Публикация на тему

Git

Основы git, git rebase, git merge

Анотация

-
-

Автор

Михалькевич Александр Викторович

Публикация

Наименование Git

Автор А.В.Михалькевич

Специальность Основы git, git rebase, git merge,

Анотация -

Anotation in English -

Ключевые слова git

Количество символов 15430

Содержание

Введение

1 Основы GIT

Git — распределённая система управления версиями.

Скачать git можно по этой ссылке https://git-scm.com/downloads

Благодаря Git и системы ветвления у разработчиков имеется возможность работать в команде над одним проектом.

часто используемые команды git:

Обычно разработка начинается с клонирования репозитория, либо создания репозитория в папке с проектом.

1 .1 Клонирование

Предположим, у нас имеется git-репозиторий с проектом, который необходимо развернуть

локально - https://github.com/mikhalkevich/laravel_blog

```
cd domain // перешли в папку, где будем разварачивать проект
git clone https://github.com/mikhalkevich/laravel_blog
```

Далее открываем проект в своей любимой IDE и решаем поставленные задачи.

1 .2 Инициализация пустого репозитория

Git-репозиторий сперва необходимо создать локально, предворительно перейдя в папку с проектом:

```
cd project
git init
```

1 .3 Добавление в область видимости git, фиксация, pull и push

После внесения изменений в проект, новые файлы должны быть добавлены в область видимости git, а изменённые должны быть зафиксированы, для этого имеются консольные команды:

```
git add *
git commit -m "first commit"
git push
```

Если есть права доступа в репозиторий, то изменения можно залить с помщью команды `push`:

```
git push
```

Соответственно, команда `pull` предназначена для скачивания изменений с удалённого репозитория:

```
git pull
```

2 Тэгирование

Тэгирование - это способ добавить постоянную метку к коммиту. Существует много причин для тегирования, но две наиболее распространенные — это отметка точной версии кода, которая выпускается для клиентов, и удобный способ вернуться к определенной версии кода.
Создание тэга:

```
git tag version-1-0-beta
```

Вывод списка тэгов:

```
git tag --list
```

Удаление тэга:

```
git tag --delete version-1-0-beta
```

3 Ветвление

Зачем нужны ветки? Они позволяют разрабатывать код, который изолирован от стабильной кодовой базы, пока он не будет готов к добавлению в эту кодовую базу. Вот типичный рабочий процесс, демонстрирующий, как это происходит:

1. Код продукта находится в репозитории Git. Стабильная версия этого кода находится в ветке внутри этого репозитория, которая называется main.

2. Вам поручено написать функцию для вашего продукта, которая позволяет пользователям входить в систему.

3. Вы создаете ветку login-feature.

4. Вы переключаетесь на ветку login-feature.

5. Вы редактируете файлы и добавляете один или несколько коммитов в эту ветку.

6. Другие члены команды просматривают изменения в этих коммитах и дают вам обратную связь.

7. Вы добавляете еще один коммит, который включает обратную связь.

8. Ваш руководитель группы одобряет вашу работу, заявляя, что функция входа была реализована правильно. Ваша команда QA также может одобрить вашу работу.

9. Вы объединяете ветку login-feature с основной веткой. Это означает, что все коммиты, которые вы сделали в ветку login-feature, теперь также являются частью основной ветки. Функция входа в систему теперь является частью основной кодовой базы продукта.

10. Поскольку ветвь функции входа в систему была объединена и больше не служит никакой цели, вы можете безопасно удалить ее.

Давайте изучим команды Git, которые понадобятся для выполнения описанного ранее рабочего процесса. Вот как создать новую ветку с именем login-feature:

```
git branch login-feature
```

Эта команда выведет список созданных ранее веток:

```
git branch
```

В git существует две команды - `checkout` и `switch`, с помощью которых можно переключаться в созданную ветку:

```
git checkout login-feature
git switch login-feature
```

3 .1 Слияние веток

Перед слиянием необходимо убедиться в том, что все изменения в ветках зафиксированы.

Чтобы объединить все коммиты, которые находятся в login-feature, в main, сперва необходимо переключиться в целевую ветку main:

```
git checkout main
```

Затем выполнить слияние, указав исходную ветку

```
git merge login-feature
```

При необходимости,

```
git branch -—delete login-feature
```

Если вы не объединили ветку перед тем, как попытаться удалить ее, Git не удалит эту ветку. Однако с помощью параметра `--force` можно удалить ветку принудительно:

```
git branch -—delete -—force login-feature
```

3 .2 Конфлиткы и решение конфликтов при слиянии

При попытке объединить одну ветку с другой, иногда мы сталкиваемся с так называемым конфликтом слияния. Это означает, что кто-то другой отредактировал те же строки того же файла, что и вы, и что он уже объединил свою ветку с основной до того, как вы получили возможность объединить свою ветку с основной. Когда вы пытаетесь объединить свою ветку, Git не уверен, следует ли сохранить изменения, внесенные другим разработчиком, или изменения, которые вы пытаетесь объединить.

Чтобы продолжить слияние, вам сначала нужно разрешить конфликт слияния. Есть несколько способов сделать это.

1) Использование специальных инструментов Git GUI, таких как Sourcetree (macOS и Windows) или Sublime Merge (Linux, macOS и Windows), являются самым простым и интуитивно понятным способом решения конфликтов слияния.

2) Некоторое разработчики предпочитают разрешать конфликты слияния вручную, используя команды терминала Git и текстовый редактор.

3) У пользователей GitLab есть другой вариант: вы можете использовать встроенный графический инструмент разрешения конфликтов слияния GitLab. Независимо от того, какой подход вы выберете, вам придется каким-то образом сообщить Git, какие изменения принять, какие отбросить и когда продолжить слияние.

4 Merge request на практике

Итак, мы склонировали проект, и начинаем работать над задачей.

По умолчанию исходный код находится в ветке `master`. Поэтому прежде чем приступать к задаче, необходимо создать ветку под текущую задачу. Название ветки имеет значение:
- префикс `feet/` означает feature, он всегда должен присутствовать в начале названия;
- номер задачи, который идёт сразу после префикса и

- через знак "_" нижнее подчёркивание описываем одним/двумя словами на английском языке суть задачи.

Пример:

```
git branch feet/4546_form
git checkout feet/4546_form
```

Теперь можно приступать к задаче. Вносим необходимые изменения в проект. Например, в файл test.php добавили код

```
phpinfo();
```

> Примечание. Если приступили к задаче и забыли переключиться на нужную ветку - не проблемма. Это можно сделать и на текущем этапе. Главное следить за тем, чтобы локально была установлена текущая версия проекта (почаще делаем `git pull` из ветки master), иначе могут возникнуть конфликты с `merge` (слиянии веток). Если в удалённом репозитории настроен `merge request`, то попытка залить в master ни к чему не приведёт.

Далее необходимо залить текущую ветку с изменениями на удалённый репозиторий. Команды:

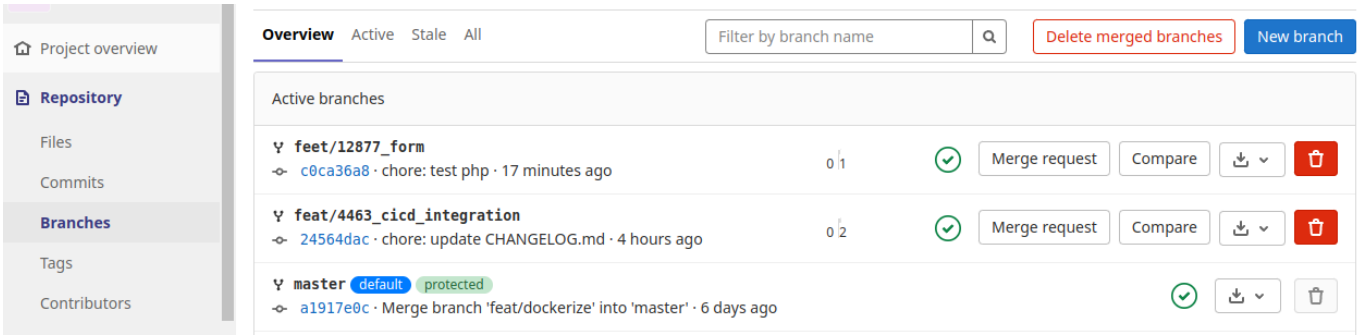
```
git add *
git commit -m "chore: test php"
```

> Примечание. Если задача ещё выполнена, то в начале комита необходимо прописать `chore`, если задача выполнена - то, в `message` пишем что угодно: что делали, то и пишем (желательно использовать английские буквы).

Теперь необходимо изменения из ветки задачи залить в master - тогда мы сможем протестить задачу на удалённом сервере. Для этого необходимо авторизоваться в gitlab и перейти на страницу с ветками проекта.

Repository -> branches

Если всё правильно сделали, то увидим следующее:

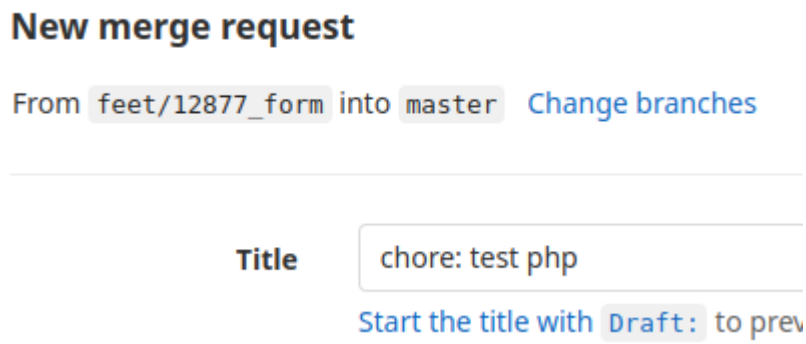


Далее необходимо сделать Merge request. Нажимаем на кнопку `Merge request` своей ветки и переходим на соответстующую страрицу.

> Примечание. Merge request можно сделать также перейдя по соответстующей ссылке в репозитории, но тогда в выпадающем списке веток необходимо выполнить `compare branches` для своей векти, которую предворительно нужно выбрать из

выпадающего списка.

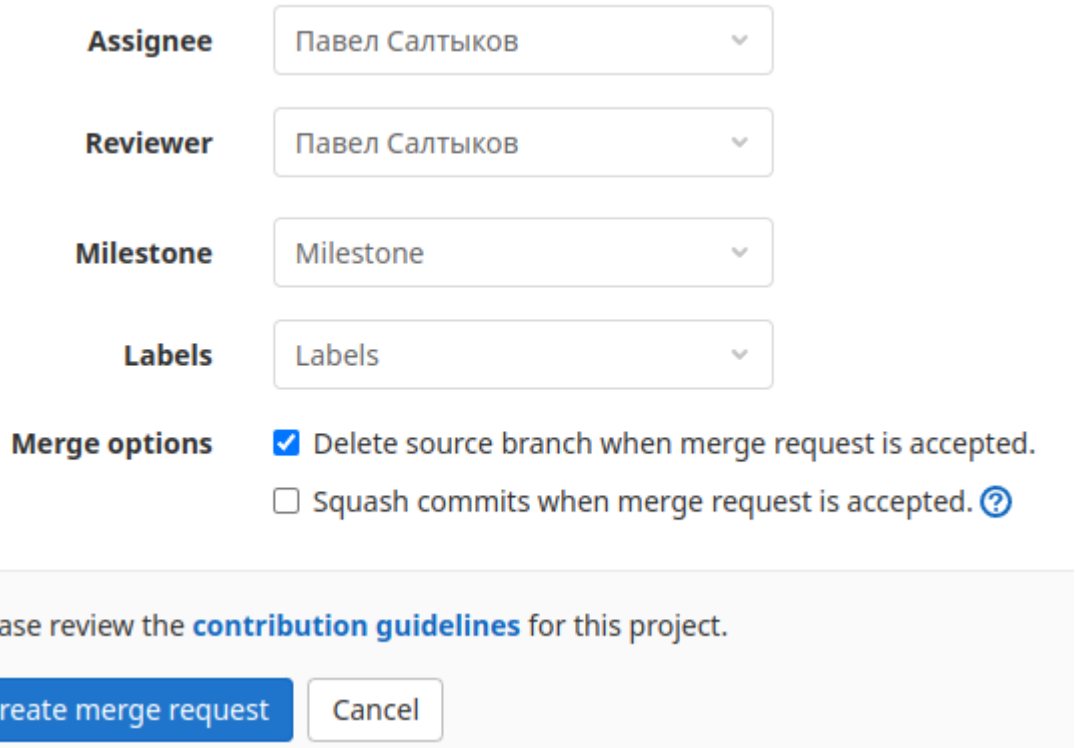
Если всё правильно сделали, увидим следующее:



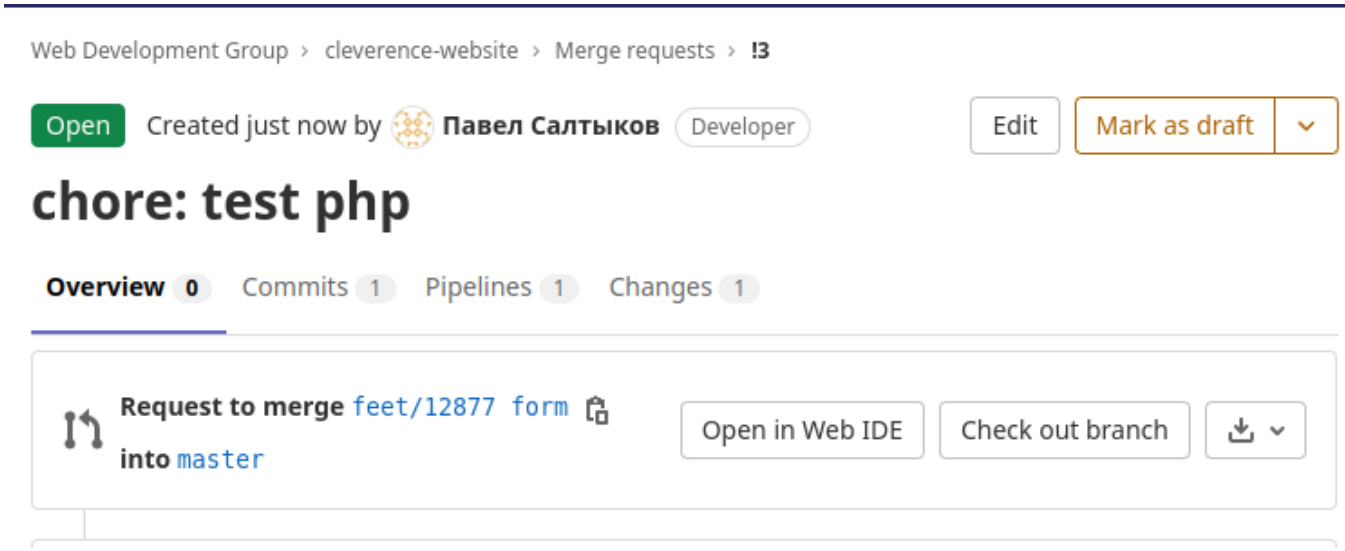
Меняем Title на следующий:

```
[12877] Form adding
```

Находясь на этой же странице, чуть ниже, выбираем Assigne (текущий пользователь, т.е. тот, кто делает задачу), Reviewer (тот, кто будет проверять задачу) и нажимаем `Merge request`



`Merge request` создан. Но если задача не доделана, помечаем её как черновик `Mark as draft`



Merge request в состоянии draft позволяет отредактировать Title и Description и проверить какие изменения были внесены в код.

Если задача выполнена - переводим Merge request в состояние `Mark as ready`. После чего возвращаемся в локальный репозиторий и создаём ещё один комит, но уже с другим названием: `feet: [4546](href_link_to_task)`, где в квадратных скопках - название, в круглых - ссылка на задачу.

Заключение

Список использованных источников

Приложения