Оглавление

[Типичные задачи. 2](#_Toc152091228)

[Работа с командной строкой 4](#_Toc152091229)

[Настройка git 5](#_Toc152091230)

[Клонирование репозиториев. 9](#_Toc152091231)

[Добавление локальной ветки клона, которая будет отслеживать изменения в оригинальной ветке 9](#_Toc152091232)

[Структура проекта 10](#_Toc152091233)

[Шпаргалки по git 11](#_Toc152091234)

[Про коммиты 12](#_Toc152091235)

[Удаление коммитов 13](#_Toc152091236)

[Правка последнего коммита 15](#_Toc152091237)

[Удаление и переименование файлов 16](#_Toc152091238)

[Ветки 17](#_Toc152091239)

[Слияние веток 18](#_Toc152091240)

[Удаление веток 18](#_Toc152091241)

[История переключений веток: лог ссылок reflog 19](#_Toc152091242)

[Удаление лишних файлов и незакомиченных изменений 19](#_Toc152091243)

[Сравнение коммитов, веток и не только 19](#_Toc152091244)

[Вывод истории gitlog 20](#_Toc152091245)

[Поиск в git log и фильтры 20](#_Toc152091246)

[Слияние веток. 20](#_Toc152091247)

[Слияние перемоткой 21](#_Toc152091248)

[Истинное слияние 22](#_Toc152091249)

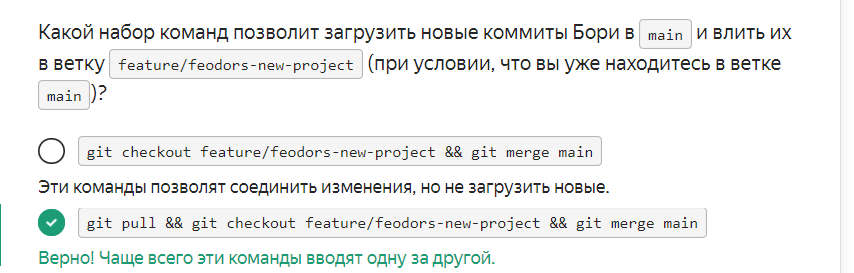
[Коммит слияния. 23](#_Toc152091250)

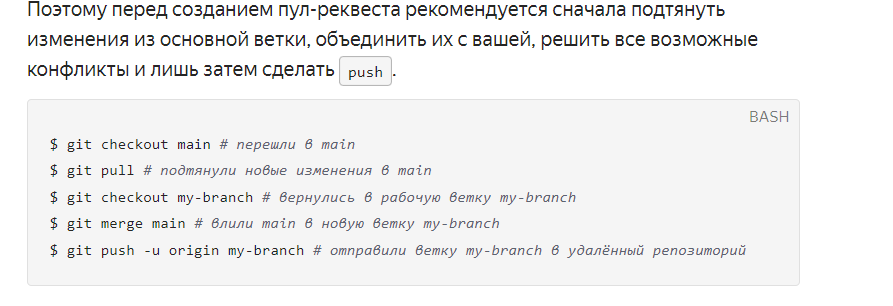
[Отмена слияния. 23](#_Toc152091251)

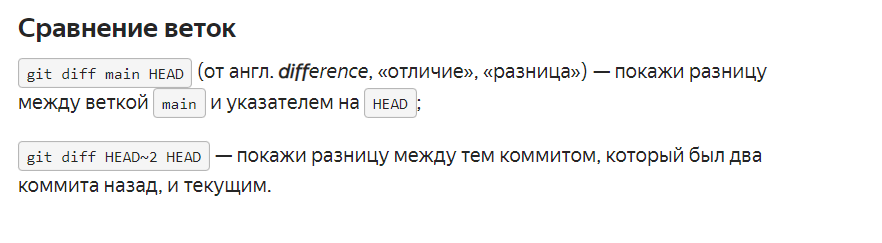
[Семантические конфликты. 23](#_Toc152091252)

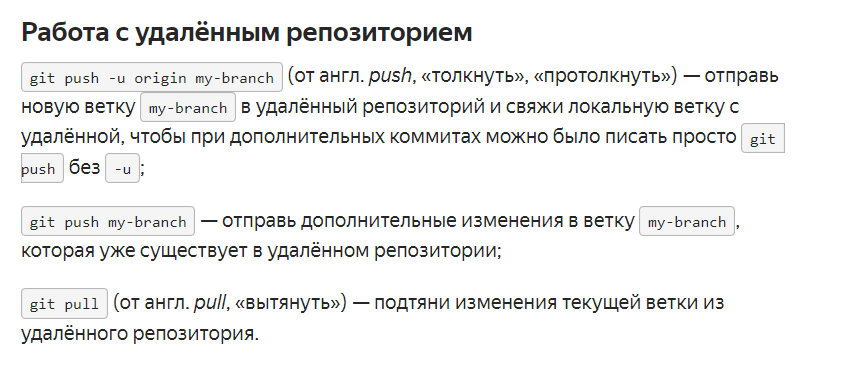
[Копирование коммитов. 24](#_Toc152091253)

Типичные задачи.









Работа с командной строкой

Команды:

~ - перейти к домашней директории

Cd – сменить директорию

Pwd – в какой директории сейчас

Ls – вывести содержимое директории

Touch – создать файл

Mkdir – создать директорию

Структура директорий через флаг -p dir1/dir2/dir3

Cp (что копируем , куда копируем) – для копирования

Mv(что перемещаем, куда перемещаем) – для перемещения файлов

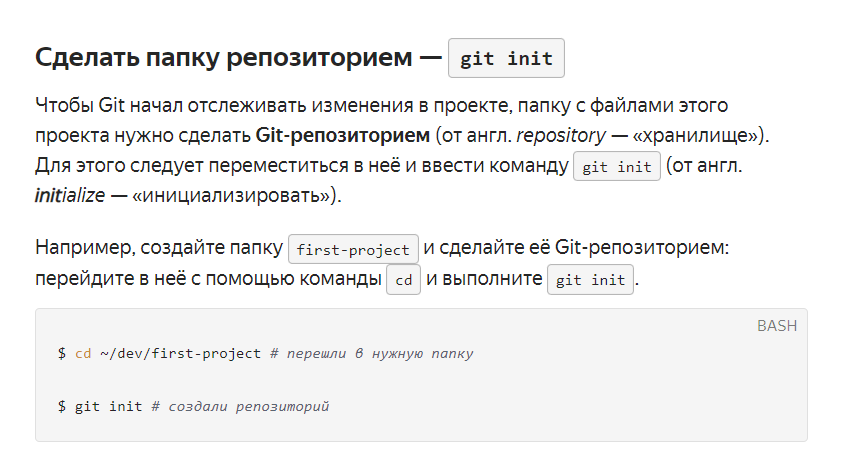
Cat – прочитать txt файл

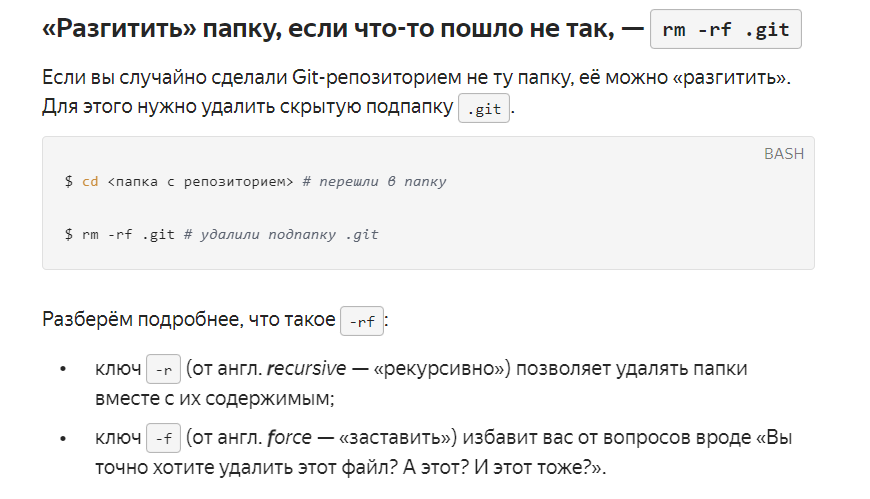
Rm – удалить файл

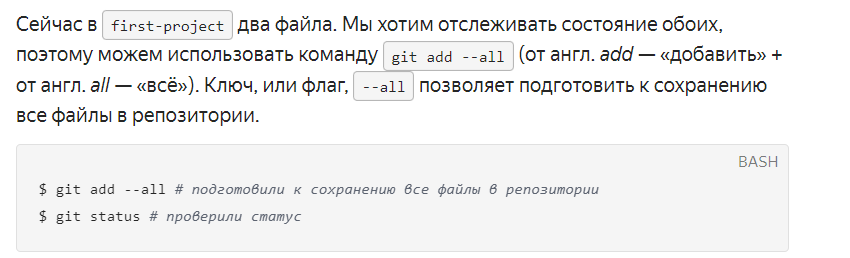
Rmdir – удалить папку (если она пустая)

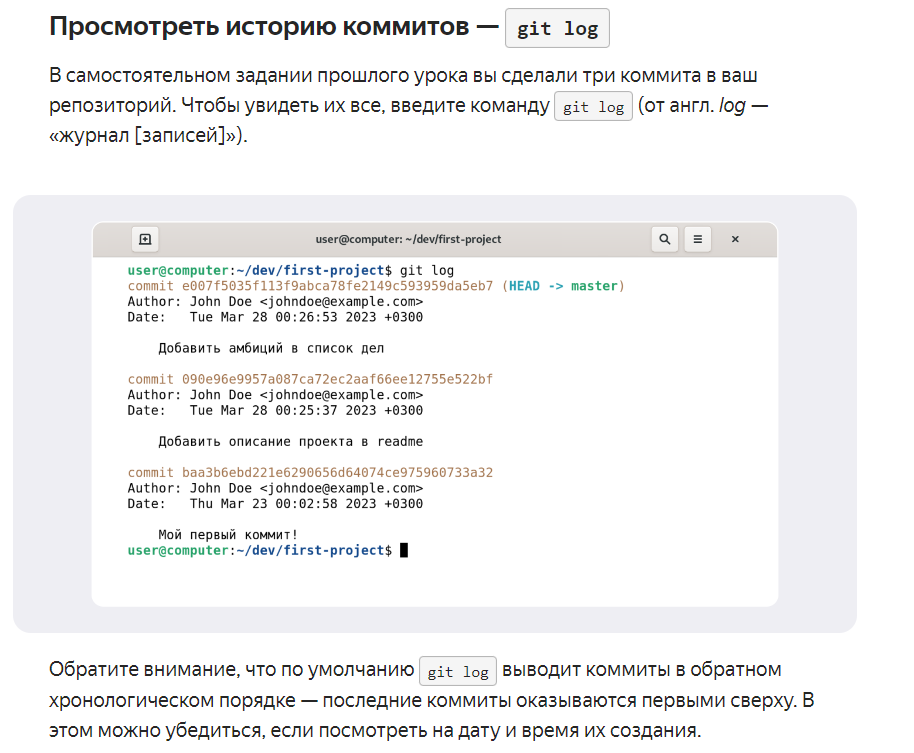
Rm -r – удалить папку вместе с ее содержимым

Настройка git

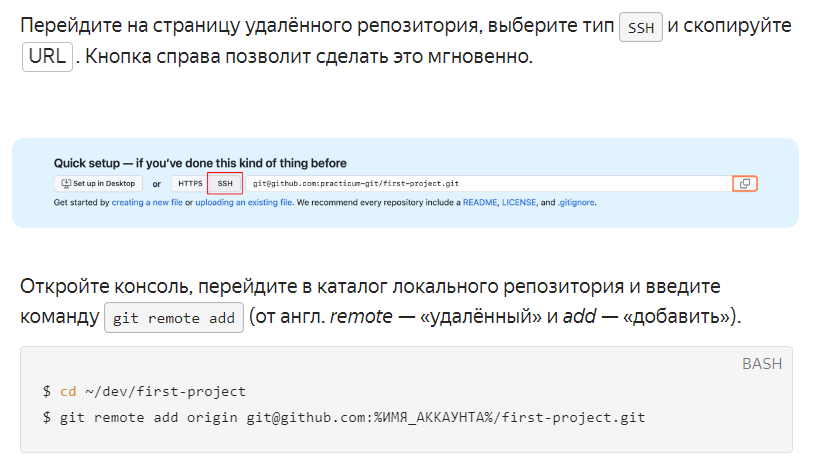


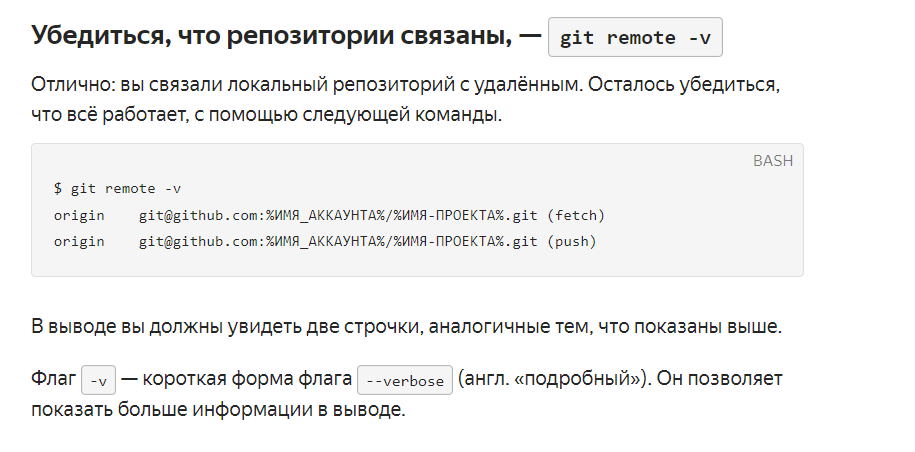


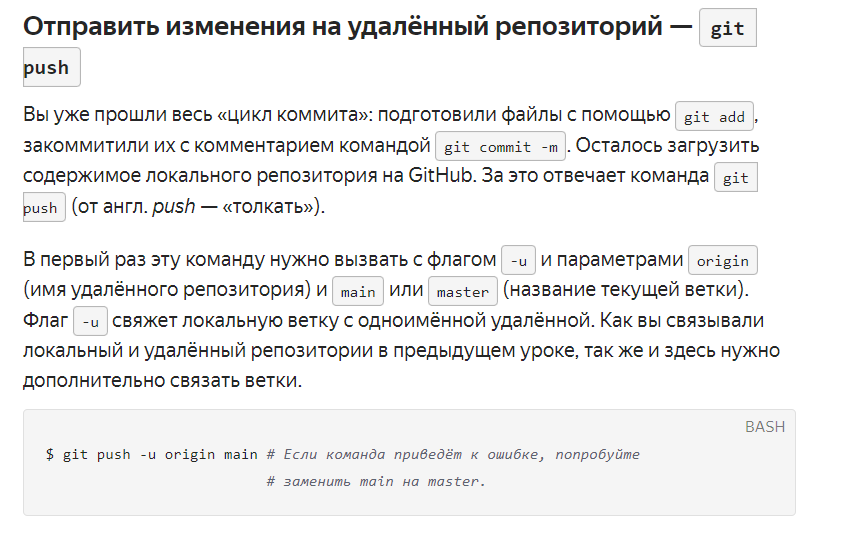




Cвязать локальный и удаленный репозиторий







Клонирование репозиториев.

Чтоб клонировать репозиторий нужно быть в папке , в которой хранится репозиторий(выше него на одну)

Написать **git clone \*имя репоза\* \*имя клона\***

Получить список удаленных веток(от оригинального репозитория)

**Git branch -a**

**Git fetch** + **git merge origin/master** – подтянуть изменения из оригинального репозитория в клон (fetch покажет подтянутые коммиты, но сам клон никак не изменится, и файлы в нем будут старой версии)

**Git pull – аналогично двум верхним командам**

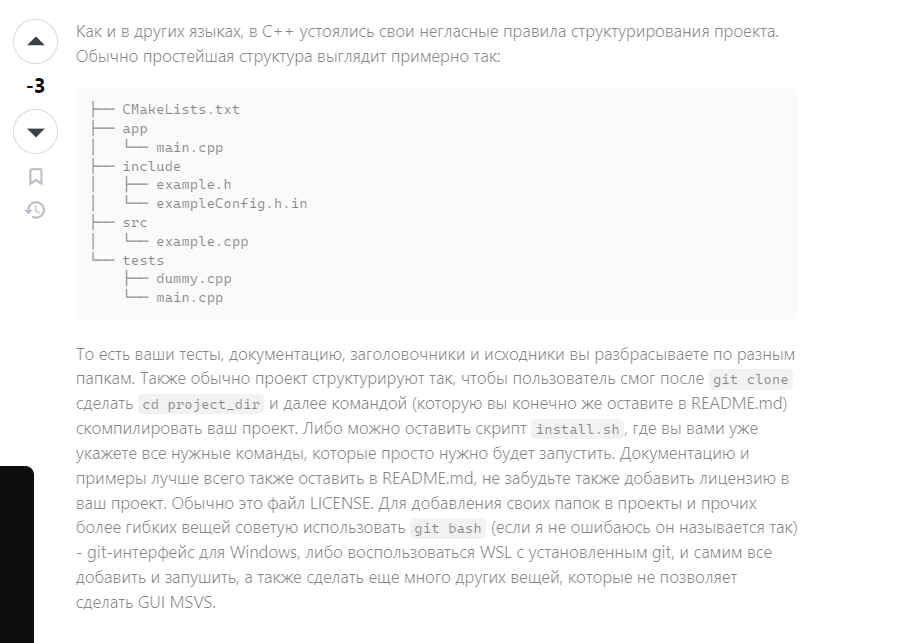
Добавление локальной ветки клона, которая будет отслеживать изменения в оригинальной ветке

**Git branch –track \*название , которое хотим в локальном репозитории\* \*название ветки удаленного репозитория\***

Обычно названия веток удаленного репозитория начинаются с remotes/origin. Пример: **origin/style**

Структура проекта

ОБЯЗАТЕЛЬНО ДОБАВИТЬ .GITIGNORE ЧТОБЫ НЕ УЧИТЫВАТЬ ЛИШНИЕ ФАЙЛЫ



Шпаргалки по git

<https://proglib.io/p/git-cheatsheet>

<https://habr.com/ru/companies/ruvds/articles/599929/#22>

<https://htmlacademy.ru/blog/git/git-basic-commands>

<file:///C:/Users/%D1%84%D0%B2%D1%8C%D1%88%D1%82/Downloads/Telegram%20Desktop/Git%20%D0%B8%20Github%20%D0%B4%D0%BB%D1%8F%20%D0%BD%D0%B0%D1%87%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B8%D1%85.pdf>

<https://www.markdownguide.org/cheat-sheet/>

шпора маркдаун

**git pull** – забрать изменения с github

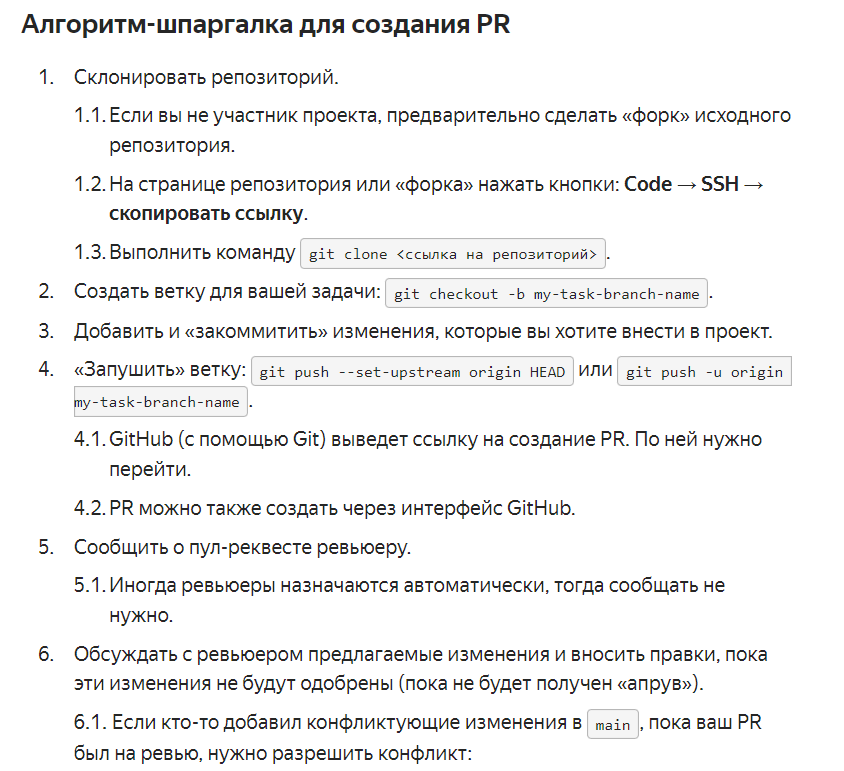
**git checkout «имя файла»** - откатить все изменения в файле до состояния последнего коммита(если они еще не проиндексированы)

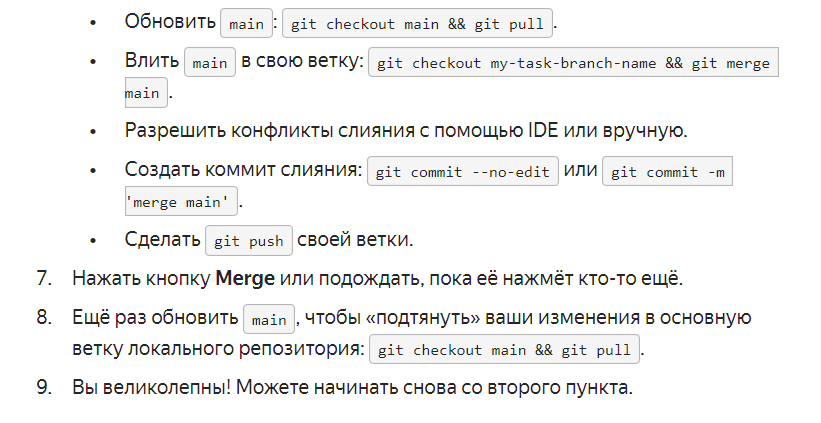
Если же они уже проиндексированы, то до этого написать

Git reset HEAD «имя файла»

Pull-request.

Для добавления изменений в main из своей ветки. Перед добавлением код проходит ревью.



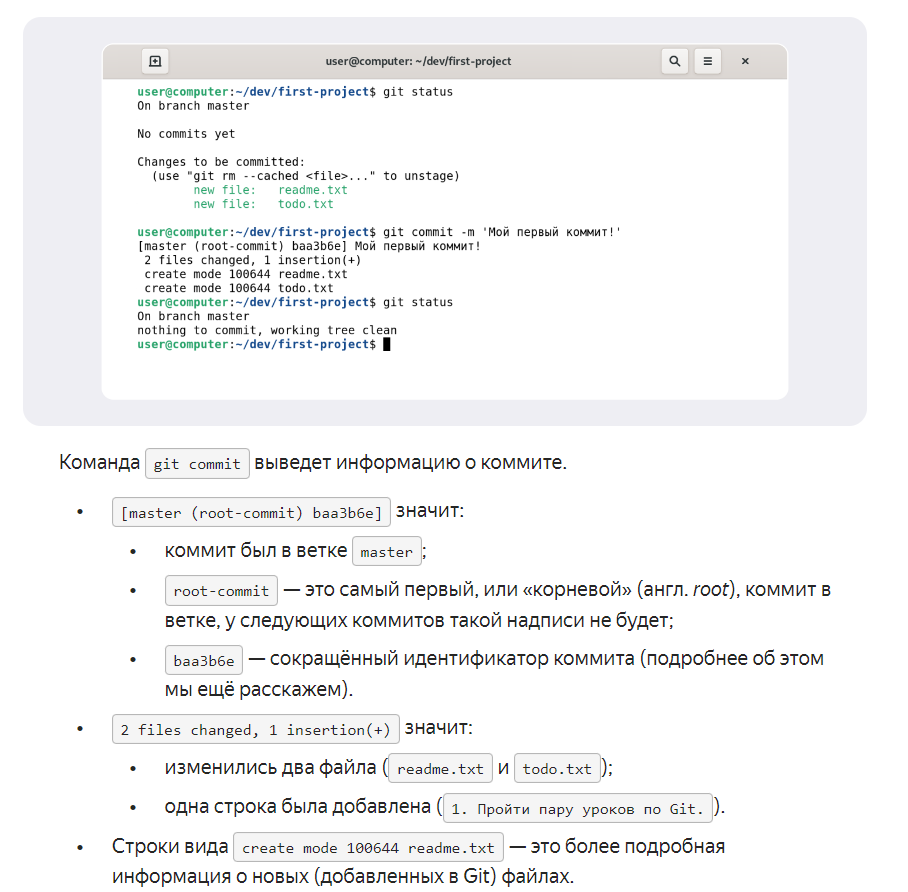


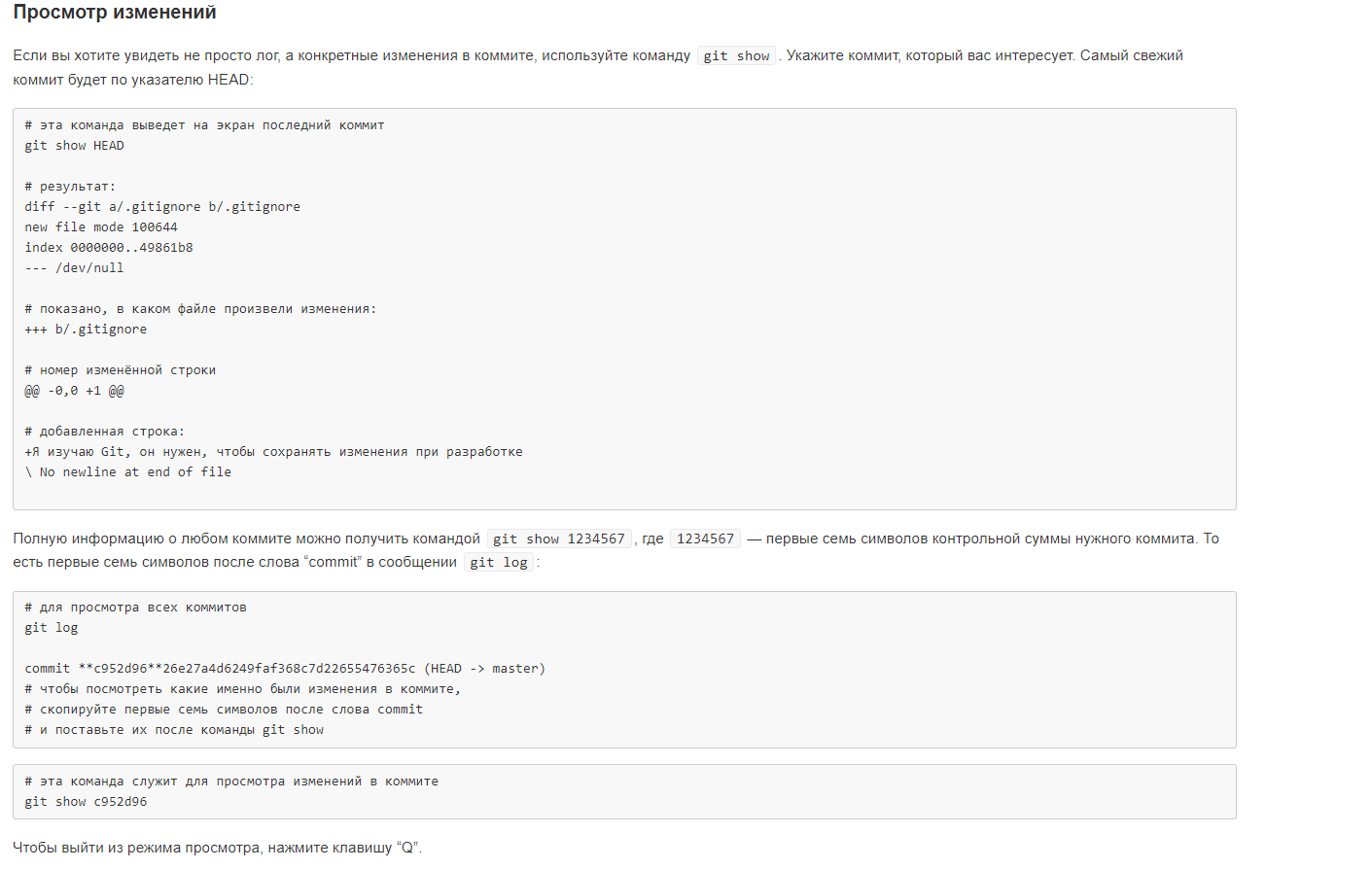
Про коммиты

Первым делом обязательно перед тем как первый раз запушить проект , нужно закоммитить добавленные файлы!!

Git add – не сохраняет состояние файлов в репозиторий!! Фиксацией состояния файлов занимается коммит

Git commit -m – коммит с сообщением





git commit --author='John Smith <john@me.com>'

--date='…' - указать автора изменений

git commit -am “commit” - взять все изменения , сохранить их и закоммитить

(ТОЛЬКО ДЛЯ ОТСЛЕЖИВАЕМЫХ ФАЙЛОВ)

git add -p index.html - добавить определенные изменения из одного файла

Восстановить какой-нибудь файл на момент прошлого коммита git checkout \*хеш на коммит\* \*имя файла\*

Посмотреть что было в определенном файле в момент прошлых коммитов git show \*хеш\* : \*имя файла\* , выведет в консоль файл полностью

Удаление коммитов

**Git revert HEAD** – отменит последний коммит , путем создания нового коммита и изменений в директории.

**Git reset –hard \*commit\_id\*** - commit\_id идентификатор коммита к которому нужно вернуться (обновляет директорию вместе с индексом)

Т.е. старые коммиты больше не участвуют в процессе разработки и потом будут удалены из базы

После этого делаем **git push –force** для сохранения изменений на гитхабе

**Git reset \*commit\_id\*** - откатить изменения вплоть до коммита commit\_id

**Ещё одно применение!** Че то поделали , добавили в add, но еще не закоммитили и передумали вообще это коммитить

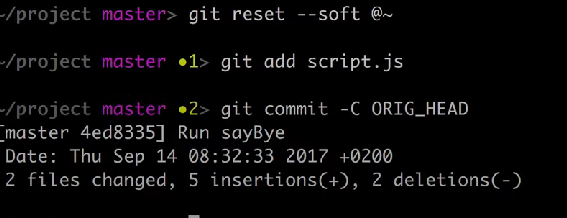
Пишем **git reset –hard** и все откатывается

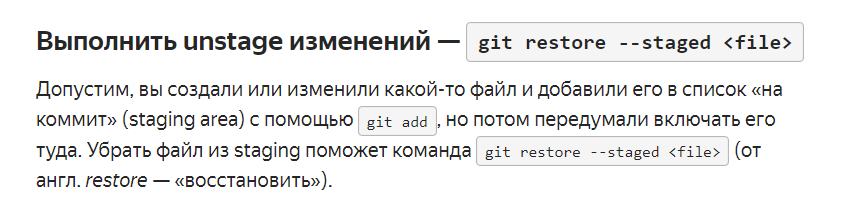
**Git reset \*хеш\*** - делает то же что и –hard , но не трогает директорию

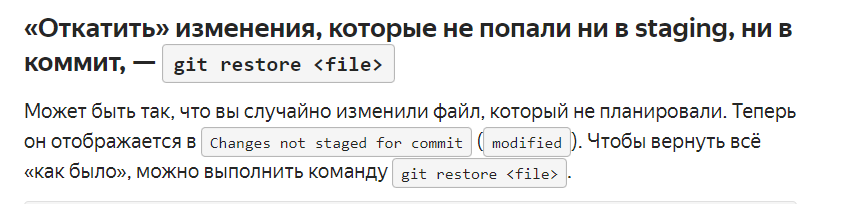
**Еще одно применение!** Git reset HEAD – отменить изменения добавленные в индекс

**Git reset 54a4 index.html** – откатить индекс для файла на состояние прошлых коммитов

Git reset –soft – переносит ветку , но индекс и директорию не трогает! Используется когда забыли что-то доделать в прошлом коммите, удаляем его, но сам код не изменяется, доделываем его и коммитим по новой!



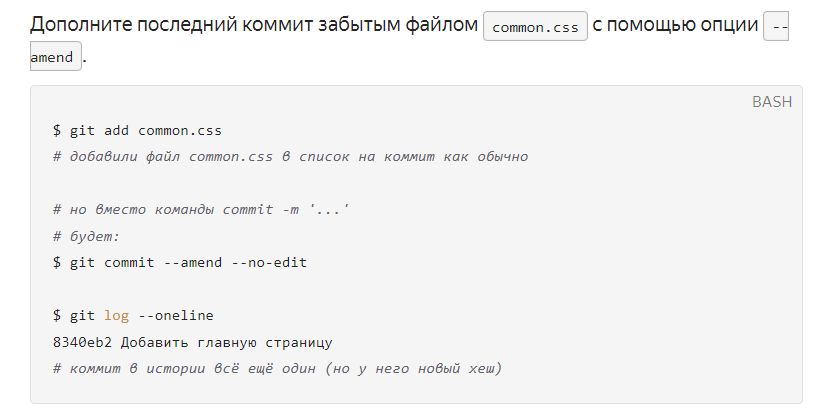




Он откатит изменения в файле

Правка последнего коммита

**Git commit --amend** , делает два в одном , когда нужно в коммит добавить еще что-то и перезаписать его описание.



Удаление и переименование файлов

git rm -r - удаление с индексированием

git rm -f - принудительное удаление файлов с несохраненными изменениями

git rm -r —cached - удаление с индексированием , но файл остается в каталоге (для служебных файлов добавленных по ошибке)

git mv \*что переименовать\* \*во что переименовать\* - переименование

Ветки

git branch -v - информация о ветке, и на что она указывает(head) т.е. последний коммит в ветке

git checkout - перейти к определенной ветке , при этом в директории останутся файлы относящиеся только к ней

Чекаут не работает если в ветке в которой мы были до этого есть не сохраненные изменения , поэтому:

git stash - перекинуть не сохраненные изменения в стек

git stash pop - достать из стека

git checkout -b ветка - создать ветку и перейти в нее

Чтобы сохранить незакомиченные изменения в новой ветке нужно сначала создать ее и переключиться , а потом коммитить

Чтобы передвинуть указатель на последний коммит ветки git branch -f master \*хеш куда переместить\*

git checkout -B \*что переместить\* \*хеш куда переместить\* - переместить ветку и переключиться на нее

Восстановить какой-нибудь файл на момент прошлого коммита git checkout \*хеш на коммит\* \*имя файлк\*

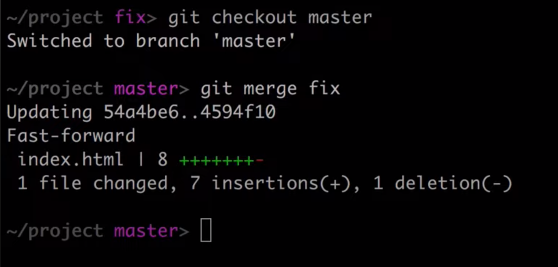
Посмотреть что было в определенном файле в момент прошлых коммитов git show \*хеш\* : \*имя файла\* , выведет в консоль файл полностью

Слияние веток

Например: исправляли баг в отдельной ветке, теперь нужно совместить эту ветку с той, для которой исправляли баг

Сначала делаем чекаут на ту ветку, в которую хотим добавить другую

Git merge \*ветка откуда добавить изменения\*



Если после слияния мы передумали!

Merge сохраняет старую голову ветки в переменной ORIG\_HEAD

Поэтому используем git branch -f master ORIG\_HEAD

Удаление веток

Git branch -d \*название ветки\* (удаляет только ссылку на ветку, не сами коммиты)

**Если хотим снести ветку со всеми коммитами** – то не делаем слияние с другими ветками , уходим с этой ветки и пишем

Git branch -D \*название ветки\*

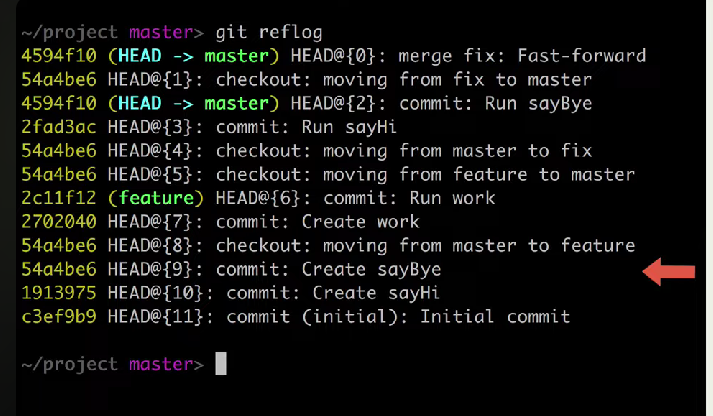
Если хотим вернуть ветку, которую удалили с помощью прошлой команды

Git branch \*то же название ветки\* \*хеш последнего коммита\*

История переключений веток: лог ссылок reflog

Git reflog \*ссылка на ветку\*

Git reflog – выведет логи для HEAD



Git branch feature HEAD@{6} – возьмет хеш из reflogа

Удаление лишних файлов и незакомиченных изменений

Git clean -dxf – удаляет все неотслеживаемые файлы

Git checkout -f HEAD – откатит все изменения и индекс до последнего HEAD (кроме неотслеживаемых файлов)

Git reset –hard – откатит все изменения и индекс до последнего HEAD

Сравнение коммитов, веток и не только

Git diff \*хеш 1 коммита\* \*хеш 2 коммита\*

Git diff \*имя 1 ветки\*…\*имя 2 ветки\* - показывает что изменилось во второй ветке после выхода из первой

Git diff HEAD – покажет изменения с момента последнего коммита

Git diff –cached – покажет изменения которые уже проиндексированы, но еще не в репозитории

Вывод истории gitlog

Git log выводит историю коммитов

Git log oneline – выводит коммиты в формате хеш + описание

git log master – вывести историю коммитов для ветки master

git log master feature –graph – вывести историю коммитов для веток master и feature

Поиск в git log и фильтры

**Git log –grep «слово»** - выведет коммиты в названии которых есть определенное слово

**Git log -G«строка» -p** - выведет коммиты в изменениях которых есть определенная строка (именно в коде)

**Git log –author = «имя»** - ищет коммиты автора

**Git log –before ‘2017-09-13’** – коммиты до определенной даты (или after)

**Git blame \*имя файла\*** - посмотреть кто че в файле менял

Слияние веток.

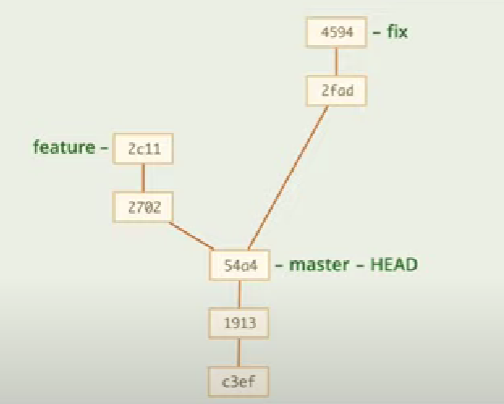
Слияние перемоткой(fast-forward)

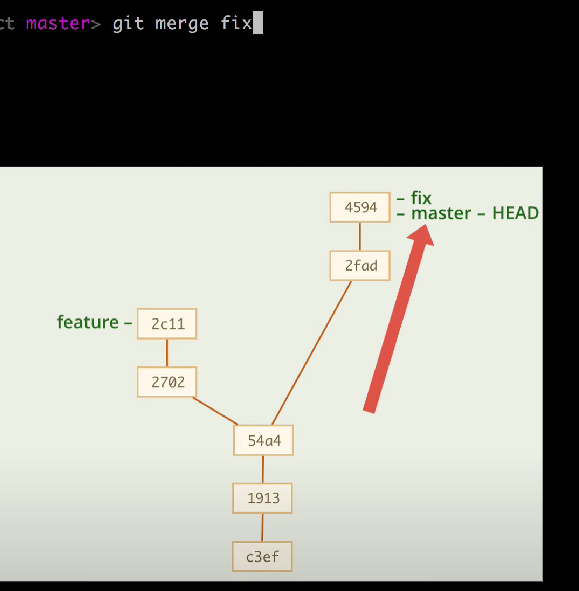
НЕ ОЧЕНЬ ПРИВЕТСТВУЕТСЯ! ЛУЧШЕ ОТКЛЮЧИТЬ

**git merge --no-edit --no-ff**

ЛИБО

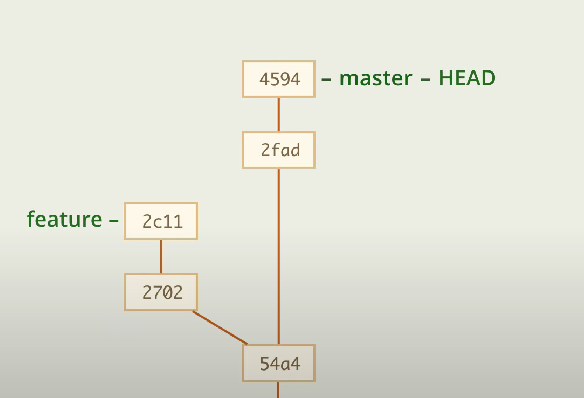
**git config [--global] merge.ff false** – для всех проектов



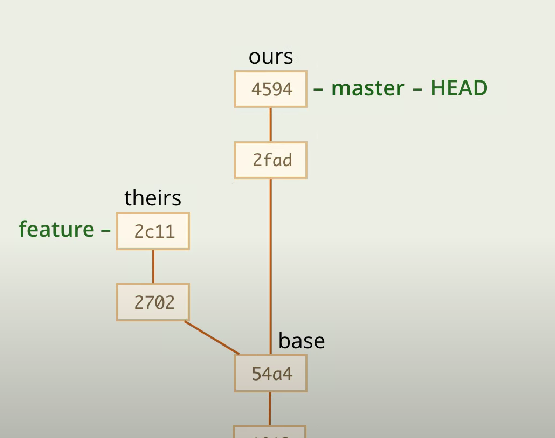


Истинное слияние

Анализирует все изменения из веток, объединяет их и создает новый коммит со слиянием.



Если написать git merge , то гит будет искать изменения между тремя версиями:



Base – общий предок

Ours – ветка, на которой сейчас находимся

Theirs – ветка, изменения из которой нужно подтянуть

Во время слияния могут возникнуть **конфликты**, если мы в разных ветках делали изменения в одних и тех же файлах в одних и тех же местах.

Мы можем либо отменить слияние через **git merge –abort**

Либо довести слияние до конца и исправить файл самостоятельно и добавить эти изменения в индекс и написать **git merge –continue**

**Git merge –no-ff \*ветка\***– отменить слияние перемоткой , использовать только истинное слияние

Git merge –squash – изменения внесены, но нет связи с веткой откуда они были взяты (чтобы в истории главной ветки не было всякой херни из другой ветки)

Коммит слияния.

У коммита слияния есть особенность – у него два родителя от двух веток, которые сливались.

Git diff HEAD^2 – отличия от второго родителя

Git diff HEAD^1 – отличия от первого родителя

Git log master –oneline –first-parent – посмотреть историю коммитов только у определенной ветки(без коммитов от веток, с которыми сливали)

Отмена слияния.

Git reset –hard @~ - сделали шаг назад и отменили коммит слияния и слияние.

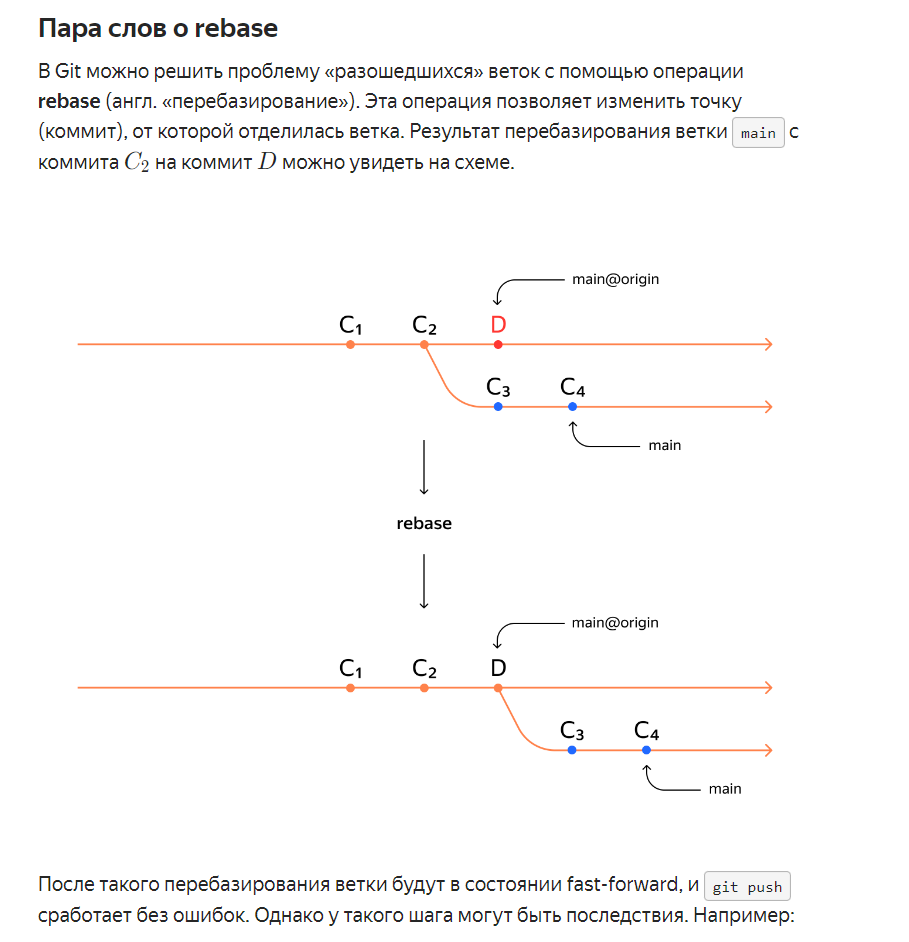
Семантические конфликты.

Конфликты по смыслу, которые гит не может обнаружить, но это ломает проект! Т.к. мы не делаем коммиты , которые ломают проект, то нужно это исправить и не делать коммит сразу.

Git merge work –no-commit – стопается перед коммитом.

Вносим исправления, и только тогда делаем коммит.

Rebase



Копирование коммитов.

Когда в нескольких ветках нужно было сделать одни и те же изменения и копировать коммит.

Git cherry-pick D

\*доработать\*