#### 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 23845

## Содержание

#### Введение

- 1 <u>Основы GIT</u>
  - 1.1 Клонирование
  - 1.2 Инициализация пустого репозитория
  - 1.3 Добавление в область видимости git, фиксация, pull и push
  - 1.4 <u>Состояния файлов в репозиториях Git</u>
- 2 Тэгирование
- 3 Ветвление
  - 3.1 Слияние веток
  - 3.2 Конфлиткы и решение конфликтов при слиянии
- 4 История коммитов
- 5 Визуализация различий
- 6 Действия с файлами и отмена изменений
- 7 Командная удалённая работа
  - 7.1 Clone или fork
  - 7.2 Merge request или pull request

<u>Заключение</u>

-

# 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:

## **GIT Cheatsheet** 000 // Repository qit init // Initialize a new Git repo git clone <repo-url> // Clone a repo from a URL // Show changes status git status git add <file> // Add changes to staging git commit -m "Message" // Commit changes with a message // View commit history qit loq // Branching git branch // List branches // Create a new branch git branch <branch-name> git checkout <branch-name> // Switch to a branch git merge <branch-name> // Merge changes from a branch git branch -d <branch-name> // Delete a branch // Remote Repositories git remote // List remotes git remote add <name> <url> // Add a remote git push <remote> <branch> // Push changes to a remote git pull <remote> <branch> // Pull changes from a remote // Undoing Changes git pull // Fetch and merge changes git fetch // Fetch changes without merging // Discard changes git reset --hard HEAD git revert <commit-hash> // Revert changes in a commit rammcodes in 🔟 🔽 💆 🍘 Ram Maheshwari

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

```
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
```

# 1.4 Состояния файлов в репозиториях Git

С точки зрения репозитория Git файл может быть в одном из трех состояний.

- 1 Неотслеживаемые
- Отслеживаемые:
- 2 Индексированный
- 3 Неизменный

Изменение файла запускает цикл 1,2,3

## 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 История коммитов

Для просмотра истории коммитов текущей ветки можно воспользоваться командой

git log

Данную команду можно выполнять с флагами: --all - покажет историю коммитов всех веток, --oneline - выведет всё красиво (как буд-то по умолчанию, мы хотим видеть не красивый вывод консоли:) и --graph - указание отображать комиты ввиде графа. Эти флаги можно использовать совместно:

git log --oneline --all --graph

Обратите внимание, что вызов данной команды (так же как и комадны git diff) выводит ответ в встроенный редактор с разбиением на страницы. В данном редакторе доступны следующие действия и кнопки:

- Q выход, отмена разбиения вывода на страницы
- ↑ переход по списку вверх
- ↓ переход по списку вниз

# 5 Визуализация различий

Визуализация различий - это поиск различий или сравнение файлов из рабочего каталога с индексами файлов в Git.

Конечно, существуют инструменты визуального сравнения (утила **difftool** или программа **SublimeMerge**). Однако здесь будем придерживаться инструментов, которые Git представляет из коробки, по нескольким причинам: программой на реальном боевом сервере не воспользуешься, утила тоже может быть не доступна, и все приложения визуального сравленния под копотом используют стандартные git команды. В частности, команда

git diff

Данная команда сравнит версии файлов в индексе и в рабочем каталоге. В консоль выводит ответ по одному файлу за раз, разделенный на отдельные области изменений.

Чтобы сравнить содержимое базы данных объектов с содержимым индекса нужно указать флаг --cached:

git diff --cached

Различия между ветками:

git diff master develop

Внимание: если не указать имя второй ветки, то первая ветка будет сравниваться с рабочим каталогом.

Эту же команду можно использовать в поиске различий в комитах:

```
git diff 38a9232 879C123
```

В целом, такие git-команды как status, branch, log и diff называются безопасными, т.е. они только запрашивают информацию у репозитория, но никак не меняют его.

# 6 Действия с файлами и отмена изменений

Действия с файлами: создание, изменение, перемещение и удаление.

#### Удаление файлов из репоизтория Git

Git удаляет файлы из рабочего каталога и индекса, но никогда из объектной базы.

git rm file.md

#### Переименование и перемещения файлов

```
git mv file1.md file2.md
```

где file1.md - это старое имя файла, file2.md - новое имя файла.

Удаление, перемещение, переименование и изменение файла необходимо закомитить.

git commit -m "message for commit"

#### Отмена изменений

Для отмены изменений в рабочем каталоге и индексе можно воспользоваться командой resotore

```
git restore readme.md
```

Команда git resotre отменит любые изменения в рабочем каталоге. Что является противоположностью команды git add. Данную команду с флагом --staged используют для восстановления файлов из объектной базы данных в индекс, что является противоположностью команды git commit.

git restore readme.md --staged

#### Удаление комитов с помощью команды reset

git reset --hard

#### **HEAD**

К комитам можно обращаться с помощью ключа HEAD и операторов ~ и ^. Пример:

```
git reset HEAD~1 // обращение к прямому родителю git reset HEAD~3 // родитель родителя родителя
```

```
git reset HEAD^1 // первое поколение (или ветка) после слияния git reset HEAD^2\sim1 // первый родитель второго поколения
```

HEAD можно использовать и в команде diff:

```
git diff HEAD~1 HEAD
```

где HEAD~1 - родительский комит, а HEAD - текущий.

# 7 Командная удалённая работа

Основное преимущество работы с репозиториями git становится заметно после того, как к проекту подключается несколько разработчиков и у проекта появляется удалённый репозиторий (github, gitlab, bitbacket...).

## 7.1 Clone или fork

Если вы проект начинаете с "нуля" и на локалке, то для создания локального репозитория уже использовали команду qit init.

Однако для разворачивания существующего репозитория на локалке сперва необходимо определиться: fork или clone:

**fork** делает копию проекта на вашем аккаунте, подразумевается что у вас нет доступа к оригинальному репозиторию с которого делаете копию проекта;

**clone** делает копию проекта на локалке, подразумевается что у вас есть доступ к оригинальному репозиторию и в последствии все комиты будут уходить в этот репозиторий.

Компаниям и командам, работающими над проектами, как правило, нет необходимости делать полные копии своих проектов на других аккаунтах, поэтому работы над командным проектом начинается с команды

git clone https://путь к репозиторию.git

## 7 .2 Merge request или pull request

Собственно, это одно и тоже: u pull request и merge request подготавливают изменения к заливке в интеграционную ветку из других веток. Так исторически сложилось, что github использует pull request, a gitlab - merge request.

Рассмотрим процесс слияния веток подробнее.

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 switch feet/4546 form
```

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

#### phpinfo();

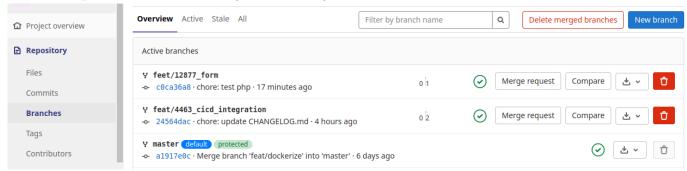
Примечание. Если приступили к задаче и забыли переключиться на нужную веткуне проблемма. Это можно сделать и на текущем этапе. Главное следить за тем, чтобы локально была установлена текущая версия проекта (почаще делаем git pull из ветки master), иначе могут возникнуть конфликты с merge (слиянии веток). К тому же, 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"
```

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

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.

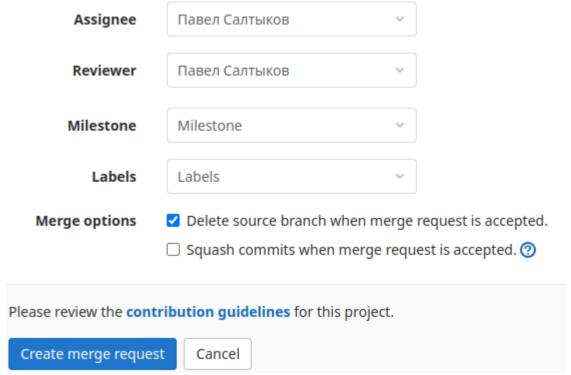
Если всё правильно сделали, увидим следующее:

#### New merge request

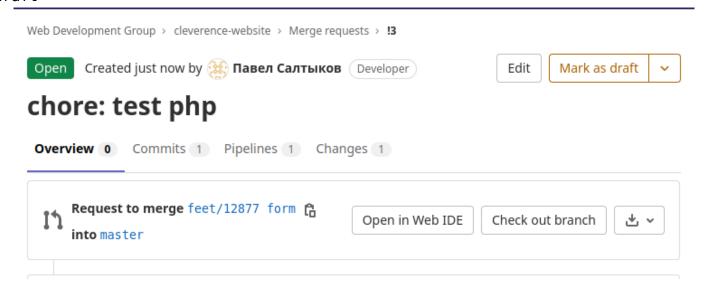
From	feet/12877_	form	into	master	Chang	ge brand	hes
	Ti	cl	chore: test php				
				rt the title	e with	Draft:	to prev

При необходимости, меняем Title.

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