

Принципы работы с GitLab

Версия: 0.2
Дата: 09.05.2020
Исполнители: Карапетян Т.В.

Содержание

- 1. Установка SSH-соединения с сервером GitLab**
- 2. Инициализация репозитория**
- 3. Работа с репозиторием в рамках одной ветки**
 - a) Фиксация локальных изменений (commit).
 - b) Добавление локального файла в репозиторий
 - c) Загрузка commit'ов на сервер (push).
 - d) Обновление локального репозитория (pull).
- 4. Ветвление репозитория**
 - a) Создание новой ветки репозитория.
 - b) Переключение между ветками.
 - c) Слияние веток.
 - d) Защищенные ветки
- 5. Методика работы с ветками**
- 6. Создание релиза**
- 7. Игнорирование файлов**
- 8. Справочная информация**

1. Установка SSH-соединения с сервером GitLab

Порядок действий для генерации SSH-ключа используя Git Bash:

1. Сгенерировать RSA-ключ консольной командой

ssh-keygen -t rsa -b 2048 -C "email@example.com"

2. Опционально изменить путь сохранения ключа и пароль

Generating public/private rsa key pair.

Enter file in which to save the key (~/.ssh/id_rsa): [enter to keep path]

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

3. Скопировать публичный ssh-ключ

cat ~/.ssh/id_rsa.pub | clip

1. Установка SSH-соединения с сервером GitLab

Порядок действий для генерации SSH-ключа используя Tortoise Git:

1. Генерация ssh-ключа

Запустить *PuTTYgen*. В области *Actions* выбрать *Generate* и поводить курсором мыши внутри окна приложения, после чего появится сгенерированный публичный ключ, который необходимо скопировать.

2. Указание пароля и сохранение ключа в файле (опционально)

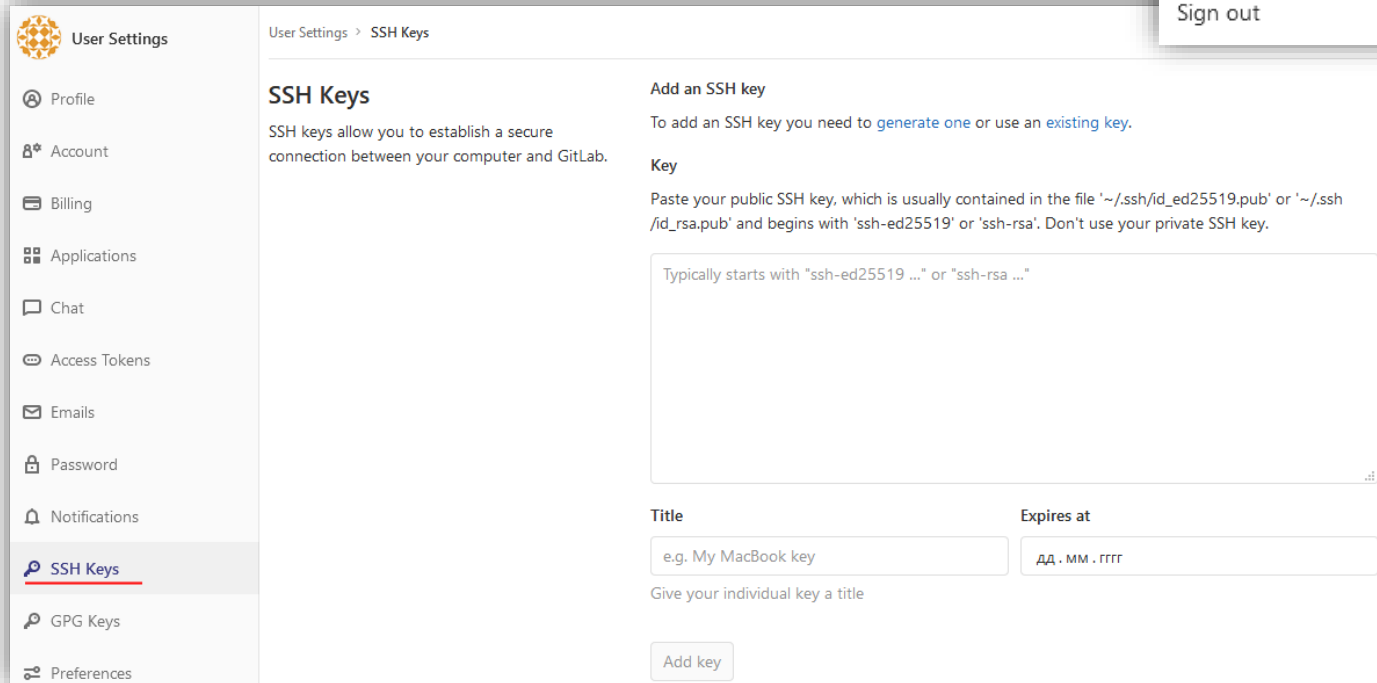
После генерации ключа, можно сохранить публичный и приватный ключи в отдельных файлах. Также при сохранении можно указать пароль для ключа в поле *Key passphrase*. При сохранении приватного ключа рекомендуется указать пароль, т.к. данный файл служит для авторизации конкретного пользователя.

1. Установка SSH-соединения с сервером GitLab

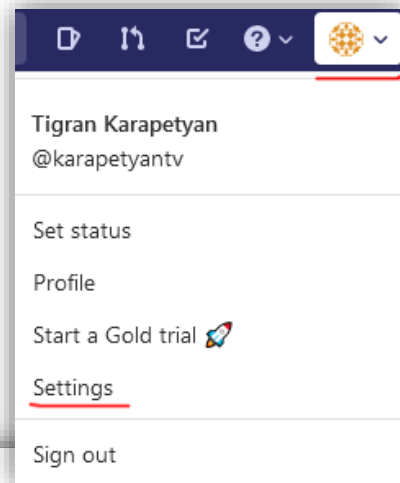
4. Привязать скопированный ключ к своему профилю в GitLab

Независимо от того ведется работа с Git Bash или Tortoise Git необходимо авторизовавшись в GitLab перейти в *settings* -> *SSH Keys* и вставить скопированный ранее публичный ключ в пустую область.

Опционально можно задать ключу название в поле Title. Поле *Expires at* оставить пустым, тогда ключ будет действовать неограниченно по времени. После чего нажать *Add key*



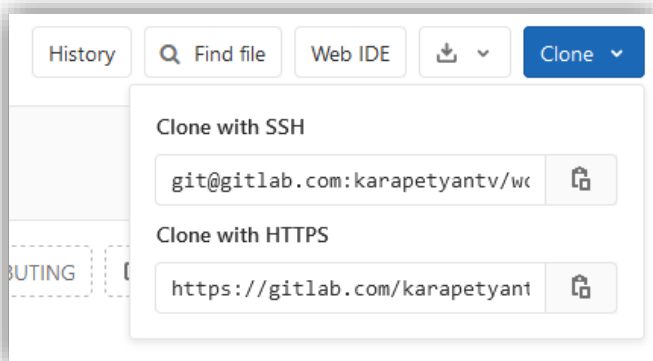
The screenshot shows the GitLab User Settings page for 'SSH Keys'. The left sidebar contains a list of settings: Profile, Account, Billing, Applications, Chat, Access Tokens, Emails, Password, Notifications, **SSH Keys** (highlighted), GPG Keys, and Preferences. The main content area is titled 'SSH Keys' and includes a description: 'SSH keys allow you to establish a secure connection between your computer and GitLab.' Below this is a text area for pasting the public SSH key, with a hint: 'Typically starts with "ssh-ed25519 ..." or "ssh-rsa ..."'. At the bottom, there are two input fields: 'Title' (with the example 'e.g. My MacBook key') and 'Expires at' (with the example 'ДД . ММ . ГГГГ'). An 'Add key' button is located at the bottom right of the form.



2. Инициализация репозитория

Для того, чтобы загрузить репозиторий с сервера на локальную машину необходимо:

1. В web-интерфейсе GitLab скопировать ссылку на репозиторий



2. Перейти в директорию, в которой необходимо инициализировать репозиторий:

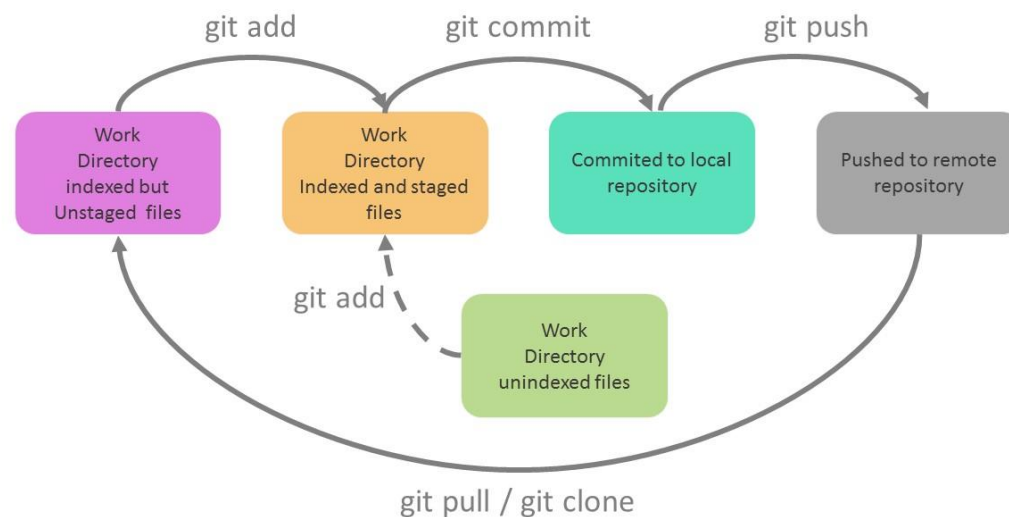
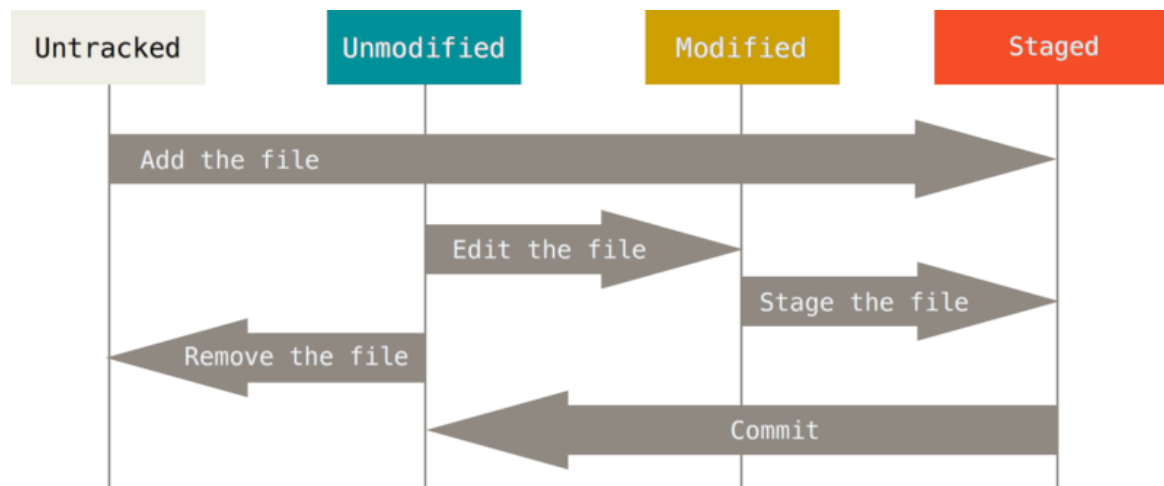
Для **GitBash** выполнить команду

git clone [скопированная ссылка]

В Tortoise Git

Вызвать контекстное меню, выбрать пункт *git clone* и в появившемся окне вставить скопированную ранее ссылку

3. Работа с репозиторием в рамках одной ветки



3. Работа с репозиторием в рамках одной ветки

Наиболее распространенные действия при работе с репозиторием:

Фиксация локальных изменений (commit)

git commit -m "Описание коммита"

Коммит должен быть атомарным, т.е. одно изменение – один коммит.

Тема сообщения начинается с заглавной буквы, не завершается точкой и ограничивается 50 символами.

Основной текст отделяется от темы пустой строкой. Каждая строка ограничена 72 символами.

В тексте сообщения предпочтительно описать что было изменено и причину изменений. Если внесенные изменения нарушают работу какого-либо другого функционала, необходимо также указать это в тексте сообщения.

Так как текст коммита занимает несколько строк, необходимо предварительно сформировать текст сообщения в файле (в случае Tortoise Git используется отдельное окно редактора и отсутствует необходимость в файле) и использовать его при формировании коммита. Для этого необходимо воспользоваться следующей командой:

git commit -F Имя Файла

Отмена нежелательного коммита

git revert commit_id

commit_id, а также список коммитов может быть открыт командой *git log*

3. Работа с репозиторием в рамках одной ветки

Добавление нового локального файла в репозиторий:

git add [имя_файла] [. – добавить все файлы текущей директории]

Для того чтобы изменения вносимые в какой-либо файл отслеживались в репозитории этот файл необходимо предварительно добавить файл в репозиторий при помощи команды *git add*. Новые создаваемые файлы по умолчанию не включаются в репозиторий.

При использовании командной строки при работе с git, необходимо добавлять файл, в котором произведены изменения перед каждым коммитом. В случае если работа ведется через Tortoise Git или встроена в IDE, то данная операция будет выполняться автоматически.

Загрузка локальных коммитов на сервер (push)

git push origin [имя_ветки]

Загрузка всех локальных коммитов (не загруженных ранее) из локального репозитория в удаленный. Если указать имя ветки, то будут загружены изменения только этой ветки, иначе будут загружены локальные изменения во всех ветках.

Обновление локального репозитория (pull)

git pull origin [имя_ветки]

Загрузка всех изменений из удаленного репозитория в локальный. Если указано имя ветки, то загрузится только конкретная ветка, иначе будут обновлены все ветки локального репозитория.

4. Ветвление репозитория

Создание новой ветки репозитория

git branch имя_ветки – создание новой ветки (без переключения)

git checkout -b имя_ветки – создание новой ветки и переключение на неё

Создает новую ветку репозитория, идентичную той ветке, из которой вызывается одна из представленных выше команд. В новой ветке будут все файлы и все коммиты исходной ветки на момент создания новой ветки.

Переключение между ветками

git checkout имя_ветки

Переводит репозиторий в состояние, соответствующее указанной ветке. Для того чтобы посмотреть список всех веток локального репозитория используется команда *git branch -l*, чтобы посмотреть список веток удаленного репозитория необходимо использовать ключ *-r*.

Защищенные ветки

GitLab позволяют сделать ветки защищенными. Защищенную ветку запрещено удалять. В защищенную ветку запрещено выполнять push всем кроме пользователей с соответствующими разрешениями. Соответствующие разрешения выдаются руководителям проекта. Как правило статус защищенной присваивается веткам, отражающим релизное состояние проекта (например ветка master). Внесение изменений в такие ветки осуществляется посредством процедуры слияния веток, описанной далее.

4. Ветвление репозитория

Слияние веток

Слияние веток – перенос всех коммитов из одной ветки, в другую. В GitLab для осуществления слияния используется merge request (далее MR), представляющий собой запрос на слияние веток.

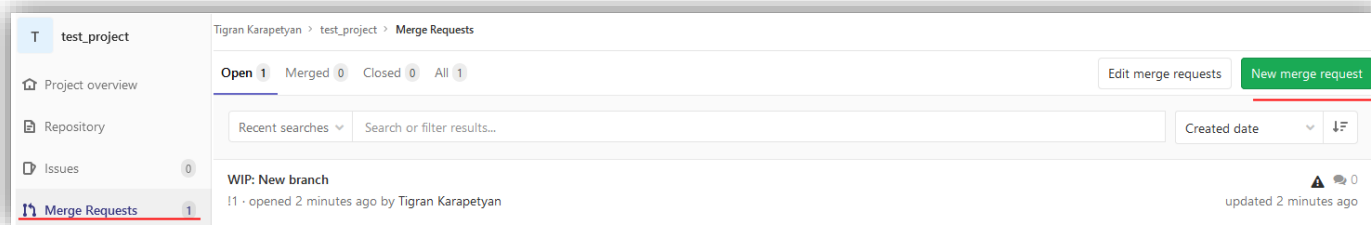
MR позволяет обсудить вносимые изменения с другими разработчиками. Также данный механизм позволяет предотвратить внесение некорректных изменений, нарушающих функционирование проекта.

Перед созданием MR необходимо выполнить push текущих коммитов в source-ветку. После этого в web-интерфейсе GitLab перейти в **Merge Requests -> New merge request** и выбрать source и target ветки. Затем нажав **Compare branches and continue** необходимо дать название MR дать его короткое описание и указать участника проекта, которому назначен данный MR в пункте **Assignee**, после чего нажать кнопку **Submit merge request**. После этого созданный MR будет отображаться в списке **Merge Requests** где выбрав конкретный MR можно принять участие в его обсуждении.

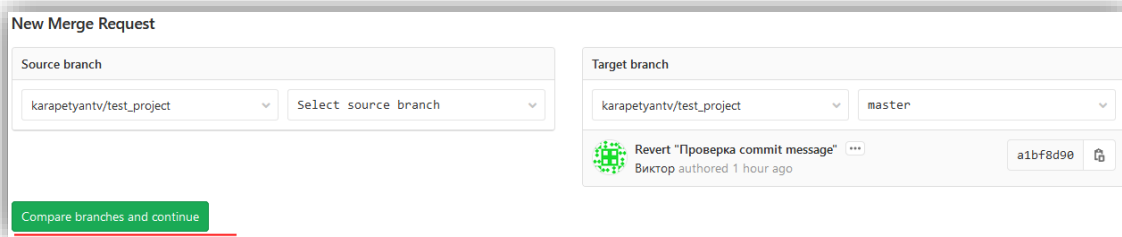
MR может быть закрыт, таким образом предлагаемые к внесению изменения будут отклонены. Если же изменения принимаются руководителем проекта, то MR будет принят.

4. Ветвление репозитория

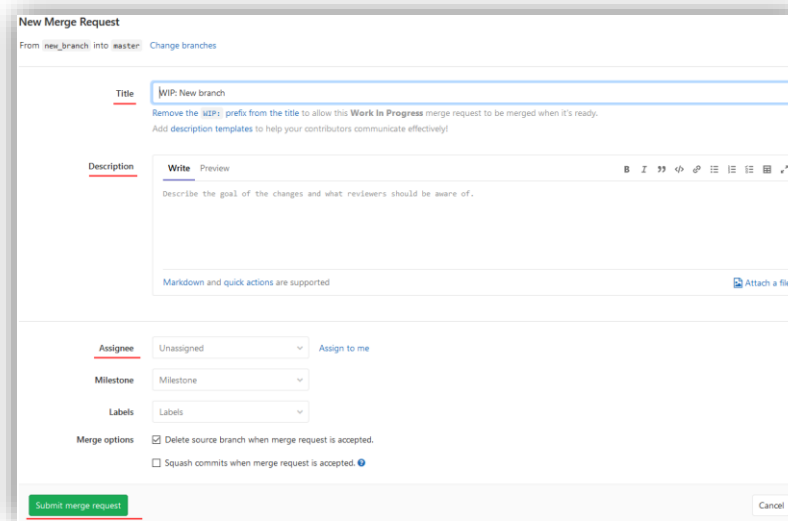
1. Создание MR



2. Выбор source и target веток



3. Выбор названия, описания и участника, которому назначен MR



5. Методика работы с ветками

В каких случаях создавать новую ветку репозитория?

Создавать новую ветку репозитория стоит каждый раз когда начинается работа над каким-либо новым функционалом, исправление ошибок, эксперимент и т.д.

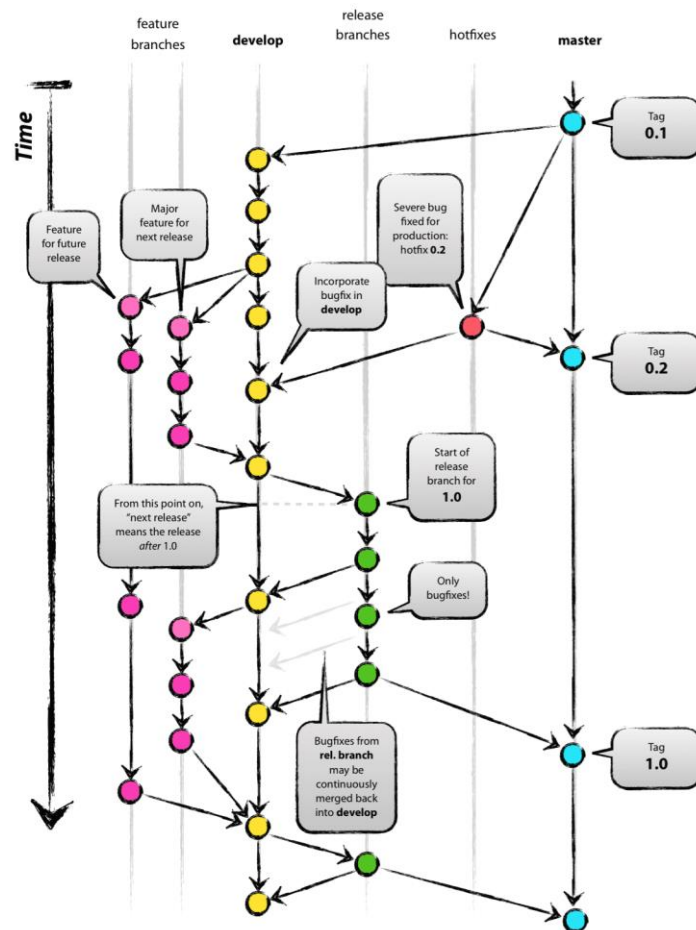
Перед созданием новой ветки необходимо четко обозначить конкретный функционал или исправление ошибки, которое будет выполняться в рамках данной ветки. Все изменения должны быть направлены исключительно на решение обозначенной задачи. **Решение нескольких задач в рамках одной ветки недопустимо!**

Обязательные ветки

В каждом репозитории обязательно должны быть следующие защищенные ветки:

master – релизные версии проекта;

dev – ветка для разработки.



5. Методика работы с ветками

Как именовать ветки?

Название ветки должно состоять из группового токена, отражающего характер работы, которая проводится в ветке и кратко сформулированной основной части названия, отражающей смысл работы, проводимой в ветке.

Предлагается использовать следующие групповые токены:

- *wip* – долгосрочная разработка, может быть несколько слияний в ходе разработки без удаления ветки;
- *feat* – новый функционал или расширение имеющегося. При слиянии ветка удаляется разработки какого-либо функционала;
- *bug* – исправление ошибок;
- *junk* – одноразовая ветка для экспериментов.

Основная часть названия

Основная часть названия должна ёмко отражать суть проводимой работы. Основная часть может состоять из нескольких частей, разделяемых символом /, формируя таким образом составное название. Однако одновременно в репозитории не может быть двух веток с составной основной частью, имеющих одинаковую начальную часть составного имени (например *feat/a/b* и *feat/a/c*).

Допускается использование нескольких слов в одной части названия. В таком случае слова следует разделять -.

Примеры названий веток:

- *wip/plots*;
- *feat/benchmark-enhancement/rastrigin*;
- *feat/searching-behaviour*.

6. Создание релиза

Создание релиза

Релиз представляет собой фиксацию определенного состояния master-ветки. В GitLab для этого используется инструмент Tags/Releases.

Для создания нового релиза необходимо в боковой панели web-версии GitLab перейти в **Project overview -> Releases -> New release**. Далее указать имя релиза, его описание. **Создавать релиз обязательно только из master ветки!**

Также GitLab позволяет помимо исходного кода добавлять в релиз ссылку на внешние файлы, содержащие, например, документацию или бинарные файлы собранного проекта. Для этого после создания релиза необходимо также перейти в **Project overview -> Releases -> New release** и нажать пиктограмму редактирования релиза. После чего внизу страницы будет доступно поле Release assets, в котором можно указать внешнюю ссылку на файлы, которые необходимо прикрепить к релизу

Release assets

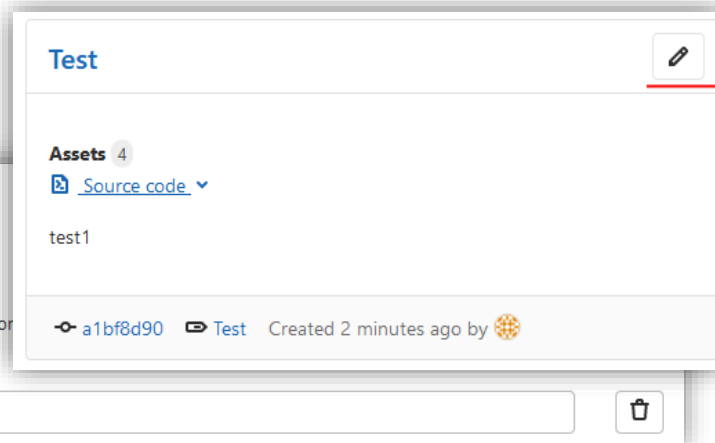
Add [assets](#) to your Release. GitLab automatically includes read-only assets, like source code and release evidence.

Links

Point to any links you like: documentation, built binaries, or other related materials. These can be internal or external links from

URL

Link title



7. Игнорирование файлов

При работе с системой контроля версий Git можно исключать некоторые локальные файлы из репозитория. Для этого используется файл с именем *.gitignore*. В таком файле указываются пути к файлам или директориям, изменения в которых не будут отслеживаться в репозитории. Такой файл может находиться как в основной директории репозитория, так и в каждой отдельной субдиректории.

При указании путей файлов могут быть использованы следующие спецификаторы:

- # - комментирование строки;
- / - разделитель пути к директории. Если в начале или середине пути файла есть данный символ, то путь к файлу воспринимается относительно конкретного файла *.gitignore*. Если же такой символ отсутствует, то будет проигнорирован любой файл с совпадающим именем в текущей директории или на уровнях ниже текущей директории;
- * - все файлы, совпадающие с паттерном. Может также использоваться для исключения всех файлов определенного расширения (*.zip);
- ! – отменяет игнорирование какого-либо файла, перед которым установлен. Например, необходимо игнорировать в директории все файлы, кроме одного. Тогда все файлы директории игнорируются *, а необходимый файл исключается из спецификатора символом !.

7. Игнорирование файлов

Какие файлы следует игнорировать?

В число игнорируемых файлов следует очевидно включать файлы, изменения в которых нет необходимости отслеживать. Как правило относят почти все файлы, не являющиеся файлами исходного кода, т.к. только эти изменения являются принципиальными для контроля.

Примеры файлов, которые рекомендуется исключать:

- файлы логов;
- файлы кэша;
- файлы и артефакты сборки;
- директория build;
- конфигурационные файлы окружения конкретного пользователя;
- любые другие выходные файлы генерируемые в результате работы программы. Такие файлы распространяются отдельно от контроля версий;

8. Справочная информация

1. Официальная документация Git - <https://git-scm.com/book/ru/v2>
2. Тэг **git** на StackOverflow - <https://stackoverflow.com/questions/tagged/git>
3. Подробнее об оформлении коммитов - <https://habr.com/ru/post/329992/#132-kratkiy-obzor-struktury-soobscheniy-kommitov>
4. Сборник распространенных файлов .gitignore - <https://github.com/github/gitignore>

Спасибо за внимание!