты, чтобы указать, какие файлы сохранять в качестве артефактов задачи. Артефакты задачи - это список файлов и каталогов, которые присоединяются к задаче при его успешном выполнении, сбое или всегда.

По умолчанию задачи на более поздних этапах автоматически загружают все артефакты, созданные ими на более ранних этапах. Вы можете управлять поведением загрузки артефактов в задачах с зависимостями.

При использовании ключевого слова needs задачи могут загружать артефакты только из задач, определённых в конфигурации needs.

Артефакты задач по умолчанию собираются только для успешных задач, а артефакты восстанавливаются после кэширования.

artifacts:paths

Пути относятся к каталогу проекта и не могут напрямую ссылаться за его пределы.

Тип ключевого слова: Ключевое слово для задачи. Вы можете использовать его только как часть задачи или в разделе по умолчанию.

Возможные значения:

- Массив путей к файлам относительно каталога проекта.

Пример

```
artifacts:
  paths:
    - bin/usr/
    - bin/path
    - frontend/saw
```

artifacts:exclude

Используйте artifacts:exclude, чтобы предотвратить добавление файлов в архив артефактов.

Тип ключевого слова: Ключевое слово для задачи. Вы можете использовать его только как часть задачи или в разделе по умолчанию.

Возможные значения:

- Массив путей к файлам относительно каталога проекта.

Пример

```
artifacts:
  exclude:
    - binaries/**/*.o
```

Данный пример сохраняет все файлы в binaries/, исключая *.o файлы, находящиеся в дочерних директориях binaries/.

---

needs

Данное поле может содержать массив названий задач, которые необходимы для выполнения текущей задачи.

Пример

```
job-0:
  stage: test
  script: echo "Running job 0"

job-1:
  stage: test
  needs: job-0
  script: echo "Running job 1..."
```

---

tags

Используйте tags для настройки раннера.

Вы можете указать теги для определенного раннера в настройках раннера

Возможные значения

- Массив названий тегов

```
job:
  tags:
    - java
    - postgres
```

---

allow_failure

Используйте allow_failure, чтобы определить, должен ли конвейер продолжать работу в случае сбоя задания. Значение по умолчанию каждой задачи - false

- Чтобы конвейер продолжал выполнять последующие задачи, используйте allow_failure: true.
- Чтобы запретить конвейеру выполнение последующих задачи, используйте allow_failure: false.

Возможные значения:

- true или false

---

except

Используйте except чтобы указать ветки на которых задача не будет создана.

Возможные значения

- Массив названий веток

```
job:
  except:
    - master
    - deploy
```

---

only

Используйте only чтобы ветки на которых будет работать задача.

Возможные значения

- Массив названий веток

```
job:
  only:
    - master
```

Общая информация

Глоссарий

- Репозиторий каталог файловой системы, в котором находятся: файлы конфигурации, файлы журналов операций, сами файлы проекта.
- Локальный репозиторий — репозиторий, расположенный на локальном компьютере разработчика в каталоге. Именно в нём происходит разработка и фиксация изменений, которые отправляются на удалённый репозиторий.
- Удалённый репозиторий — репозиторий, находящийся на удалённом сервере. Это общий репозиторий, в который приходят все изменения и из которого забираются все обновления.
- Форк (Fork) — копия репозитория. Копия вашего публичного репозитория может быть сделана любым пользователем, после чего он может прислать изменения в ваш репозиторий через пулреквест.
- Клонирование — скачивание репозитория с удалённого сервера на локальный компьютер в определённый каталог для дальнейшей работы с этим каталогом как с репозиторием.
- Ветка — это параллельная версия репозитория. Она включена в этот репозиторий, но не влияет на главную версию, тем самым позволяя свободно работать в параллельной. Когда вы внесли нужные изменения, то вы можете объединить их с главной веткой.
- Коммит (Commit) — фиксация изменений или запись изменений в репозиторий. Коммит происходит на локальной машине.
- Пулл (pull) — получение последних изменений с удалённого сервера репозитория.
- Пуш (Push) — отправка всех неотправленных коммитов на удалённый сервер репозитория.
- Фикс - исправление проблемы, ошибки.
- Фича - новый функционал.
- Лейбл - ярлык, метка.
- Релиз - промежуточная стадия разработки программного обеспечения. На данном этапе издатель признаёт ПО стабильным и либо вносит в него лишь необходимые исправления.
- Селектор - меню выбора, часто представлено в выпадающем списке.
- Вебхук — это «пользовательский обратный вызов по HTTP». Обычно вебхуки запускаются каким-либо событием, например, отправкой кода в репозиторий или комментарием (в запросе на слияние или проблеме). Когда происходит это событие, сайт отправляет HTTP-запрос на URL-адрес, указанный для вебхука.

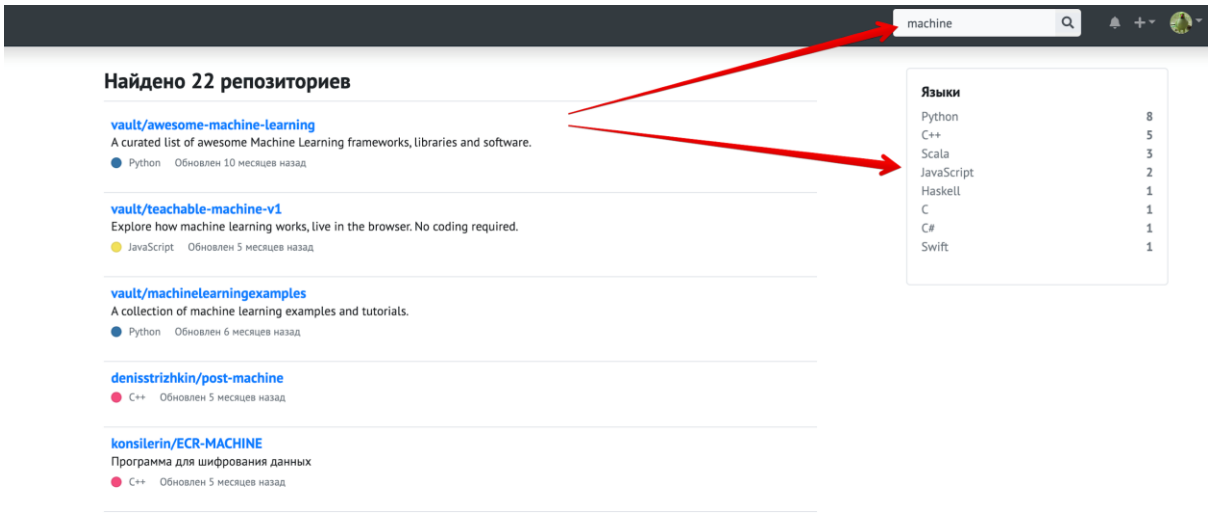
Пользователи могут настроить их так, чтобы события в проекте вызывали действия на другом сервисе.

Поиск

Поиск открывает вам доступ ко всем публичным проектам, расположенным в сервисе GitFlic.

Чтобы произвести поиск по проектам, в верхнем правом углу в поле введите ключевое слово и нажмите на кнопку поиска. Вас переведет на экран с результатами поиска. Воспользуйтесь фильтром по языку программирования, чтобы сузить выдачу результатов.

Поиск производится только по публичным проектам. Поисковый запрос сравнивается с названием и с описанием проекта. Чтобы облегчить поиск вашего проекта, укажите в описании больше ключевых слов.

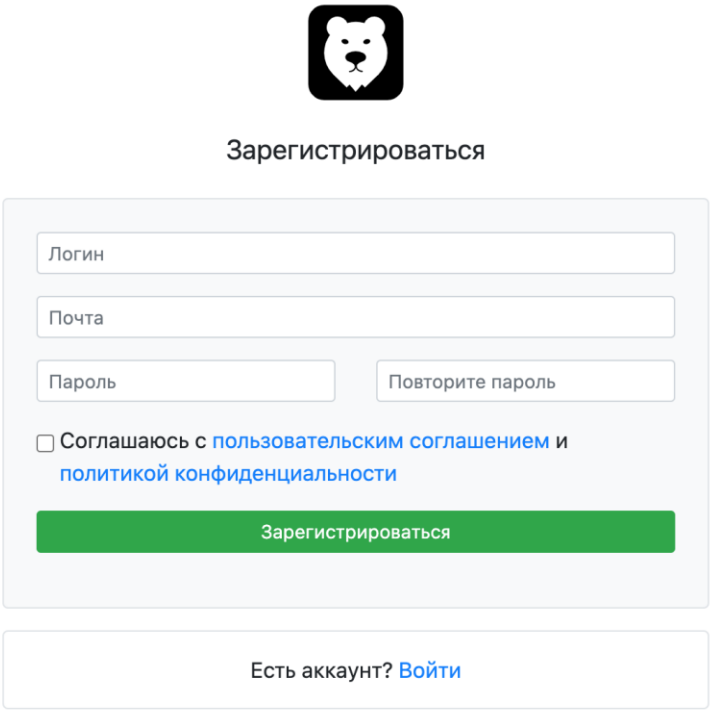


Если вы не хотите, чтобы ваши проекты отображались в поиске, укажите вашему проекту тип “Приватный.”

Регистрация

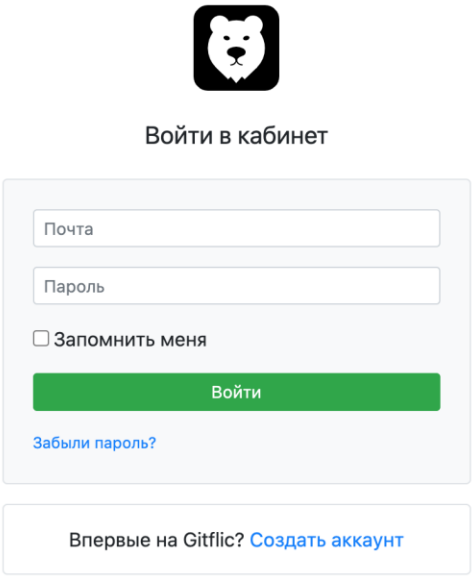
Создание нового профиля

Для регистрации нового профиля GitFlic на странице регистрации укажите логин, почту для профиля и пароль. Ознакомьтесь с Пользовательским соглашением и Политикой конфиденциальности, поставьте галочку, подтверждающую ваше согласие с документами, и нажмите “Зарегистрироваться”. Вам на почту будет отправлено письмо с подтверждением, перейдите по вложенной ссылке для активации вашего профиля.

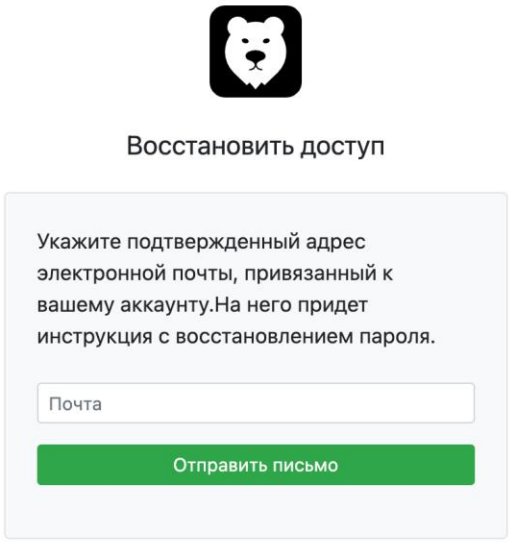


Вход в профиль

Вход в профиль осуществляется при помощи почты и пароля. Также можете использовать опцию продления сессии авторизации, для этого поставьте галочку напротив “Запомнить меня”



Если вы забыли пароль, который указывали при регистрации, то воспользуйтесь функцией восстановления пароля. Нажмите кнопку “Забыли пароль?”, вас переведет к окну ввода почты профиля. Укажите ваш емейл и нажмите “Отправить письмо”, вам придет письмо, следуйте инструкциям в письме, для создания нового пароля.



Начало работы с Git

Итак, вы решили начать использовать git ,но не знаете с чего начать? Данную документацию можно использовать в качестве настолько пособия и обращаться по возникающим вопросам. Для начала вам следует создать свой проект. Когда вы создали свой новый репозиторий, необходимо подготовить локальное рабочее пространство. Используйте консоль на вашем ПК для работы.

Для большего удобства, пользователям Windows, рекомендуем установить отдельную консоль для работы с git. В поисковике запросом *“Git для Windows”* выбрать предпочитаемый и установить его.

Когда откроете консоль, выполните следующие команды для создания локальной директории для вашего будущего репозитория:

Создание локальной директории

```
cd ~/
mkdir repos
cd ~/repos
```

После выполнения предыдущих команд следует указать свои данные, выполнив следующие команды:

Глобальные настройки Git

```
git config --global user.name "gitflic-user"
git config --global user.email "mail@gitflic.ru"
```

Далее необходимо выбрать один из вариантов, который подходит вашей ситуации и выполнить команды по порядку

Создание нового репозитория

```
git clone http://gitflic.ru/project/user/proekt.git //Здесь ссылка на ваш проект
cd proekt
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Использовать существующую директорию

```
cd existing_folder
git init
git remote add origin http://gitflic.ru/project/user/proekt.git //Здесь ссылка на ваш проект
git add .
```

```
git commit -m "Initial commit"
git push -u origin master
```

Запушить существующий репозиторий

```
cd existing_folder
git remote rename origin old-origin
git remote add origin http://gitflic.ru/project/user/proekt.git //Здесь ссылка на ваш проект
git push -u origin --all
git push -u origin --tags
```

После выполения команд мы видим как локальная/удаленная дериктория наполнились файлами. Теперь, как мы разобрались с настройкой репозитория, самое время разобраться как пользоваться ветками в системе git

Для создания ветки и мгновенного переключения на новую ветку используется следующая команда, где *omega*, параметр *-b* указывает на ветку:

Как создать ветку в git

```
git checkout -b omega
```

Также можно выполнить то же самое действие в 2 команды, отдельно создать ветку, отдельно в нее переключиться:

```
git branch omega
git checkout omega
```

Распространенные опции для git branch

Вывод списка всех веток в репозитории.

```
git branch
или
git branch --list
```

Удаление ветки с названием *omega*. Это «безопасная» операция, так как git не позволит удалить ветку, в которой есть неслитые изменения. Данная команда удаляет только локальную ветку.

```
git branch -d omega
```

Принудительное удаление указанной ветки, даже несмотря на то, что в ней есть неслитые изменения. Эту команда следует использовать, если вы хотите полностью удалить все коммиты, которые относятся к определенному направлению разработки.

```
git branch -D <branch>
git push origin :refs/heads/<branch>
```

Вывод списка всех веток удаленного проекта.

```
git branch -a
```

Для визуализации файлов под управлением git существует расширение TortoiseGit. Оно имеет логичный интерфейс и отображает иконки к файлам, находящимся под управлением gitдля отображения их статуса в git.

Генерация публичного SSH ключа

Для работы с git многие серверы используют аутентификацию по ssh-ключу. Далее расскажем как создать свой ssh-ключ для работы с git. Процесс создания ssh-ключа аналогичен на всех ОС. Первым делом убедимся, что у вас отсутствует ssh-ключ на локальном компьютере, для этого выполните следующие команды:

```
cd ~/.ssh
ls
```

Ищите файл с именем *id_dsa* или *id_rsa* и одноименный файл с расширением *.pub*. Файл с расширением *.pub* — это ваш публичный ключ, а второй файл — ваш приватный ключ. Если указанные файлы у вас отсутствуют (или отсутствует директория .ssh), вы можете создать их используя команду:

```
cd ~/
ssh-keygen -o
```

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/gitflic_user/.ssh/id_rsa):
Created directory '/home/gitflic_user/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/gitflic_user/.ssh/id_rsa.
Your public key has been saved in /home/gitflic_user/.ssh/id_rsa.pub.
The key fingerprint is:
d0:82:24:8e:d7:f1:bb:xx:yy:zz:96:93:49:da:9b:e3 gitflic_user@gitflic.ru
```

После создания вашего публичного ssh-ключа его необходимо прописать в настройках к git. Первым делом необходимо получить ssh-ключ из файла. Вы можете открыть ваш публичный ssh-ключ в текстовом редакторе и полностью скопировать содержимое, либо выполнить следующую команду в консоли:

```
cat ~/.ssh/id_rsa.pub
```

Когда скопируете ssh-ключ, пропишите его в настройках, укажите название для ключа, например *my-ssh*, оставьте поле со сроком годности пустым, тогда ваш ssh-ключ будет всегда активен. Нажмите сохранить и убедитесь что сохранение прошло успешно.

Чтобы убедиться что ключ сохранен верно, вы можете при помощи следующей команды получить хэш ключа и сравнить его с тем, что отображается в настройках профиля.

```
ssh-keygen -E sha256 -l -f id_rsa.pub
```

Возможные проблемы с работой по ssh на Windows.

Начиная с некоторых версий Windows 10, может возникать ошибка *no mutual signature algoryth* при использвании *id_rsa.pub*. Чтобы решить эту проблему, необходимо сгенерировать новый ssh-ключ по алгритму *ED25519*. Для создания нового публичного ключа по алгоритму ED25519, выполните следующую команду.

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

Далее полученный ключ, аналогично процедуре выше, сохраните в настройках вашего профиля GitFlic.

Распространенные команды при работе с git

git clone

Как правило при помощи команды *git clone* создается копия репозитория от указанного. Это делается в новой директории или в другом месте. Исходный репозиторий может находиться в локальной файловой системе или на удаленном устройстве, к которому можно получить доступ с помощью поддерживаемых протоколов.

Пример клонирования проекта в дерикторию *folder* по ssh-ключу:

```
git clone git@f.dev.gitflic.ru:user/how-to.git ./folder/
```

Аргумент *-branch* позволяет выбрать ветку для клонирования. В противном случае клонируется ветка, на которую указывает HEAD в удаленном репозитории (обычно главная ветка). Кроме того, для этих целей в команде можно задать тег вместо ветки:

```
git clone -branch new_feature git://remoterepository.git
```

git clone –bare Как и *git init –bare*, аргумент *-bare* при назначении команде *git clone* приводит к созданию копии удаленного репозитория без рабочего каталога. Это означает, что репозиторий будет содержать историю проекта, к которой можно выполнять запросы *push* и *pull*, но которую нельзя редактировать напрямую. Кроме того, в репозитории, клонированном с опцией *-bare*, не будут настроены удаленные ветки. Как и *git init –bare*, эта команда создает удаленный репозиторий, который разработчики не смогут редактировать напрямую.

git clone –mirror Вместе с аргументом *–mirror* команде неявно назначается и аргумент *–bare*. Поэтому можно сказать, что опция *–mirror* наследует поведение *–bare*, создавая чистый репозиторий без изменяемых рабочих файлов. Кроме того, *–mirror* клонирует ссылки удаленного репозитория и сохраняет конфигурацию отслеживания удаленных веток. Затем вы можете выполнить команду *git remote update* на созданном зеркале, в

результате чего будут перезаписаны все ссылки из исходного репозитория. Так вы получите идентичные функциональные возможности для работы.

git add

Команда *git add* добавляет изменения рабочей директории в промежуточную область. Он сообщает git, что вы хотите включить обновления определенного файла в следующий коммит. Однако *git add* на самом деле не влияет на репозиторий каким—либо существенным образом - изменения фактически не записываются до тех пор, пока вы не выполните git commit.

Добавление всех изменений в следующий коммит выполняется следующей командой:

```
git add .
```

Чтобы добавить в следующий коммит только один измененный файл, то это делается командой, где - полное имя файла:

```
git add <file-name>
```

В сочетании с этими командами вам также понадобится *git status* для просмотра состояния рабочего каталога и промежуточной области:

```
git status
```

git commit

Коммит проиндексированного состояния кода производится по следующей команде:

```
git commit
```

Стоит отметить, что эта команда откроет текстовый редактор, введите комментарий к коммиту. После ввода сохраните файл и закройте текстовый редактор, чтобы выполнить коммит. Выполнение коммита со всеми изменениями в рабочей директории. Эта команда включает только изменения отслеживаемых файлов (которые были добавлены командой *git add*):

```
git commit -a
```

Данная команда создаст коммит с указанным комментарием. По умолчанию команда *git commit* открывает локальный текстовый редактор для ввода комментария к коммиту. При передаче параметра *-m* используется добавленный комментарий, минуя текстовый редактор:

```
git commit -m "commit message"
```

Также есть параметр, который позволяет команде *commit* изменять последний коммит. Вместо создания нового, все изменения добавляются в последний. Кроме того, после выполнения команды откроется текстовый редактор и предложит изменить ранее указанный комментарий к комиту:

```
git commit --amend
```

git pull

Команда git pull запускает команду git fetch для загрузки содержимого из указанного удаленного репозитория. Затем выполняется команда git merge, осуществляющая слияние ссылок и указателей удаленного содержимого в новый локальный коммит:

```
git pull <remote>
```

Выше указанная команда берет указанную удаленную копию текущей ветви и объединяет ее с локальной копией. Это то же самое, что и git fetch ＜remote＞, за которым следует git merge origin/＜current-branch＞.

Существует команда, подобная команде по умолчанию, только она не создает новый коммит со слитым содержимым:

```
git pull --no-commit <remote>
```

git push

Публикация указанной ветки в удаленном репозитории вместе со всеми необходимыми коммитами и внутренними объектами. Эта команда создает локальную ветку в репозитории назначения. Чтобы предотвратить перезапись коммитов, git не позволит опубликовать данные, если в репозитории назначения нельзя выполнить ускоренное слияние:

```
git push <remote> <branch>
```

Если для параметра указать значение *origin* и оставить пустым параметр *branch*, то изменения будут отправленны в ветку, которая выбрана в данный момент.

Параметр -u аналогично –set-upstream указывает удаленную ветку “по умолчанию”, все последующие команды *git pull/push* будут автоматически общаться между текущей локальной и выбранной удаленной ветками. Данная команда указывается единожды, до тех пор, пока не понадобится указать другую удаленную ветку “по умолчанию”.

```
git push -u origin master
```

Существует процедура для очистки локальной и удаленной веток, которую целесообразно выполнять для поддержания порядка и не допущения накопления изменений, которые не будут загружены в основную ветку проекта:

```
git branch -D alpha
git push origin :alpha
```

Первая команда очистит локальную ветку *alpha*. Если в команде *git push* перед именем ветки поставить двоеточие, будет стерта удаленная ветка.

git merge

Для слияние веток используется команда *git merge*. Обычно, в веб интерфейсе есть механизм для создания и управления мерж-реквестами, однако, это можно сделать и через консоль.

git revert

Эта команда отменяет внесенные в коммит изменения и добавляет новый коммит с обращенным содержимым. В результате история в git не теряется, что важно для обеспечения целостной истории версий и надежной совместной работы. К примеру вы хотите вернуть ваш рабочий процесс к коммиту *1f08a70*, команда возврата будет выглядеть следующим образом:

```
git revert 1f08a70
```

git reset

Универсальная команда для отмены изменений. Другими словами, если вопрос какой командой создается новый коммит, применяющий изменения, обратные для указанного аргументом коммита? Следует внимательно ознакомиться с этой командой. У команды *git reset* имеется несколько опций, разберем следующие:

Параметр *–soft* сбрасывает указатель *HEAD* до указанного коммита:

```
git reset --soft 1f08a70
```

Параметр *–mixed* сброс указателя *HEAD* до выбранного коммита в истории и отмена изменений в индексе:

```
git reset --mixed 1f08a70
```

Параметр *–hard* сбрасывает указатель *HEAD* до выбранного коммита в истории, отмяет изменения в индексе и отменяет изменения в рабочем каталоге. Не используйте данный параметр, если не уверены в своих действиях:

```
git reset --hard 1f08a70
```

git log

Эта команда позволяет просмотреть и фильтровать историю проекта, а даже искать конкретные изменения. Для сравнения с *git status* можно просматривать рабочую директорию и раздел проиндексированных в ней файлов, в то время как *git log* показывает только историю коммитов проекта. Этот же журнал коммитов можно найти на странице проекта в gitflic.ru. Каждая запись будет состоять из 4-х частей: хеш коммита, автор коммита, дата и комментарий к коммиту. Пример ответа:

```
commit 9936edb7ba9af7bf2443xxx11129e66834b9c2fa (HEAD -> master, origin/master)
Author: gitflic_user <user@gitflic.ru>
Date:   Thu Jun 3 11:01:00 2020 +0300
```

git init

Данная команда часто выполняется в автоматическом режиме, при клонировании проекта, однако, ей необходимо воспользоваться, когда вы заливаете ваш локальный репозиторий в сервис хранения кода. В момент выполнения команды создается папка *.git* со служебными файлами и назначается ветка master для вашего проекта.

```
git init
```

Пример ответа:

```
Initialized empty Git repository in /Users/user/path/project/.git/
```

Права доступа ролей

В GitFlic вы можете управлять ролями других пользователей, который добавляете в свой проект или организацию. Роли идут в порядке увеличения количества прав и позволяют эффективнее работать в группе. Вы можете настроить доступ к каждому репозиторию в вашей организации, назначив роли, предоставив людям доступ к функциям и задачам, которые им нужны.

Доступные роли.

От наименьшего доступа к максимальному доступу роли для репозитория и для организации следующие:

- Гость - пользователь с правами только на просмотр. Не имеет доступа изменять и сохранять данные.
- Докладчик - пользователь с правами на создание проблем в проекте. Роль подойдет аудитору или тестировщику.
- Разработчик - пользователь, который имеет непосредственный доступ к разработке и базовым изменениям в проекте, работая с проектом.
- Администратор - пользователь, который имеет доступ к настройкам проекта или компании. Может создавать правила доступа и добавлять новых участников.
- Владелец - то же, что и Администратор, но может передать проект другому пользователю или организации.

Таблица ролей участников проекта.

Действие пользователя	Гость	Докладчик	Разработчик	Администратор	Владелец*

Клонирование репозитория, скачивание файлов	✔	✔	✔	✔	✔
Создавать проблемы (issue)		✔	✔	✔	✔
Закрывать проблемы, которые создали сами		✔	✔	✔	✔
Закрывать проблемы, которые создали другие участники				✔	✔
Назначить на них проблему	✔	✔	✔	✔	✔
Создание запроса на слияние			✔	✔	✔
Просмотр созданных запросов на слияние	✔	✔	✔	✔	✔
Оставлять комментарии на запросы на слияние		✔	✔	✔	✔
Управление правилами для запросов на слияние		✔	✔	✔	✔
Просмотр опубликованных релизов	✔	✔	✔	✔	✔
Создание новых релизов			✔	✔	✔
Редактирование релизов			✔	✔	✔
Управление правилами защиты веток				✔	✔
Управление правилами защиты тегов				✔	✔
Создание тегов, соответствующих правилу защиты тегов			✔	✔	✔
Создание вики проекта				✔	✔
Редактирование вики проекта (при наличии доступа к проекту)			✔	✔	✔

Создание книги проекта				✔	✔
Редактирование книги проекта (при наличии доступа к проекту)			✔	✔	✔
Доступ к настройкам репозитория и изменение основных данных				✔	✔
Добавление новых участников в проект				✔	✔
Добавление команд к проекту				✔	✔
Настройка веток по умолчанию				✔	✔
Управление вебхуками в проекте (в т.ч. создание, редактирование, удаление)				✔	✔
Управление приватностью проекта				✔	✔
Передача проекта другому пользователю или организации					✔

*Роль Владелец можно назначить участнику проекта.

Таблица ролей участников компании.

Действие пользователя	Гость	Докладчик	Разработчик	Администратор
Просмотр публичных проектов компании*	✔	✔	✔	✔
Просмотр участников компании	✔	✔	✔	✔
Просмотр команд в компании	✔	✔	✔	✔
Добавление новых участников в компанию				✔
Добавление новых команд в компанию				✔
Доступ к настройкам компании и изменение основных данных				✔

Доступ к странице платежей и смене тарифа				✔

*На публичные проекты компании распространяются права из компании. Для приватных проектов необходим отдельный доступ непосредственно из настроек проекта.

API

Введение

Введение и начало использования GitFlic API

В этом руководстве вы найдете информацию о принципах работы GitFlic API.

Для начала использования API необходимо получить access token

Все запросы для взаимодействия с API необходимо отправлять на:

1) для SaaS версии api.gitflic.ru

2) для self-hosted localhost:8080/rest-api

Получение accessToken

1. Авторизация

Для авторизации отправить get-запрос по адресу:

1) для SaaS https://oauth.gitflic.ru/oauth/authorize

2) для self-hosted localhost:8080/oauth/authorize

Запрос принимает 4 обязательных параметра: scope, clientId, redirectUrl, state

Параметр	Тип	Описание
scope	Enum	Список предоставляемых прав(scope'ов)

clientId	String	ID клиента, который можно получить в личном кабинете в настройках
redirectUrl	String	URL на который будет перенаправлен после запроса
state	String	Параметр, идентифицирующий конечного пользователя

Типы scope

Scope	Описание
USER_READ	Просмотр информации о пользователе
USER_WRITE	Редактирование профиля пользователя
PROJECT_READ	Просмотр информации о проектах пользователя
PROJECT_WRITE	Создание проектов от лица пользователя
PROJECT_EDIT	Редактирование существующих проектов пользователя

Пример запроса:

https://oauth.gitflic.ru/oauth/authorize?scope=USER_READ,USER_WRITE,PROJECT_READ,PROJECT_WRITE,PROJECT_EDIT&clientId=973d8a-c4-427c-83ee-f29ba163bb53&redirectUrl=https://example.org/&state=12345

2. Получение JSON на обратный вызов авторизации

На URL обратного вызова авторизации, указанный в личном кабинете, посылается JSON-объект следующей структуры:

Поле	Тип	Описание
code	String	Персональный код
state	String	Параметр, идентифицирующий конечного пользователя

3. Получение Access Token

Для получение access token отправить get-запрос по адресу https://oauth.gitflic.ru/api/token/access

Запрос принимает один обязательный параметр: code - это персональный код

В ответ придёт JSON-объект следующей структуры:

Поле	Тип	Описание
accessToken	String	Токен для доступа к API
refreshToken	String	Токен для получения нового accessToken
expires	String	Дата истечения срока accessToken

Пример запроса:

https://oauth.gitflic.ru/api/token/access?code=e462eb8c-f812-4458-b55c-d12cba0cb852

Пример получаемоего JSON-объекта:

```
{
   "accessToken": "868f3765-b22g-4a0b-b363-13faw8a3ca68",
   "refreshToken": "cb4917e7-5c18-4914-876d-9bb604b42e0c",
   "expires": "2023-01-12T15:45:01.961001"
}
```

Теперь мы можем использовать API, вставляя полученный токен в headers нашего запроса:

Authorization: token <Полученный токен>

Метод branch

Описание структуры JSON-объекта, описывающего ветку

Поле	Тип	Описание
name	String	Название ветки
fullName	String	Полное имя ветки
lastCommit	Object	Последний коммит
default	Boolean	Дефолтная ветка
merged	Boolean	Смердижная ветка

work	Boolean	Рабочая ветка

Метод для получения всех веток проекта

GET /project/{userAlias}/{projectAlias}/branch

Запрос возвращает массив всех веток, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/branch	Список веток

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для получания информации о ветке

GET /project/{userAlias}/{projectAlias}/branch/{branchName}

Запрос возвращает информацию о ветке

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/branch/{branchName}	Информация о ветке

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя

projectAlias	String	Псевдоним проекта
branchName	String	Название ветки

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для получения дефолтной ветки

GET /project/{userAlias}/{projectAlias}/branch/default

Запрос возвращает дефолтную ветку

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/branch/default	Информация о дефолтной ветке

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта

Responses STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для сравнения веток по файлам

GET /project/{userAlias}/{projectAlias}/branch/compare?compare={compareBranch}&base={baseBranch}

Сравнение веток, запрос возвращает массив измененных файлов

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/branch/compare?compare={compareBranch}&base={baseBranch}	Сравнение веток

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
compareBranch	String	Название сравниваемой ветки (Обязательный параметр)
baseBranch	String	Базовая ветка для сравнения (Необязательный параметр), в случае отсутствия данного параметра берется дефолтная ветка

Responses STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для просмотра изменений файла между коммитами

GET /project/{userAlias}/{projectAlias}/branch/compare/file?filePath={filePath}&baseHash={baseHash}&compareHash={compareHash}

Сравнение изменений файла в различных коммитах, возращает массив объектафайл

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/branch/compare/file?filePath={filePath}&baseHash={baseHash}&compareHash={compareHash}	Сравнение файла в различных коммитах

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	Strin	Псевдоним проекта

	g	
filePath	String	Путь до файла (Обязательный параметр)
baseHash	String	Базовый коммит (Обязательный параметр)
compareHash	String	Сравниваемый коммит (Обязательный параметр)

Responses STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для сравнения коммитов

GET
/project/{userAlias}/{projectAlias}/branch/commit?compareHash={compareHash}&baseHash={baseHash}

Сравнение коммитов, возвращает массив коммитов. Есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/branch/commit?compareHash={compareHash}&baseHash={baseHash}	Сравнение коммитов

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
baseHash	String	Базовый коммит (Обязательный параметр)
compareHash	String	Сравниваемый коммит (Обязательный параметр)

Responses STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод commit

Описание структуры JSON-объекта, описывающего коммит

Поле	Тип	Описание
hash	String	Хэш коммита
message	String	Название коммита
shortMessage	String	Короткое название коммита
createdAt	ZonedDateTime	Дата создания коммита
committerIdent	Object	Данные последнего коммитера
authorIdent	Object	Данные автора коммита
user	Object	Данные пользователя

Описание структуры JSON-объекта, описывающего файл

Поле	Тип	Описание
id	String	ID файла
newMode	String	Новый режим доступа файла
oldMode	String	Старый режим доступа файла
newPath	String	Новый путь файла
oldPath	String	Старый путь файла
lastCommit*	Object	Последний коммит, с которым файл был добавлен
addedLinesCount	Integer	Количество добавленных строк в файле
removedLinesCount	Integer	Количество удаленных строк в файле
fileName	String	Имя файла

filePath	String	Путь файла
fileExtension	String	Расширение файла
isImg	Boolean	Является ли файл изображением
isCollapsed	Boolean	Являются ли изменения файла сжатыми, и требующими ручной загрузки на странице
isLarge	Boolean	Являются ли изменения файла сжатыми, и требующими перехода к файлу для его просмотра
isBinary	Boolean	Является ли файл двоичным
changeType	String	Тип изменения файла
headers	Массив String	Заголовки файла
lines	Массив Object	Строки файла

Метод для получения информации о коммите

GET /project/{userAlias}/{projectAlias}/commit/{commitId}

Запрос возвращает информацию о коммите

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/commit/{commitId}	Информация о коммите

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
commitId	String	ID коммита

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для получения списка затронутых файлов в коммите

GET /project/{userAlias}/{projectAlias}/commit/{commitId}/file

Запрос возвращает массив файлов, которые были затронуты коммитом

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/commit/{commitId}/file	Список файлов, затронутых коммитом

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
commitId	String	ID коммита

Responses

STATUS 200 Пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для просмотра изменений в файле по конкретному коммиту

GET /project/{userAlias}/{projectAlias}/commit/{commitId}/diff?filePath={filePath}

Запрос возвращает информацию об изменениях в файле

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/commit/{commitId}/diff?filePath={filePath}	Информация об изменениях в файле

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
commitId	String	ID коммита
filePath	String	Путь до файла

Responses

Метод для получения информации о коммите по файлу

GET /project/{userAlias}/{projectAlias}/commit/{commitId}/for-file?filePath={filePath}

Запрос возвращает информацию о коммите по файлу

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/commit/{commitId}/for-file?filePath={filePath}	Получить информацию о коммите по файлу

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
commitId	String	ID коммита
filePath	String	Путь до файла

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод company

Описание структуры JSON-объекта, описывающего компанию

Поле	Тип	Описание
id	string	Уникальный ID компании
alias	string	Псевдоним компании
title	string	Название компании
description	string	Описание компании
url	string	Ссылка на сайт компании
contactPhone	string	Контактный телефон компании
contactEmail	string	Контактный email компании
ownerAlias	string	Псевдоним владельца компании
avatar	string	Ссылка на аватар компании
selectorTitle	string	Селектор названия компании
private	boolean	Приватная компания (true) или публичная (false)
selectorId	string	Селектор ID
selectorAlias	string	Селектор псевдонима компании
selectorOwnerAlias	string	Селектор псевдонима владельца компании
selectorColor	string	Селектор цвета

selectorHash	string	Селектор хэша

Метод для получения списка компаний

GET /company

Запрос возвращает массив компаний, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /company	Список компаний

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения команд активного пользователя

GET /company/my

Запрос возвращает массив компаний, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /company/my	Список личных компаний

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для просмотра информации о компании по названию

GET /company/{alias}

Запрос возвращает компанию

Запрос	Описание
GET /company/{alias}	Получить информацию о компании по alias

Responses

STATUS 200 - пример JSON:


```
{
  "id": "7f4829ba-f275-4336-acd1-ae416ce0c0db",
  "alias": "privatnaya-kompaniya",
  "title": "приватная компания",
  "description": "",
  "url": null,
  "contactPhone": null,
  "contactEmail": "user@gitflic.ru",
  "ownerAlias": "user1",
  "avatar": "https://gitflic.ru/upload/img/d6fef627-adb2-43b2-95ad-88f4fb54c6d2.jpg",
  "selectorTitle": "приватная компания",
  "private": true,
  "selectorId": "7f4829ba-f275-4336-acd1-ae416ce0c0db",
  "selectorOwnerAlias": null,
  "selectorAlias": null,
  "selectorColor": null,
  "selectorHash": null
}
```


STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод issueNote

Описание структуры JSON-объекта, описывающего комментарий к проблеме

Поле	Тип	Описание
id	String	Уникальный ID комментария к проблеме
message	String	Текст комментария
author	Object	Информация о пользователе, оставившего коментарий

createdAt	LocalDateTime	Дата и время создания комментария
createTimeDifference	String	Время, прошедшее с момента отправки комментария

Метод для получения всех комментариев к проблеме

GET /project/{userAlias}/{projectAlias}/issue-discussion/{localId}

Запрос возвращает массив комментариев к проблеме, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/issue-discussion/{localId}	Список комментариев к проблеме

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
localId	Long	ID проблемы

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для создания нового комментария в проблеме

POST /project/{userAlias}/{projectAlias}/issue-discussion/{localId}/create

Запрос создает комментарий к проблеме.

Запрос	Описание
POST /project/{userAlias}/{projectAlias}/issue-discussion/{localId}/create	Оставить комментарий к проблеме

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
localId	Long	ID проблемы

Пример JSON объекта

Responses

STATUS 200 Пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для удаление комментария в проблеме

DELETE /project/{userAlias}/{projectAlias}/issue-discussion/{localId}/{commentId}/delete

Удаление комментария к проблеме

Запрос	Описание
DELETE /project/{userAlias}/{projectAlias}/issue-discussion/{localId}/{commentId}/delete	Удалить комментарий

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
localId	Long	ID проблемы
commentId	Long	ID проблемы

Responses

STATUS 200 Пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод mergeRequest

Описание структуры JSON-объекта, описывающего запрос на слияние

Поле	Тип	Описание
id	String	Уникальный ID запроса на слияние
localId	Long	Локальный ID запроса на слияние
description	String	Описание запроса на слияние
title	String	Название запроса на слияние
removeSourceBranch	Boolean	Удалить исходную ветку, после того как запрос на слияние будет принят
squashCommit	Boolean	Выполнить слияние одним коммитом
assignedUsers	List	Список ответственных пользователей, назначенных на запрос на слияние
reviewers	List	Список рецензентов пользователей, назначенных на запрос на слияние
labels	List	Список лейблов, назначенных на запрос на слияние
sourceBranch	Object	Ветка, из которой будет выполнен запрос на слияние
targetBranch	Object	Ветка, в которую будет выполнен запрос на слияние
status	Object	Статус запроса на слияние
createdBy	Object	Автор запроса на слияние

createdAt	ZonedDateTime	Дата создания запроса на слияние
updatedAt	ZonedDateTime	Дата последнего изменения запроса на слияние
sourceProject	Object	Проект, из которого выполняется запрос на слияние
targetProject	Object	Проект, для которого выполняется запрос на слияние
projectAlias	String	Псевдоним проекта
userAlias	String	Псевдоним автора
canMerge	Boolean	Статус, отвечающий за то может ли ветка быть слита
hasConflicts	Boolean	Статус, отвечающий за то имеются ли кофликты у запроса на слияние

Метод получения списка всех запросов на слияние

GET /project/{userAlias}/{projectAlias}/merge-request/list

Запрос возвращает массив запросов на слияние, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/merge-request/list	Список запросов на слияние

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения информации о запросе на слияние по локальному id

GET /project/{userAlias}/{projectAlias}/merge-request/{localId}

Запрос возвращает запрос на слияние по его локальному ID

Запрос	Описание
GET /project/{userAlias}/{projectAlias}/merge-request/{localId}	Запрос на слияние по его локальному ID

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
localId	Long	Локальный ID запроса на слияние

Responses

STATUS 200 Пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для создания нового запроса на слияние

POST /project/{userAlias}/{projectAlias}/merge-request

Запрос создает запрос на слияние

Запрос	Описание
POST /project/{userAlias}/{projectAlias}/merge-request	Создает запрос на слияние

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя

projectAlias	String	Псевдоним проекта

Пример JSON объекта

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для редактирования запроса на слияние

PUT /project/{userAlias}/{projectAlias}/merge-request

Запрос изменяет запрос на слияние

Запрос	Описание
PUT /project/{userAlias}/{projectAlias}/merge-request	Изменить запрос на слияние

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта

Пример JSON объекта

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для слития запроса

POST /project/{userAlias}/{projectAlias}/merge-request/{localId}/merge

Запрос выполняет запрос на слияние

Запрос	Описание
POST /project/{userAlias}/{projectAlias}/merge-request/{localId}/merge	Выполнить запрос на слияние

Переменная пути запроса	Тип	Описание
userAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта
localId	Long	Локальный ID запроса на слияние

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для закрытия запроса на слияние

POST /project/{userAlias}/{projectAlias}/merge-request/{localId}/close

Запрос закрывает запрос на слияние

Запрос	Описание
POST /project/{userAlias}/{projectAlias}/merge-request/{localId}/close	Выполнить запрос на закрытие запроса на слияние

Переменная пути запроса	Тип	Описание
userAlias	Strin	Псевдоним пользователя

	g	
projectAlias	Strin g	Псевдоним проекта
localId	Long	Локальный ID запроса на слияние

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для отмены запроса на слияние

POST /project/{userAlias}/{projectAlias}/merge-request/{localId}/cancel

Запрос отменяет запрос на слияние

Запрос	Описание
POST /project/{userAlias}/{projectAlias}/merge-request/{localId}/cancel	Выполнить запрос на отмену запроса на слияние

Переменная пути запроса	Тип	Описание
userAlias	Strin g	Псевдоним пользователя
projectAlias	Strin g	Псевдоним проекта
localId	Long	Локальный ID запроса на слияние

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод pagination

Для запросов, которые возвращают список каких либо объектов, есть возможность пагинации. Для этого в запрос нужно добавить следующие параметры:

Параметр	Тип	Описание
page	int	Номер страницы
size	int	Количество объектов, отображаемых на странице

Если не указать количество объектов, отображаемых на странице, то по умолчанию их будет 10.

В объекте пагинации есть 2 основных объекта _embedded и page.

Структура и описание JSON-объекта _embedded:

В объекте _embedded одно поле которое содержит массив. Для различных объектов название поля может отличаться:

Для пользователей

Поле	Тип	Описание
userList	array	Массив состоящий из объектов пользователей

Для проектов

Поле	Тип	Описание
projectList	array	Массив состоящий из объектов проектов

Для команд

Поле	Тип	Описание
teamList	array	Массив состоящий из объектов команд

Для компаний

Поле	Тип	Описание
companyList	array	Массив состоящий из объектов компаний

Структура и описание JSON-объекта page:

Поле	Тип	Описание
size	number	Максимальное количество элементов, отображаемых на странице
totalElements	number	Общее количество элементов
totalPages	number	Общее количество страниц
number	number	Текущая страница

Пример параметра запроса

Запрос /user?page=2&size=2

Возвращаемый JSON:


```
{
  "_embedded": {
    "userList": [
      {
        "id": "61e8fd05-10f7-4361-909a-db72c2c33326",
        "username": "user1",
        "name": "name",
        "surname": "surname",
        "fullName": "name surname",
        "avatar": "https://gitflic.ru/upload/img/6ea2b7e0-8ad8-4356-925f-7120f2ce03c1.jpg"
      },
      {
        "id": "25a7e267-5591-4d61-8192-b9842af9804d",
        "username": "user2",
        "name": "name",
        "surname": "surname",
        "fullName": "name surname",
        "avatar": "https://gitflic.ru/upload/img/6ea2b7e0-8ad8-4356-925f-7120f2ce03c1.jpg"
      }
    ]
  },
  "page": {
    "size": 2,
    "totalElements": 12,
    "totalPages": 6,
    "number": 2
  }
```

}

Метод project

Описание структуры JSON-объекта, описывающего проект

Поле	Тип	Описание
id	String	Уникальный ID проекта
title	String	Название проекта
description	String	Описание проекта
alias	String	Псевдоним проекта
ownerAlias	String	Псевдоним владельца проекта
defaultBranch	String	Дефолтная ветка проекта
workBranch	String	Рабочая ветка проекта
siteUrl	String	Сайт проекта
owner	Object	Владелец проекта, описание структуры
language	String	Язык проекта
forEducation	Boolean	Образовательный проект(true) или обычный(false)
forkedFromId	String	ID проекта от которого был сделан форк
pullMirrorUrl	String	URL зеркала
allowedMemory	String	Допустимая память под проект
forkUrl	String	URL проекта от которого был сделан форк
httpTransportUrl	String	Ссылка на клонирование проекта по HTTPS

sshTransportUrl	String	Ссылка на клонирование проекта по SSH
private	Boolean	Приватный проект(true) или публичный(false)
mirror	Boolean	Зеркало (true) или обычный проект(false)

Описание структуры JSON-объекта, описывающего владельца проекта

Поле	Тип	Описание
alias	String	Псевдоним владельца
type	String	Тип владельца проекта

Метод получения списка проектов

GET /project

Запрос возвращает массив проектов, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /project	Список публичных проектов

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения проектов активного пользователя

GET /project/my

Запрос возвращает массив проектов, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание

GET /project/my	Список личных проектов

Responses

STATUS 200 Пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения списка проектов с участием пользователя

GET /project/shared

Запрос возвращает массив проектов, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /project/shared	Список проектов с вашим участием

Responses

Метод получения информации о проекте по названию

GET /project/{userAlias}/{projectAlias}

Запрос возвращает проект

Запрос	Описание
GET /project/{userAlias}/{projectAlias}	Получить информацию о проекте по алиасу пользователя

Переменная пути запроса	Тип	Описание
userAlias	String	Пседоним пользователя
projectAlias	String	Псевдоним проекта

Responses

STATUS 200 - пример JSON:


```
{
            "id": "06a973fb-8007-46c5-b2f0-d72a21b16087",
            "title": "bugs-bunny-fork",
            "description": "Forked repository example",
            "alias": "bugs-bunny-fork",
            "defaultBranch": "master",
            "workBranch": "master",
            "siteUrl": null,
            "owner": {
                "alias": "team1",
                "type": "TEAM"
            },
            "language": "Scala",
            "forEducation": false,
            "forkedFromId": "87c79f93-821f-4903-898f-60d08fbbca7b",
            "pullMirrorUrl": null,
            "allowedMemory": "4GB",
            "forkUrl": "https://gitflic.ru/project/team1/bugs-bunny",
            "httpTransportUrl": "https://gitflic.ru/project/team1/bugs-bunny-fork.git",
            "sshTransportUrl": "git@gitflic.ru:team1/bugs-bunny-fork.git",
            "private": false,
            "mirror": false
}
```


STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод release

Описание структуры JSON-объекта, описывающего релиз

Поле	Тип	Описание
id	String	Айди релиза
title	String	Название релиза
projectId	String	Короткое название коммита
authorId	String	Айди автора последнего изменения релиза

description	String	Описание релиза
tagName	String	Название тэга
createdAt	Date	Дата создания
updatedAt	Date	Дата редактирования
attachmentFiles	Array	Прикрепленные [файлы]
preRelease	Boolean	Пререлиз

Описание структуры JSON-объекта, описывающего прикрепленные к релизу файлов

Поле	Тип	Описание
name	String	Название файла
link	String	Ссылка на файл

Метод для получения списка релизов

GET /project/{userAlias}/{projectAlias}/release

Запрос возвращает список релизов, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /project/{ownerAlias}/{projectAlias}/release	Информация о релизе

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним пользователя
projectAlias	String	Псевдоним проекта

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения релиза по uuid

GET /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}

Запрос возвращает релиз по uuid.

Запрос	Описание
GET /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}	Информация о релизе

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним владельца
projectAlias	String	Псевдоним проекта
releaseUuid	String	Айди прелиза

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для редоктирования информации о релизе

PUT /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}

Редактирование информации о релизе по айди.

Запрос	Описание
PUT /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}	Редактирование информации

Request

Поддерживаемый формат: application/json

Структура тела PUT-запроса:

```
{
   "title": "title",
   "description": "some description",
   "tagName": "v2",
   "preRelease": false
}
```

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним владельца
projectAlias	String	Псевдоним проекта
releaseUuid	String	Айди релиза

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для создания нового релиза

POST /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}

Создание нового релиза

Запрос	Описание
POST /project/{ownerAlias}/{projectAlias}/release	Создание нового релиза

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним владельца
projectAlias	String	Псевдоним проекта

Request

Поддерживаемый формат: application/json

Структура тела POST-запроса:

```
{
    "title": "New Release",
    "description": "New some description",
    "tagName": "v24",
    "preRelease": false
}
```

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для удаления релиза

DELETE /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}

Удаление релиза

Запрос	Описание
DELETE /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}	Удаление релиза

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним владельца
projectAlias	String	Псевдоним проекта
releaseUuid	String	Айди релиза

Responses

STATUS 200 - Удаление прошло успешно.

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для скачивания файла из релиза

GET /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}/file/{fileName}

Скачать файл из релиза

Запрос	Описание
GET /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}/file/{fileName}	Получение файла из релиза по имени файла

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним владельца
projectAlias	String	Псевдоним проекта
releaseUuid	String	Айди релиза
fileName	String	Название файла

Responses

Поддерживаемый формат: application/octet-stream

STATUS 200 - Файл был отдан

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для добавления и загрузки файла к релизу

POST /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}/file

Загрузка новых файлов к релизу с помощью multipart request

Запрос	Описание

POST /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}/file	Добавление файлов к релизу

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним владельца
projectAlias	String	Псевдоним проекта
releaseUuid	String	Айди релиза

Request Поддерживаемый формат: multipart/form-data

Ключ	Тип
files	file

Responses

STATUS 200 - Добавление прошло успешно:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для удаления файлов из релиза

DELETE /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}/file

Удаление файлов из релиза, в теле запроса принимается массив строк названия файлов для удаления.

Запрос	Описание
DELETE /project/{ownerAlias}/{projectAlias}/release/{releaseUuid}/file	Удаление файлов из релиза

Переменная пути запроса	Тип	Описание
ownerAlias	String	Псевдоним владельца

projectAlias	String	Псевдоним проекта
releaseUuid	String	Айди релиза

Request

Поддерживаемый формат: application/json

Структура тела DELETE-запроса:

```
[
  "releaseFile1.zip",
  "releaseFile2.zip"
]
```

Responses

STATUS 200 - Удаление прошло успешно:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод tags

Описание структуры JSON-объекта, описывающего тег

Поле	Тип	Описание
name	String	Имя тега
fullName	String	Полное имя тега
objectId	String	Айди объекта (для легковесных тегов совпадает с хэшом коммита)
shortMessage	String	Короткое сообщение (отсутсвует для легковесных тегов)
fullMessage	String	Полное сообщение (отсутсвует для легковесных тегов)
commitId	String	Хэш коммита на который ссылается тег.

personIdent	Объект personIdent	Информация об авторе тега
lightWeight	Boolean	Легковесный тег (true) или аннотированный (false)

Описание структуры JSON-объекта, описывающего PersonIdent

Поле	Тип	Описание
name	String	Имя автора
avatart	String	Аватар автора
emailAddress	String	Email адрес
when	DateTime	Время создания тега

Метод для получения тегов проекта

GET /rest-api/project/{ownerAlias}/{projectAlias}/tag

Запрос возвращает массив тегов, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /rest-api/project/{ownerAlias}/{projectAlias}/tag/{tagName}	Получение всех тегов проекта

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для получения информации по тегу

GET /rest-api/project/{ownerAlias}/{projectAlias}/tag/{tagName}

Запрос возвращает тег

Запрос	Описание

GET /rest-api/project/{ownerAlias}/{projectAlias}/tag/{tagName}	Получение тега по имени

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод team

Описание структуры JSON-объекта, описывающего команду

Поле	Тип	Описание
id	string	Уникальный ID команды
alias	string	Алиас команды
title	string	Название команды
description	string	Описание команды
ownerAlias	string	Псевдоним владельца команды
avatar	string	Ссылка на аватар команды
selectorTitle	string	Селектор названия команды
private	boolean	Приватная команда(true) или публичная(false)
selectorId	string	Селектор ID
selectorOwnerAlias	string	Селектор псевдонима владельца команды
selectorAlias	string	Селектор псевдонима команды
selectorColor	string	Селектор цвета

selectorHash	string	Селектор хэша

Метод получения списка всех команд

GET /team

Запрос возвращает массив команд, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /team	Список публичных команд

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения личных команд пользователя

GET /team/my

Запрос возвращает массив команд, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /team/my	Список личных команд

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения команд с участием пользователя

GET /team/shared

Запрос возвращает массив команд, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /team/shared	Список команд с вашим участием

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод получения информации команды по ее названию

GET /team/{teamAlias}

Запрос возвращает команду.

Запрос	Описание
GET team/{teamAlias}	Получить информацию о команде по алиасу

Переменная пути запроса	Тип	Описание
teamAlias	String	Псевдоним пользователя

Responses

STATUS 200 - пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод user

Описание структуры JSON-объекта, описывающего пользователя

Поле	Тип	Описание
id	string	Уникальный ID пользователя
username	string	Username пользователя
name	string	Имя пользователя
surname	string	Фамилия пользователя
fullName	string	Полное имя пользователя
avatar	string	Ссылка на аватар пользователя

Список всех пользователей

GET /user

Запрос возвращает массив пользователей, есть возможность настройки количества отображаемых объектов на странице

Запрос	Описание
GET /user	Список всех пользователей

Responses

STATUS 200 пример JSON:

STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для получения текущего пользователя

GET /user/me

Запрос возвращает пользователя

Запрос	Описание
GET /user/me	Информация об авторизованном пользователе

Responses

STATUS 200 пример JSON:


```
{
  "id": "fbc3d462-a2e4-42bf-a6ec-6990c978d786",
  "username": "user1",
  "name": "name",
  "surname": "surname",
  "fullName": "name surname",
  "avatar": "https://gitflic.ru/upload/img/2731a61f-de57-44ee-b6e1-a5ac13429653.jpg"
}
```


STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Метод для вывода пользователя по uuid

GET /user/{uuid}

Запрос возвращает пользователя

Запрос	Описание
GET user/{uuid}	Информация по пользователю по его uuid

переменная пути	Тип	Описание
uuid	String	uuid пользователя

Responses

STATUS 200 пример JSON:


```
{
  "id": "fbc3d462-a2e4-42bf-a6ec-6990c978d786",
  "username": "user2",
  "name": "name",
  "surname": "surname",
  "fullName": "name surname",
  "avatar": "https://gitflic.ru/upload/img/2731a61f-de57-44ee-b6e1-a5ac13429653.jpg"
}
```


STATUS 403 - Нет прав для доступа.

STATUS 404 - Данные по запросу не найдены.

Установка OnPremise

Установка и запуск GitFlic

Для того, чтобы запустить приложение вам понадобится Java 11 (протестировано на openjdk11), Redis (протестировано на версии 6.2) и PostgreSQL (протестировано на версии 11 и 12).

Для отправки писем из сервиса необходимо сконфигурировать SMTP сервер.

Так же необходимо сгенерировать ключ key.pem для работы ssh транспорта git. После того, как выполнены предварительные подготовительные работы можно приступить к конфигурированию приложения через файл настроек по инструкции, которая находится в самом application.properties файле.

Примечание: вы можете настроить переменные окружения, находящиеся в .env файле, а затем применить их командой export $(grep -v "^#" .env | xargs)

Генерация key.pem и подключение ssh ключа:

Для генерации key.pem откройте терминал и выполните команду:

openssl genrsa -out key.pem 4096

Для генерации ssh ключа откройте терминал и выполните команду: ssh-keygen -t rsa

На консоль будет выведен следующий диалог: Enter file in which to save the key (/home/user/.ssh/id_rsa): Нажмите на клавишу Enter.

Система предложит ввести кодовую фразу для дополнительной защиты SSH-подключения (данный шаг можно пропустить). Enter passphrase (empty for no passphrase):

После этого ключ будет создан и помещён в директорию /home/user/.ssh/

Создайте директорию, в которой хотите хранить key.pem и поместите в неё ключ id_rsa без .pub расширения. Обратите внимание, если вы вставляете сгенерированный ключ без смены имени, то необходимо указать его имя в application.properties, либо переименуйте ваш приватный ключ в key.pem.

Для подключения ssh-ключа ознакомьтесь со статьей.

Установка Java 11

- Ubuntu и debian-based дистрибутивы: apt-get install default-jre
- Для Windows перейдите по ссылке
- Astra Linux (в некоторых случаях придётся скачивать и устанавливать пакет вручную)

Установка postgresql

- Ubuntu и debian-based дистрибутивы: apt-get install postgresql-12 postgresql-contrib
- Для Windows перейдите по ссылке
- Astra Linux (в некоторых случаях придётся скачивать и устанавливать пакет вручную)

Запуск postgresql:

```
systemctl enable postgresql
systemctl start postgresql
```

Конфигурация postgresql

1. Зайдите под пользователем PostgreSQL: su - postgres
2. Создайте пользователей, базу данных: createuser gitflic createdb gitflic
3. Войдите в базу данных: psql --dbname "gitflic"
4. Дайте пароль пользователю: alter user gitflic with password 'gitflic';
5. Загрузите расширение pgcrypto: create extension pgcrypto;
6. В файле /var/lib/pgsql/data/pg_hba.conf замените строку host all all 127.0.0.1/32 ident на host all all 127.0.0.1/32 md5 для использования аутентификации по паролю.
7. Перезагрузите сервер: service postgresql restart

Установка Redis

1. Установка пакетов Redis:
2. Ubuntu и debian-based дистрибутивы: apt-get install redis
3. Для Windows перейдите по ссылке
4. Запуск Redis:

```
systemctl enable redis
systemctl start redis
```

SSH

Для того, чтобы было возможным использовать remote-url вида git@gitflic.ru:gitflic/gitflic.git нужно сконфигурировать ssh на использование вашего порта. В файле /etc/ssh/ssh_config поменяйте порт на нужное значение. (Для линукс).

Подробнее про создание ssh-ключа в статье.

Запуск gitflic

Для запуска приложения в консоли необходимо выполнить следующую команду из директории, где находится исполняемый файл gitflic.jar: java -jar gitflic.jar --spring.config.additional-location=file:default-config/

Обратите внимание, что в примере указана директория с конфигурационным файлом относительно папки, в которой расположен jar пакет. Вы можете переместить конфиг файл в любую удобную вам директорию и указать к ней путь в параметре --spring.config.additional-location. Обратите внимание, что путь к директории должен оканчиваться символом /

Стандартный пользователь и пароль:

adminuser@admin.local / qwerty123

Автозапуск Gitflic при запуске системы

При необходимости запускать Gitflic каждый раз при запуске системы, можно использовать systemd сервис. Создайте файл /etc/systemd/system/gitflic.service и поместите в него следующее:

```
[Unit]
Description=Gitflic
After=network-online.target postgresql.service redis.service
Requires=network-online.target postgresql.service redis.service


[Service]
User=user
Group=group
WorkingDirectory=/opt/gitflic
EnvironmentFile=/opt/gitflic/env
ExecStart=/usr/bin/java -jar gitflic.jar --spring.config.additional-location=file:/opt/gitflic/application.properties/
ExecStop=/bin/kill -s 15 $MAINPID
StandardOutput=journal
StandardError=journal
Restart=always
SuccessExitStatus=143

[Install]
WantedBy=multi-user.target
```

Поместите gitflic.jar, application.properties и env в директорию, указанную в WorkingDirectory, либо измените на необходимую вам. В параметрах User и Group укажите имя и группу пользователя, от имени которого будет запускаться сервис.

Добавить сервис в автозагрузку: systemctl enable gitflic

Запустить сервис: systemctl start gitflic

Запуск GitFlic в Docker

Для того, чтобы запустить приложение вам понадобится Java 11 (протестировано на openjdk11), Redis (протестировано на версии 6.2) и PostgreSQL (протестировано на версии 11 и 12).

Для отправки писем из сервиса необходимо сконфигурировать SMTP сервер.

Так же необходимо сгенерировать ключ key.pem для работы ssh транспорта git.

Для развёртывания приложения в Docker, вам понадобится настроить переменные окружения, находящиеся в .env файле, и сконфигурировать само приложение по инструкции в конфигурационном application.properties файле.

После настройки выполняем следующую команду: `docker compose up --build`

Стандартный юзер и пароль: adminuser@admin.local qwerty123

Для подключения ssh-ключа ознакомьтесь со статьей или соответствующим разделом (SSH ключи).

Запуск агента (раннера)

(Runner - в контексте с англ. Агент)

После того, как приложение было запущено, необходимо зарегистрировать и запустить агент. Для регистрации раннера скопируйте регистрационный токен, перейдя в дашборд Runners.

Директория с конфигурационным файлом application.properties должна находиться в той же директории с jar файлом. Либо можно указать путь к директории флагом `--default-config`

Перейдите в папку runner и запустите скрипт reg.sh:

```
./reg.sh -i runner.jar -i IP -p PORT -t TOKEN
```

Либо воспользуйтесь коммандой:

```
java -jar "$JAR" register --url http://"$IP":"$PORT"/-/runner/registration --registration-token "$TOKEN";
```

Запуск раннера:

```
./startup.sh -i runner.jar -c path/to/application.properties
```

Либо воспользуйтесь коммандой:

```
java -jar "$JAR" start --config=path/to/application.properties/
```

Не забудьте указать полный gitflic.transport.url в application.properties gitflic’a в виде http(s)://IP:PORT