ML Basic

Начнем в 20:01 [мск]

Git, shell

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

ML Basic

Начнем в 20:01 [мск]

Git, shell

otus.ru

ML Basic

Начнем в 20:01 [мск]

Git, shell

otus.ru

Тема вебинара

ML Basic

Катя

Екатерина Дмитриева

Telegram: @dmi3eva

Маршрут вебинара

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в учебной группе

Задаем вопрос
в чат

Вопросы вижу в чате,
могу ответить не сразу

Доброжелательный стиль общения
между всеми участниками вебинара

Условные обозначения

Индивидуально

Время, необходимое
на активность

Пишем в чат

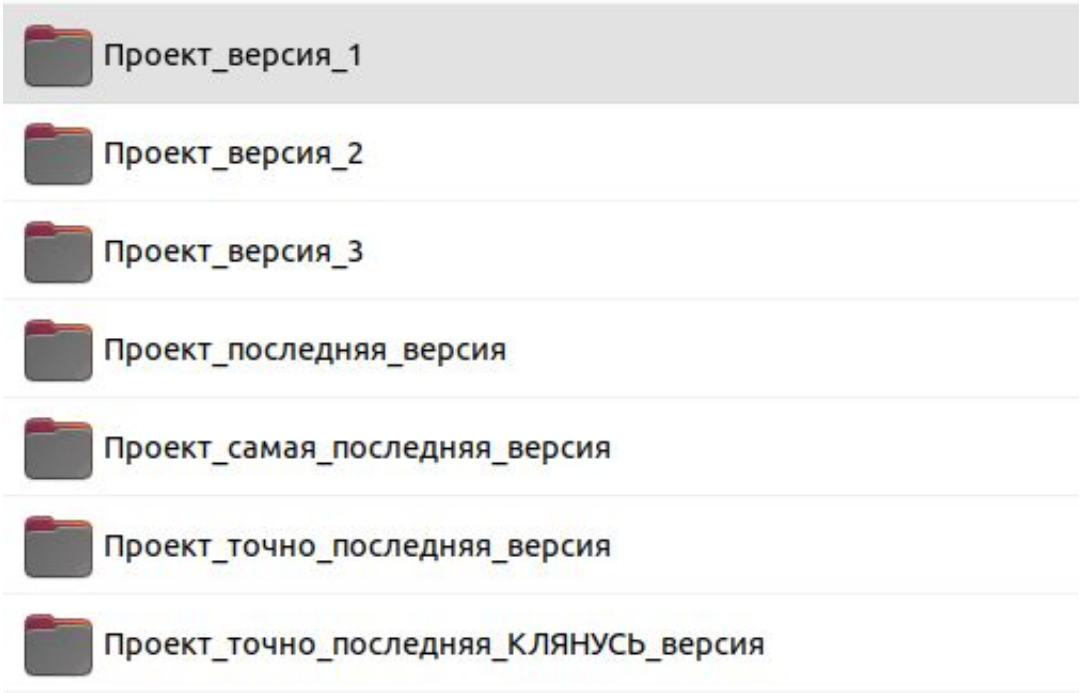
Говорим голосом

Документ

Ответьте себе или
задайте вопрос

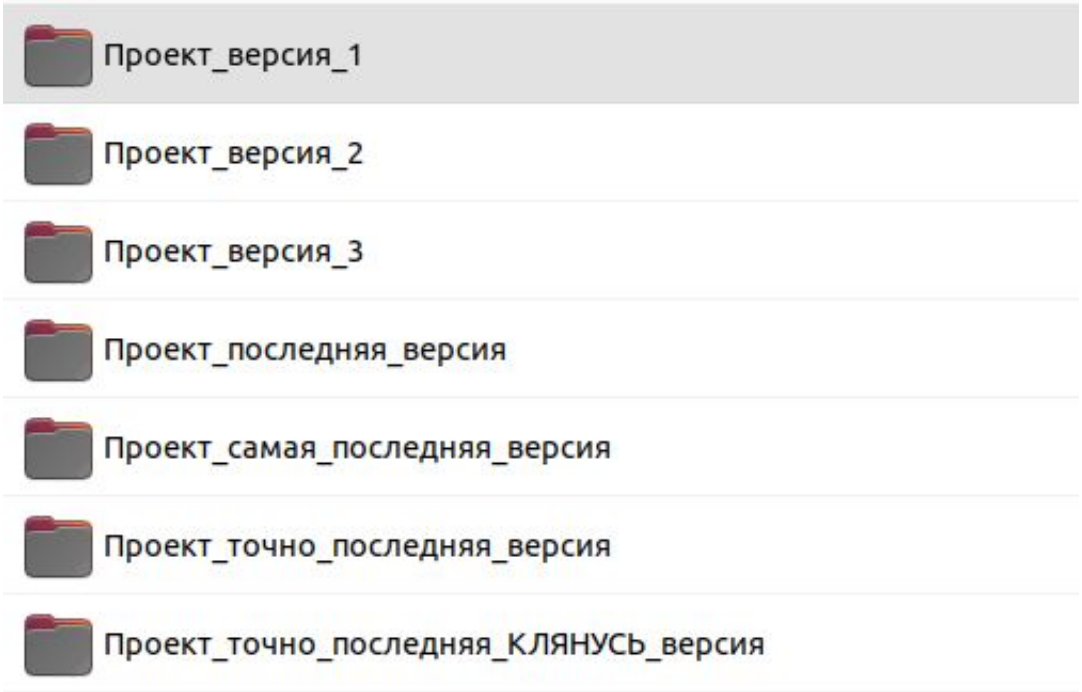
Система контроля версий

- У вас так было?



Система контроля версий

- У вас так было?
- Система контроля версий позволяет экономить место и поддерживать порядок



Система контроля версий

Нужен для:

1. Сохранения версий проекта
   Чтобы ничего не испортить новыми экспериментами и всегда можно было “откатиться” назад

Система контроля версий

Нужен для:

1. Сохранения версий проекта
   Чтобы ничего не испортить новыми экспериментами и всегда можно было “откатиться” назад
2. Для удобства командной работы над одним проектом

Система контроля версий

Нужен для:

1. Сохранения версий проекта
2. Для удобства командной работы над одним проектом
3. Наличия стимула писать код структурно и думать о процессе

Что версионировать датасаентисту?

1. Код [обсудим сейчас]
2. Данные [ключевые слова: DVC]
3. Результаты экспериментов [ключевые слова: NeptuneAI]
4. Версии моделей [ключевые слова: Huggingface, чекпоинты]

Система контроля версий

Система контроля версий

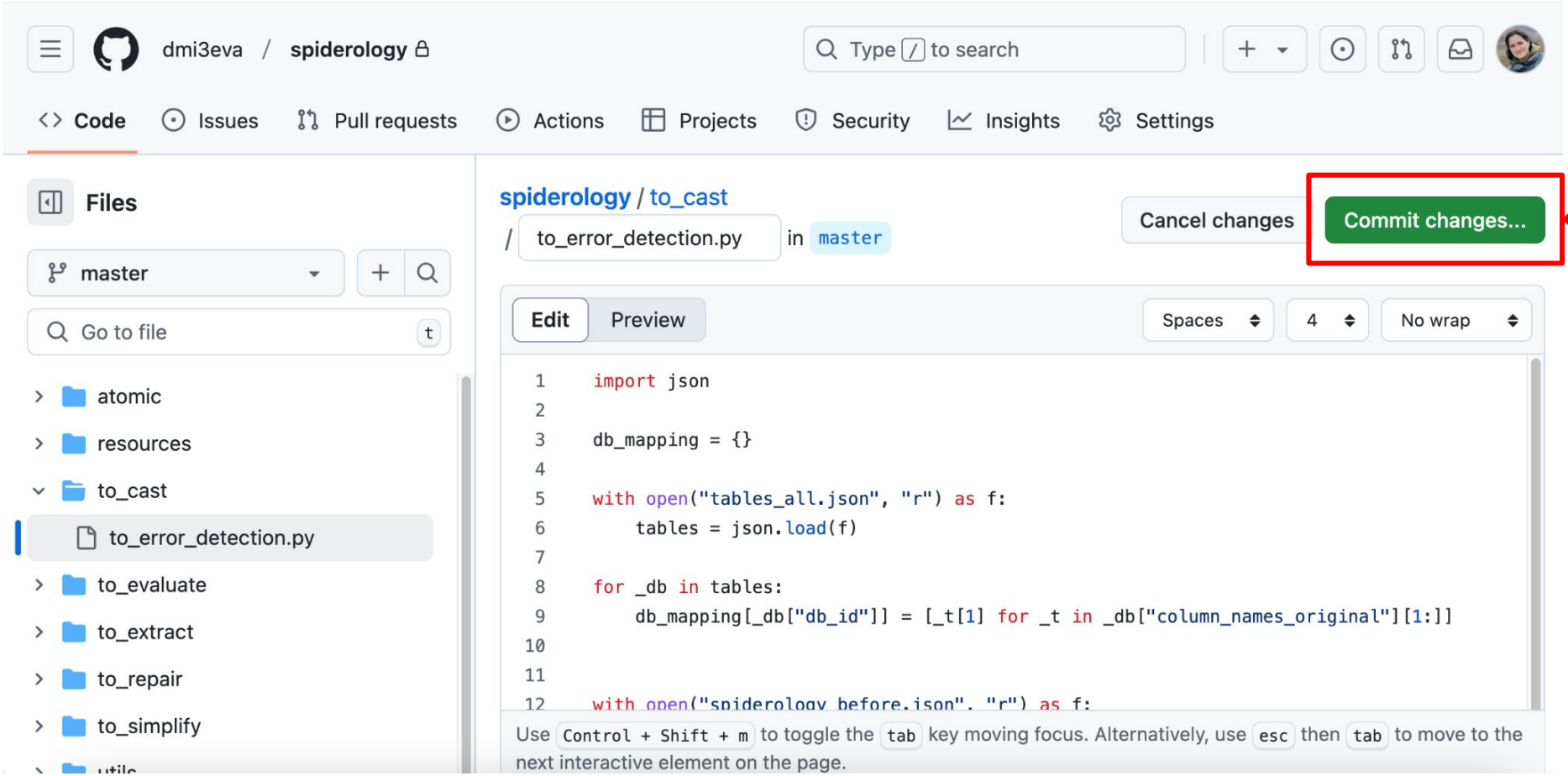
- **Система контроля версий** = System of versions control = SVC
- Инструмент, отдельный от инструмента создания проекта (чаще всего представляет из себя отдельный язык)
- Есть много разных (Git, SVN, Mercurial) - у каждого свой принцип работы, свой “язык”, свои среды
- Мы будем работать с системой контроля версий **Git.**

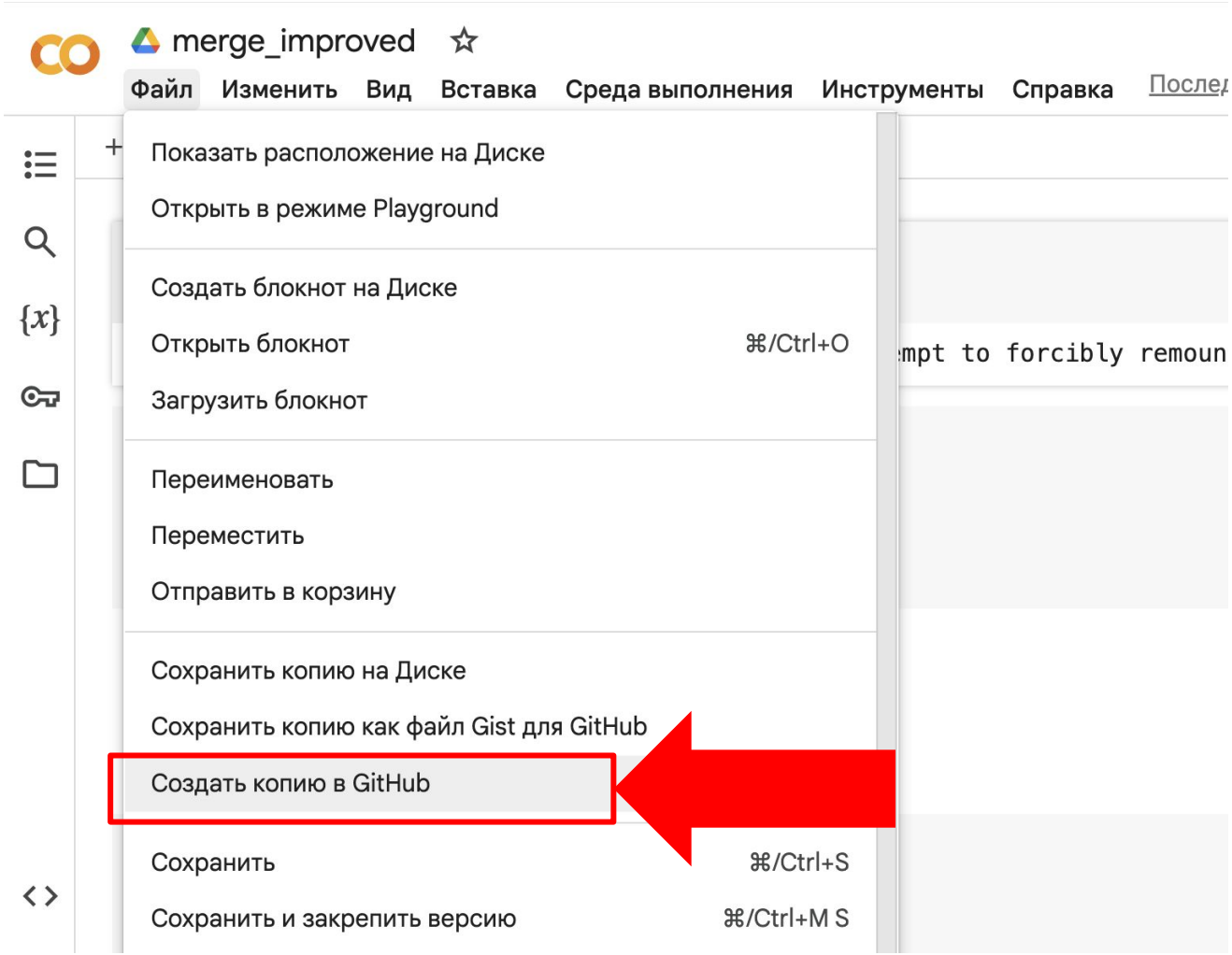
Система контроля версий

1. Специальное локальное ПО
2. Глобальный репозиторий

Commit message style

- https://www.conventionalcommits.org/en/v1.0.0/


dmi3eva / spiderology
Type / to search
Code
Issues
Pull requests
Actions
Projects
Security
Insights
Settings
Files
master
Go to file t
atomic
resources
to_cast
to_error_detection.py
to_evaluate
to_extract
to_repair
to_simplify
spiderology / to_cast
/ to_error_detection.py in master
Cancel changes
Commit changes...
Edit
Preview
Spaces
4
No wrap
1 import json
2
3 db_mapping = {}
4
5 with open("tables_all.json", "r") as f:
6 tables = json.load(f)
7
8 for _db in tables:
9 db_mapping[_db["db_id"]] = [_t[1] for _t in _db["column_names_original"][1:]]
10
11
12 with open("spiderology_before.json", "r") as f:
Use Control + Shift + m to toggle the tab key moving focus. Alternatively, use esc then tab to move to the next interactive element on the page.


merge_improved
Файл
Изменить
Вид
Вставка
Среда выполнения
Инструменты
Справка
Показать расположение на Диске
Открыть в режиме Playground
Создать блокнот на Диске
Открыть блокнот
⌘/Ctrl+O
Загрузить блокнот
Переименовать
Переместить
Отправить в корзину
Сохранить копию на Диске
Сохранить копию как файл Gist для GitHub
Создать копию в GitHub
Сохранить
⌘/Ctrl+S
Сохранить и закрепить версию
⌘/Ctrl+M S
mpt to forcibly remoun

Основные понятия

Система контроля версий

- **Вариант 0:** Вручную :)

Система контроля версий

- **Вариант 1:** С помощью встроенного в PyCharm плагина. Поэтому нам придется учить отдельный язык и устанавливать отдельную среду. Все будем делать “кнопками”, обходясь без написания команд вручную.

Система контроля версий

- **Вариант 1:** С помощью встроенного в PyCharm плагина.
  - Не надо учить отдельный язык и устанавливать отдельную среду.
  - Все будем делать “кнопками”, обходясь без написания команд вручную.

Система контроля версий

- **Вариант 1:** С помощью встроенного в PyCharm плагина.
  - Не надо учить отдельный язык и устанавливать отдельную среду.
  - Все будем делать “кнопками”, обходясь без написания команд вручную.
- **Вариант 2:** Через командную оболочку
  - Более универсально

Основные понятия

- Коммит
- Репозиторий
- Ветка

Задача 1

N хоббитов делят K кусков эльфийского хлеба поровну, не делящийся нацело остаток остается в корзинке у Сэма. Напишите функцию, которая принимает на вход параметры N и K и возвращает два числа: x - сколько кусков эльфиского хлеба достанется каждому хоббиту, и y - сколько кусков остаётся в корзинке.

Задача 1

N хоббитов делят K кусков эльфийского хлеба поровну, не делящийся нацело остаток остается в корзинке у Сэма. Напишите функцию, которая принимает на вход параметры N и K и возвращает два числа: x - сколько кусков эльфиского хлеба достанется каждому хоббиту, и y - сколько кусков остаётся в корзинке.

```
def share_bread(N, K):
    return N + K, K
```

Задача 1

N хоббитов делят K кусков эльфийского хлеба поровну, не делящийся нацело остаток остается в корзинке у Сэма. Напишите функцию, которая принимает на вход параметры N и K и возвращает два числа: x - сколько кусков эльфиского хлеба достанется каждому хоббиту, и y - сколько кусков остаётся в корзинке.

```python
def share_bread(N, K):
    return N + K, K


assert share_bread(N=3, K=14) == (4, 2)
```

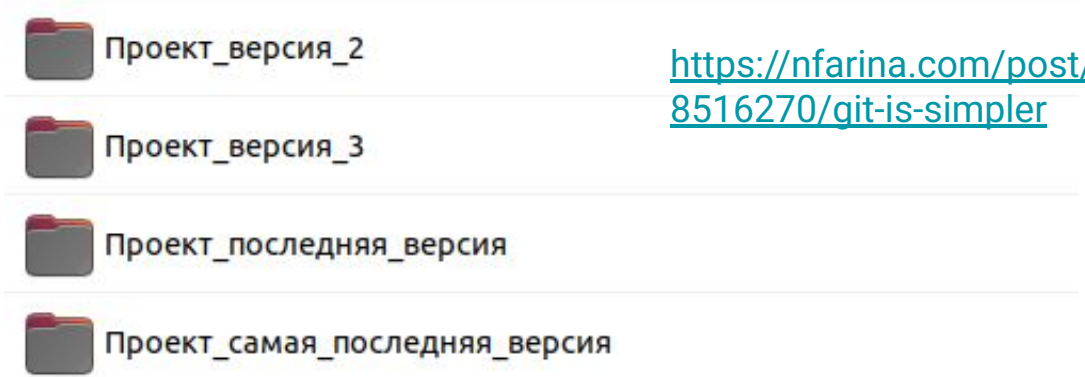
Основные понятия

- **Коммит** – набор сохраненных изменений

Основные понятия

- **Коммит** – набор сохраненных **изменений**

vs. Версия



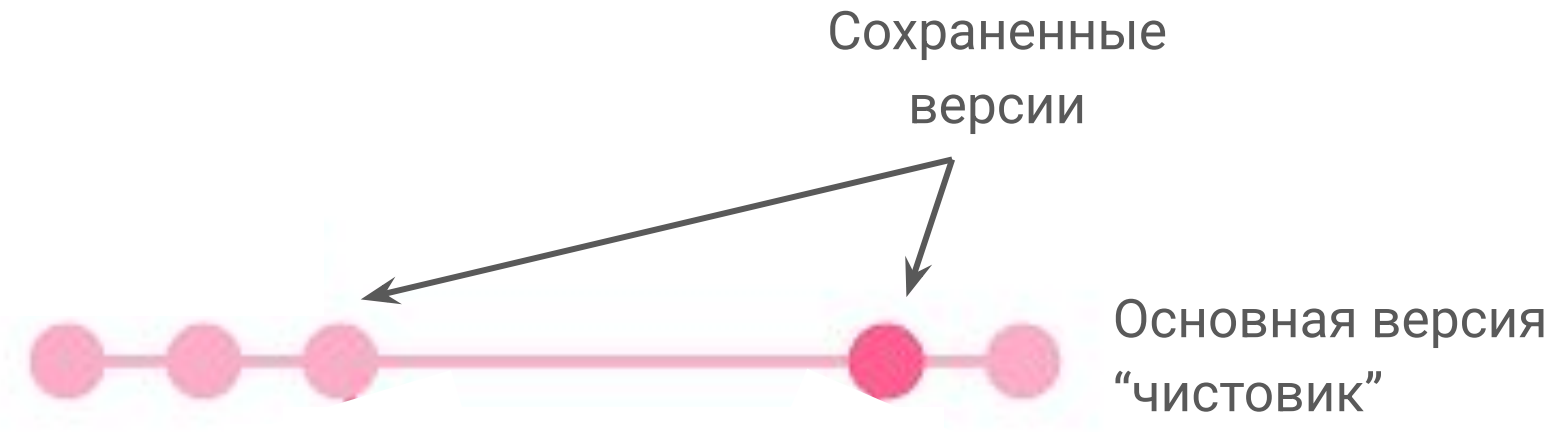
https://nfarina.com/post/9868516270/git-is-simpler

Основные понятия

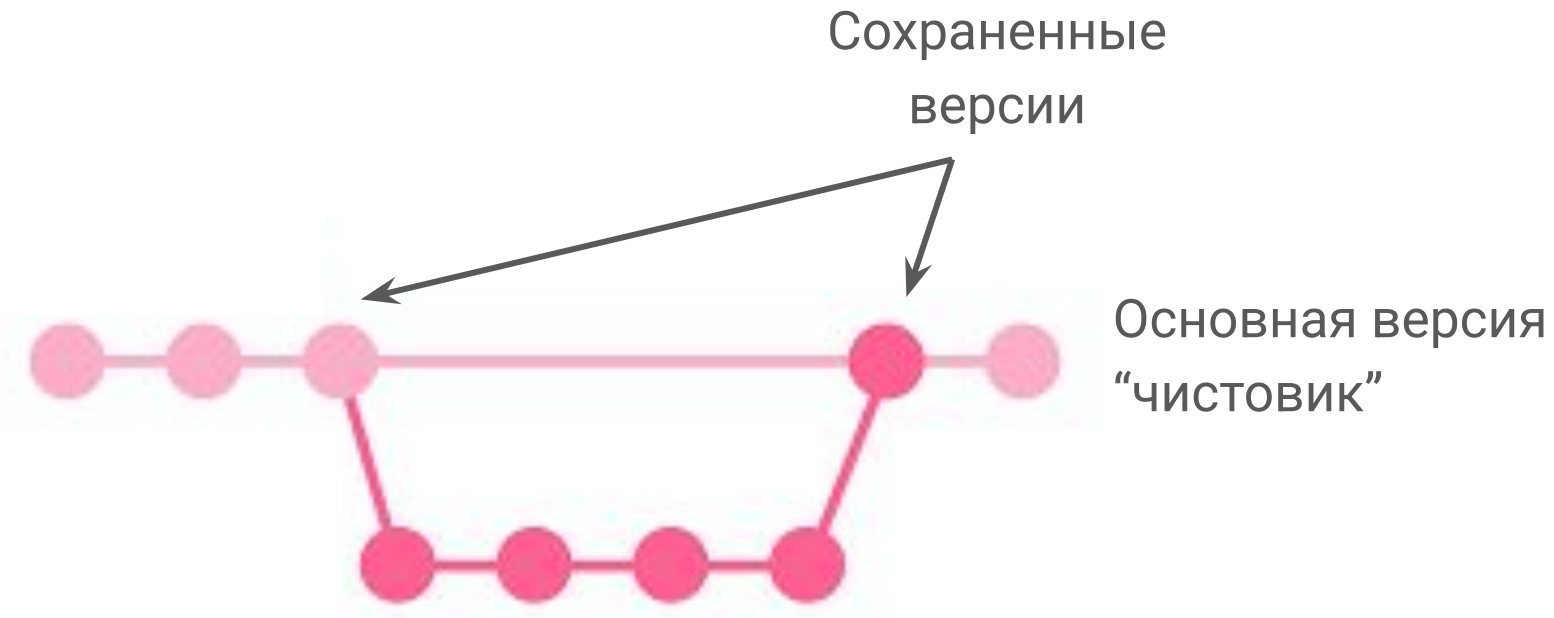
- **Репозиторий** – это хранилище, в котором расположен ваш проект и его история.

Git vs. GitHub

Основной принцип работы

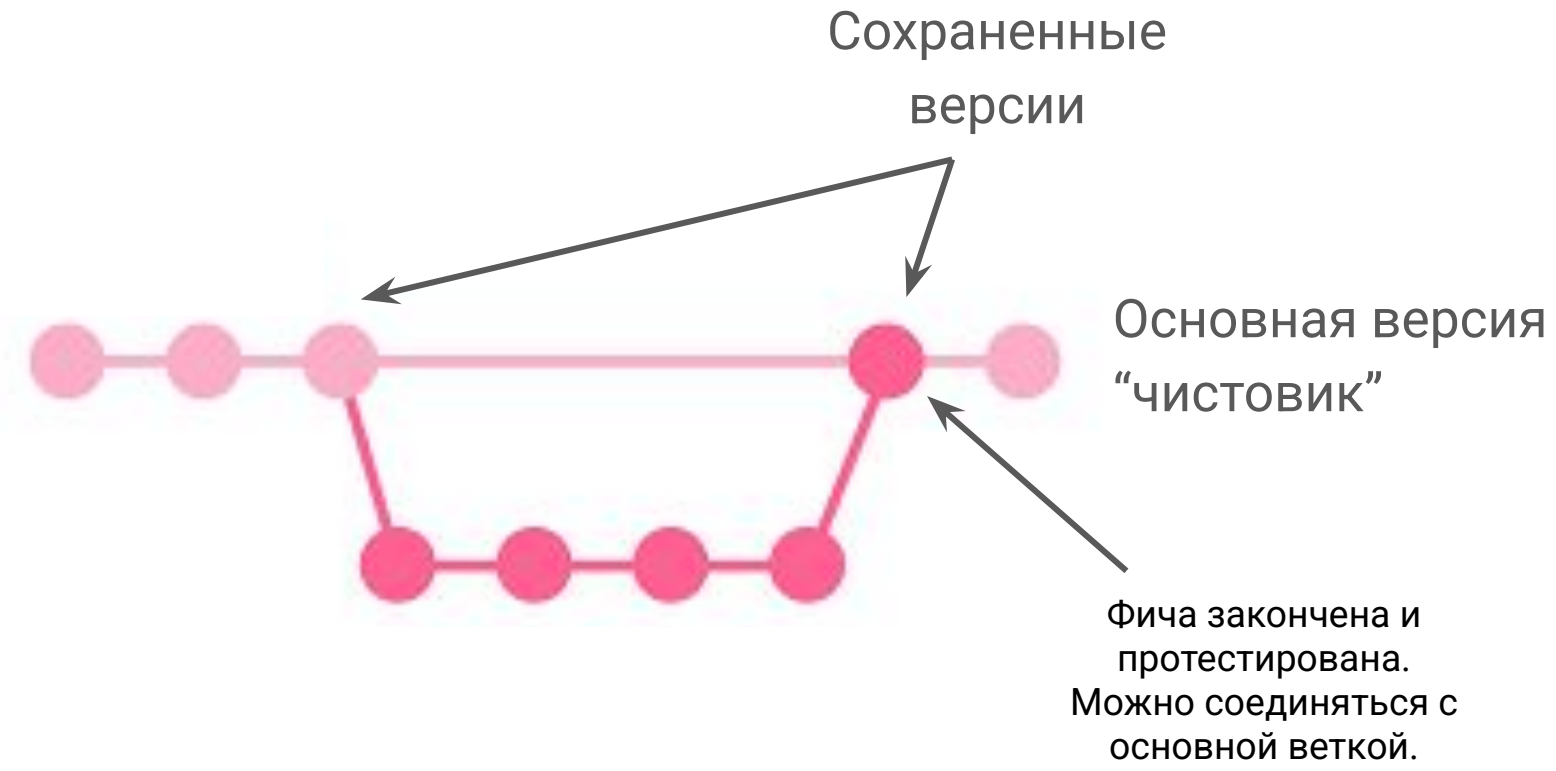


Основной принцип работы

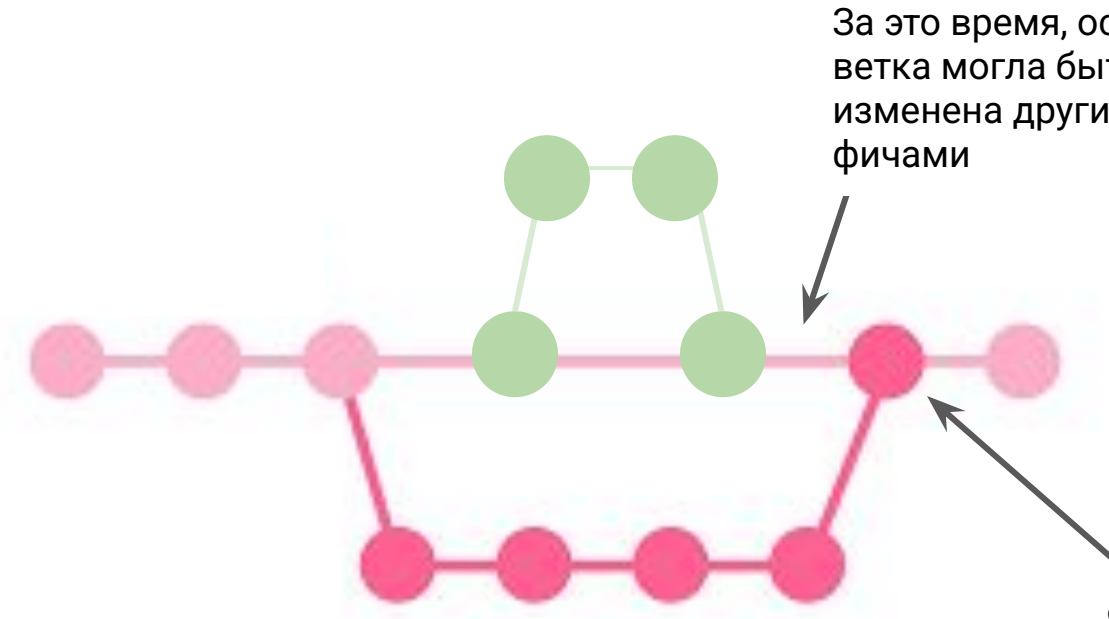


Временная ветка для создания новых “фич”. Над ней может трудиться отдельный член команды. Отдельная ветка нужна, чтобы до момента готовности фичи не “сорить” в чистовик

Основной принцип работы



Основной принцип работы

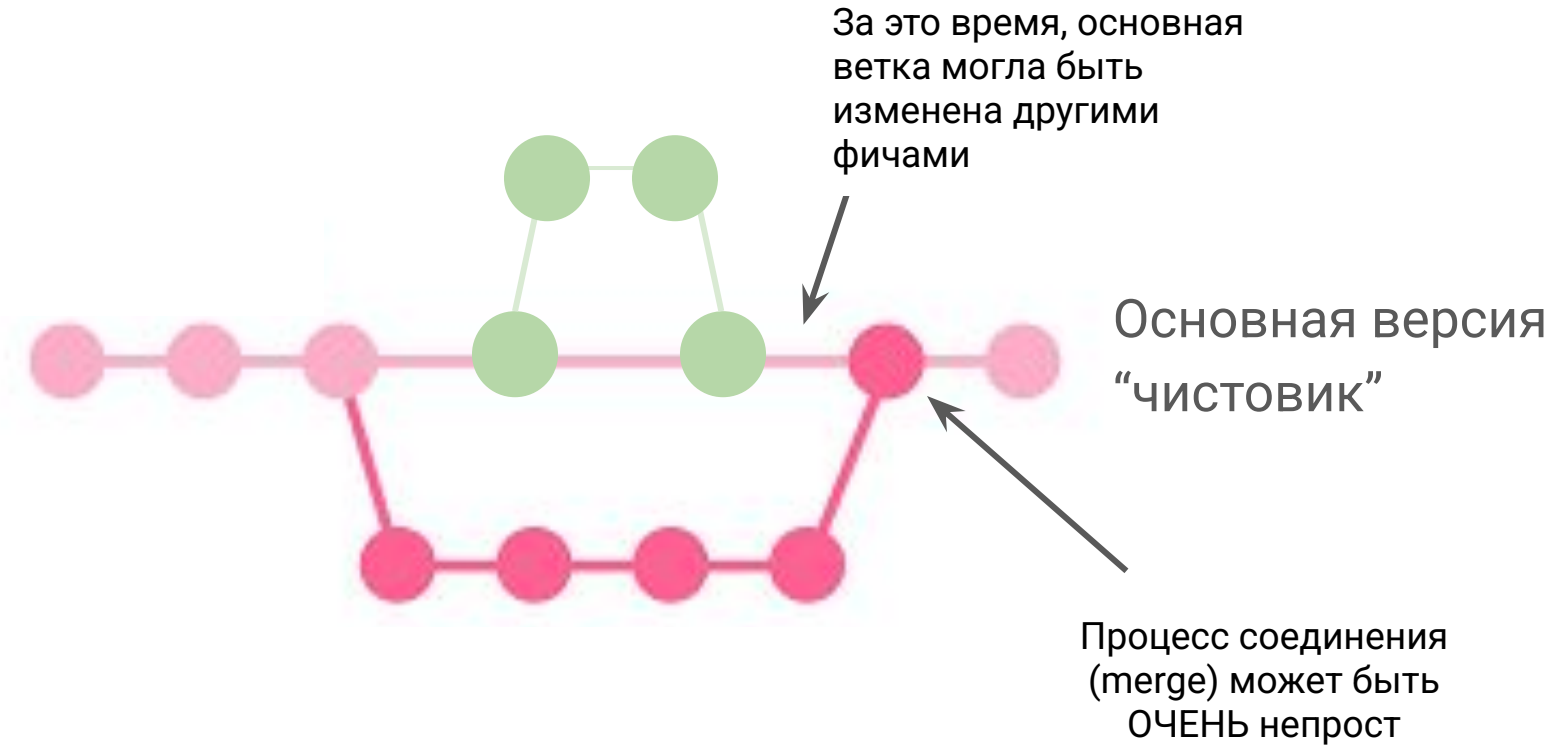


За это время, основная ветка могла быть изменена другими фичами

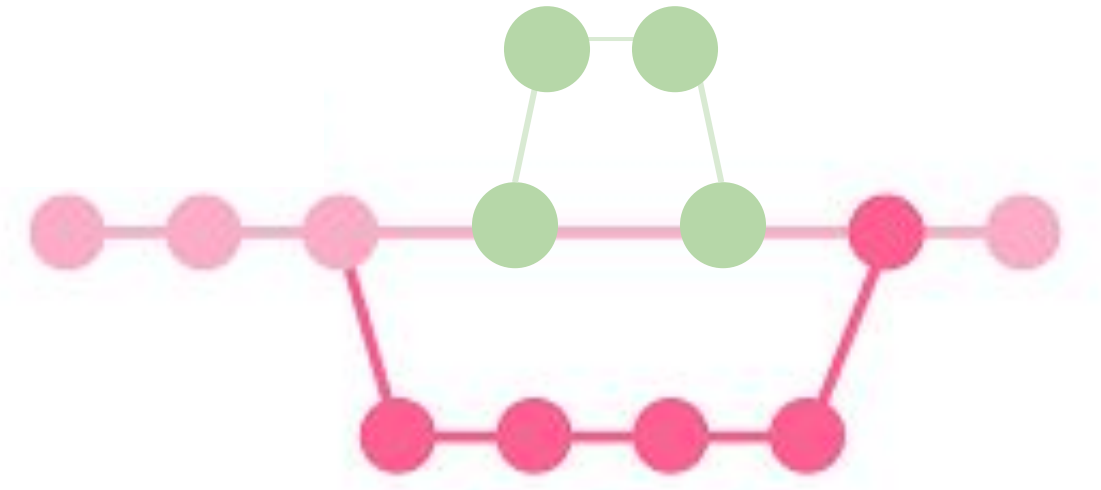
Основная версия “чистовик”

Фича закончена и протестирована. Можно соединяться с основной веткой.

Основной принцип работы

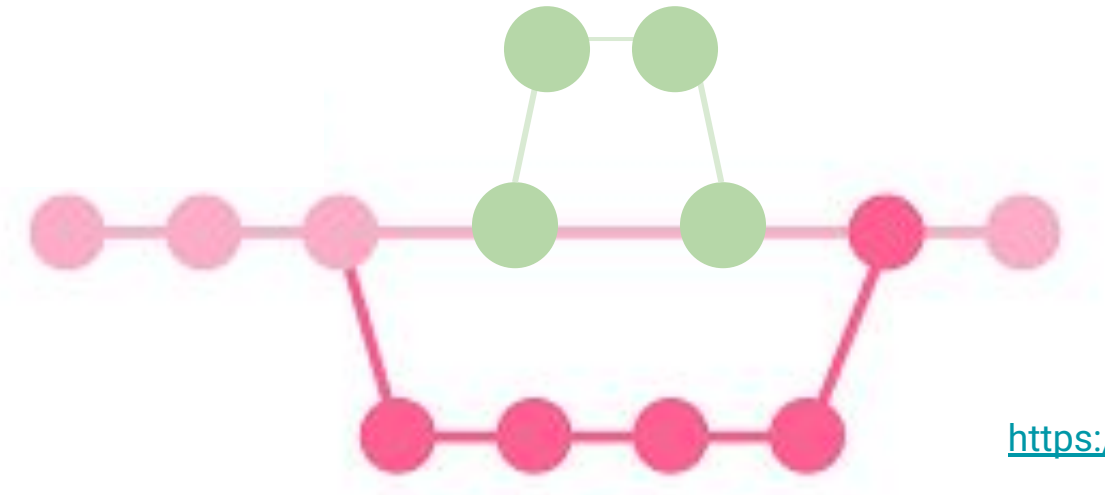


Основной принцип работы



Ветка чистовик раньше называлась **master**

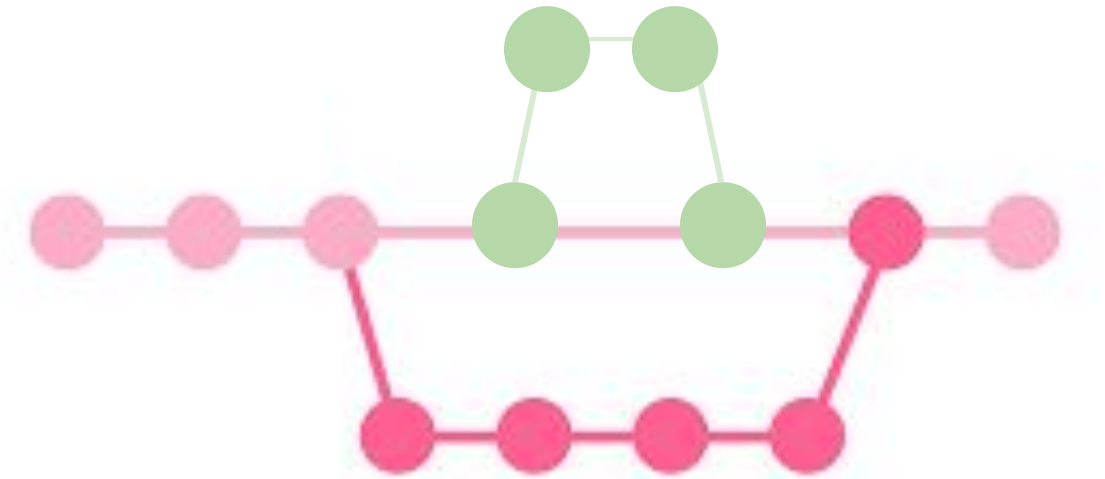
Основной принцип работы



Совсем недавно ее стали называть **main**

https://xakep.ru/2020/06/16/master-slave/

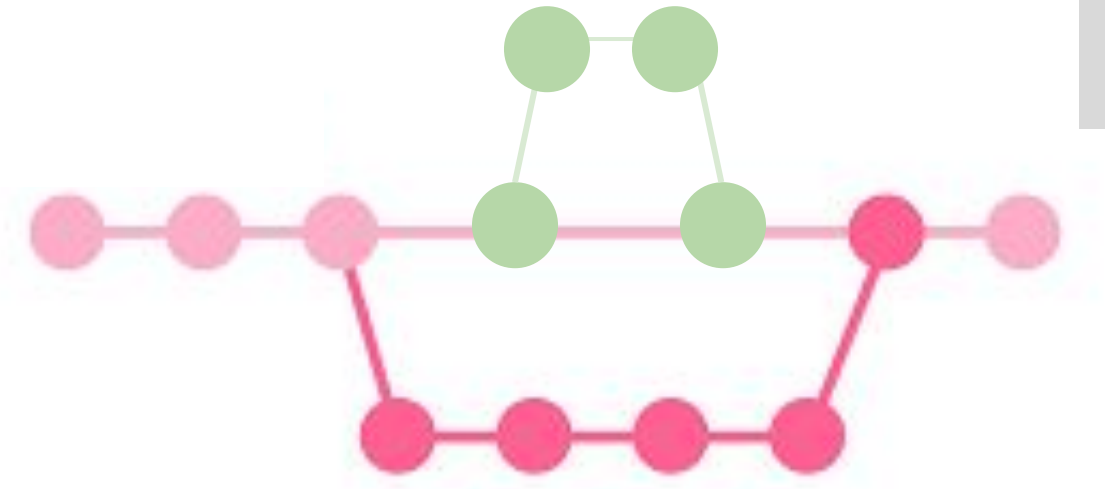
Основной принцип работы



main

Эти ветки именуются названиями фич

Основной принцип работы



Эти ветки именуются названиями фич

Есть еще несколько “особенных веток” с фиксированными названиями. Но мы пока не будем на этом останавливаться

main

Пример

Демо

- GitHub
- Colab
- PyCharm
- Git + Bash

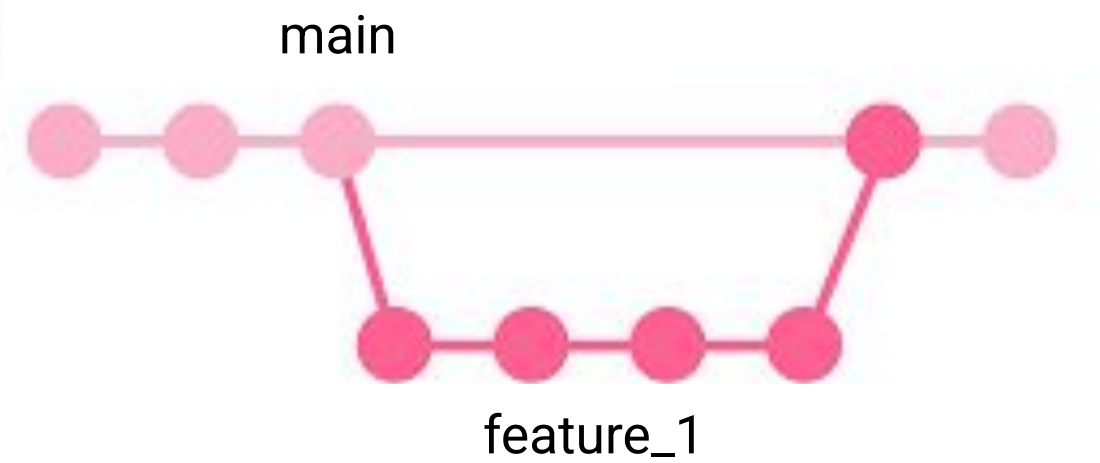

"SO, WHERE DID YOU TWO MEET?"
AT THE OFFICE
AT STARBUCKS
GITHUB
stickycomics.com
WINDOWS USERS
MAC USERS
LINUX USERS

Основные команды

Мы сейчас просто разберем команды git. А как их использовать - обсудим позже.

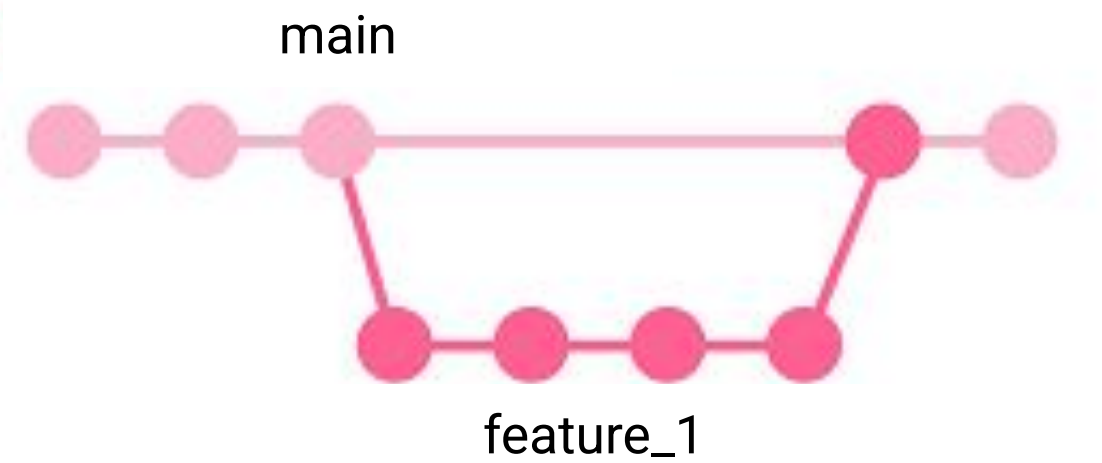
Основные команды

1. `checkout <название ветки>`



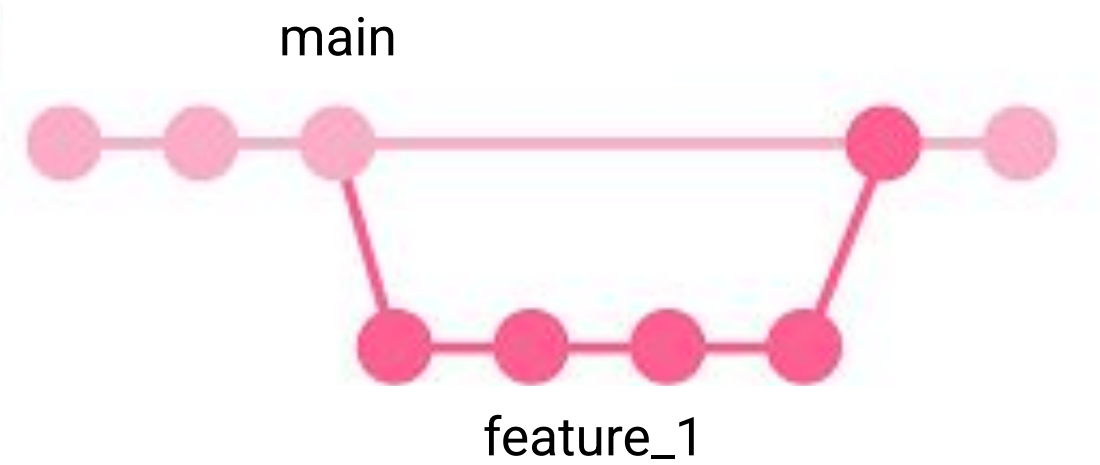
Основные команды

1. `checkout <название ветки>`
   Переключений с текущей рабочей ветки на редактирование ветки `<название ветки>`



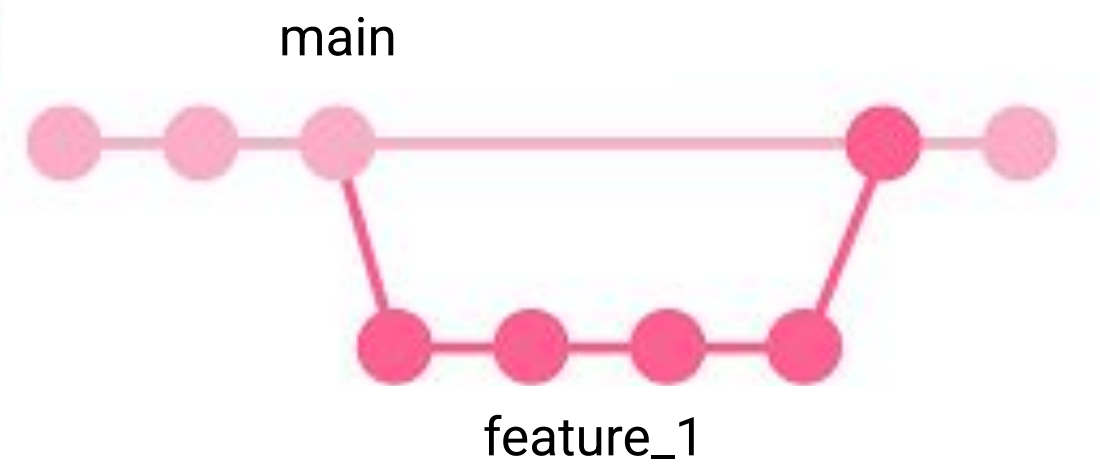
Основные команды

Проект состоит из отдельных файлов. Часть из них может отслеживаться SVC, часть - нет.



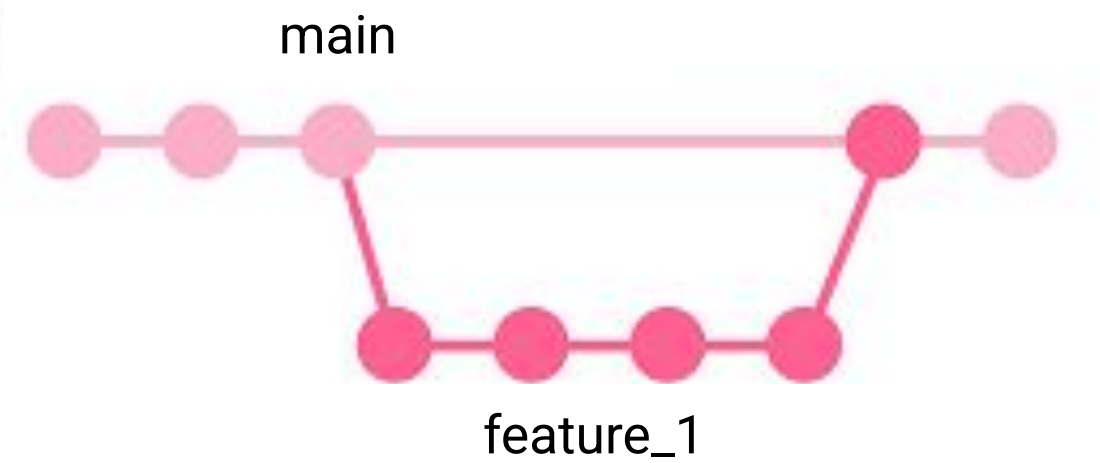
Основные команды

Совокупность всех отслеживаемых файлов = **репозиторий**



Основные команды

2. `add <название файла>`
   Подключить файл <название файла> к системе контроля версий.

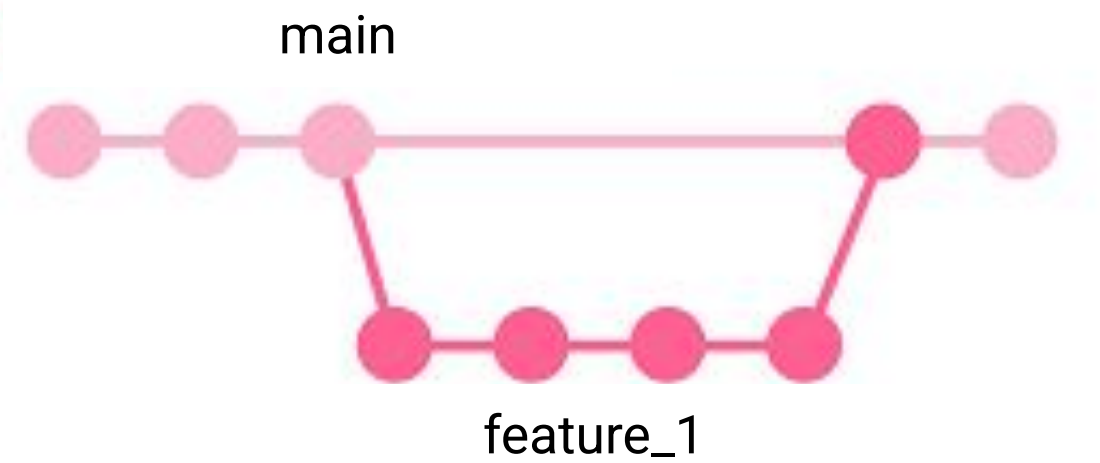


Основные команды

3. `commit <список файлов>`

Сохранить текущую версию некоторого `<списка файлов>`.

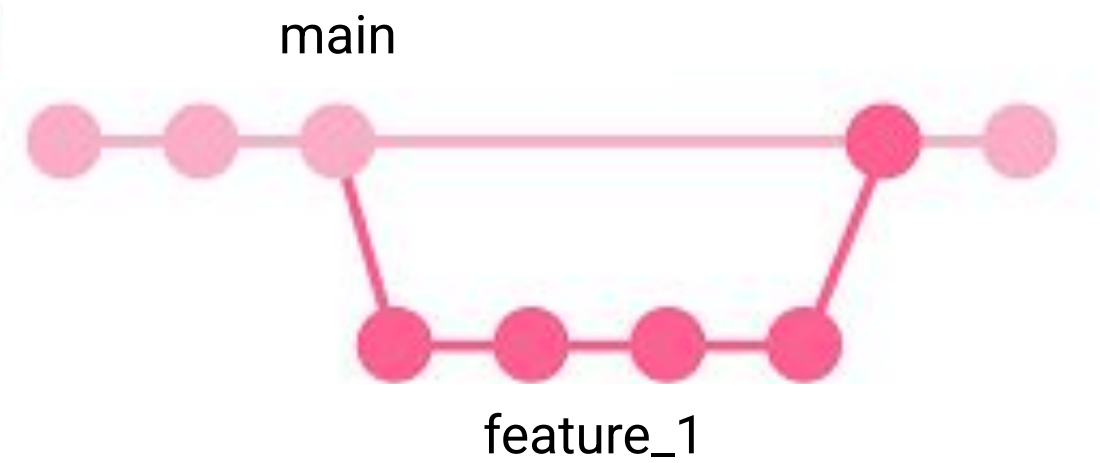
Каждой такой версии будет присвоено уникальное название.



Основные команды

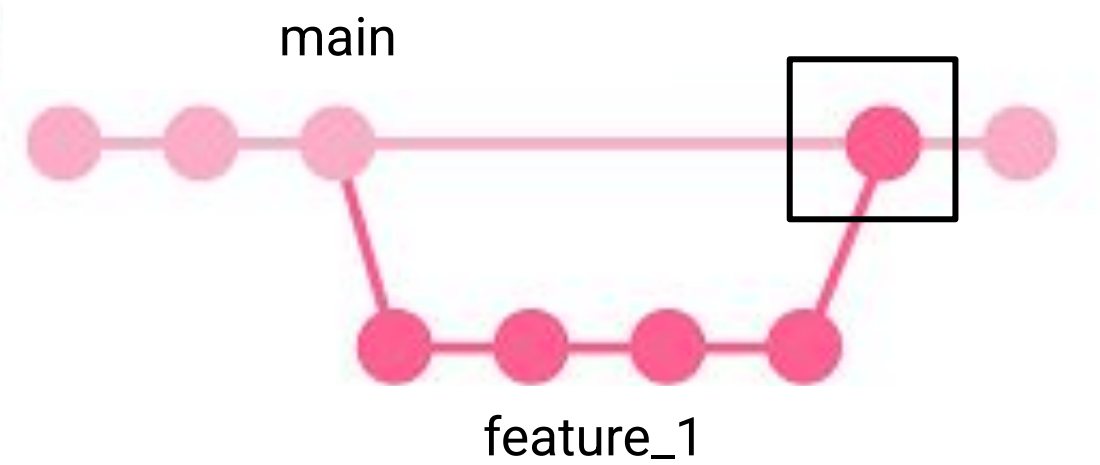
3. commit <список файлов>

Коммиты можно удалять. Если они не удалены, то к ним можно откатываться.



Основные команды

4. `merge <ветка_1> и <ветка_2>`
   `Слияние веток <ветка_1> и <ветка_2>`

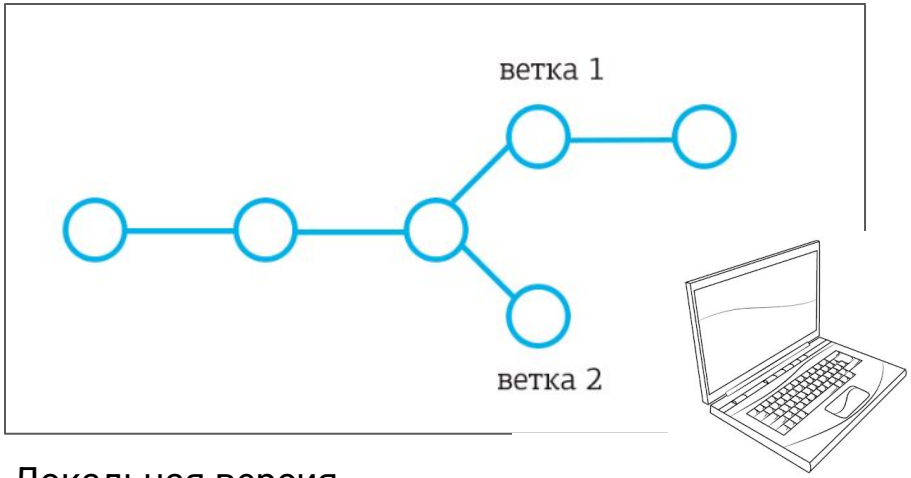


Основные команды

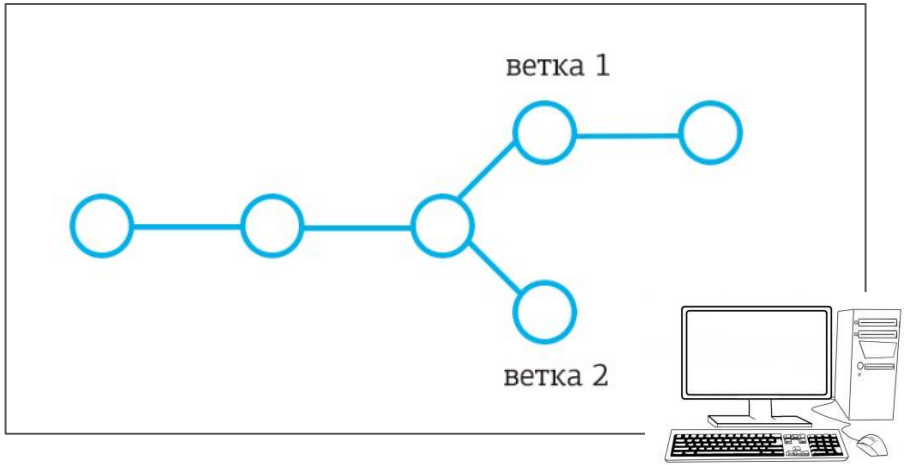
На самом деле, в физическом мире, каждая ветка существует на каждом компьютере участника проекта и на общем хостинге (главная - “Чистовик”).

И может изменяться параллельно несколькими участниками.

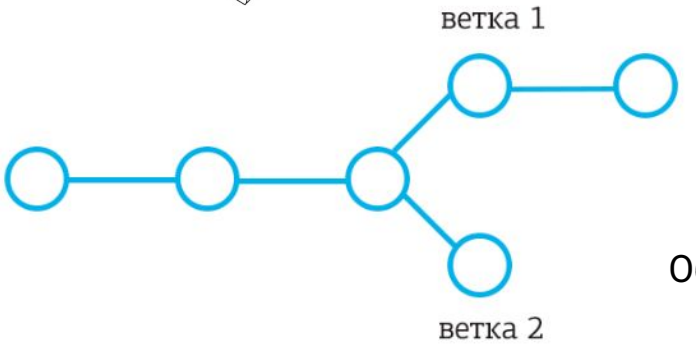
Основные команды



Локальная версия пользователя

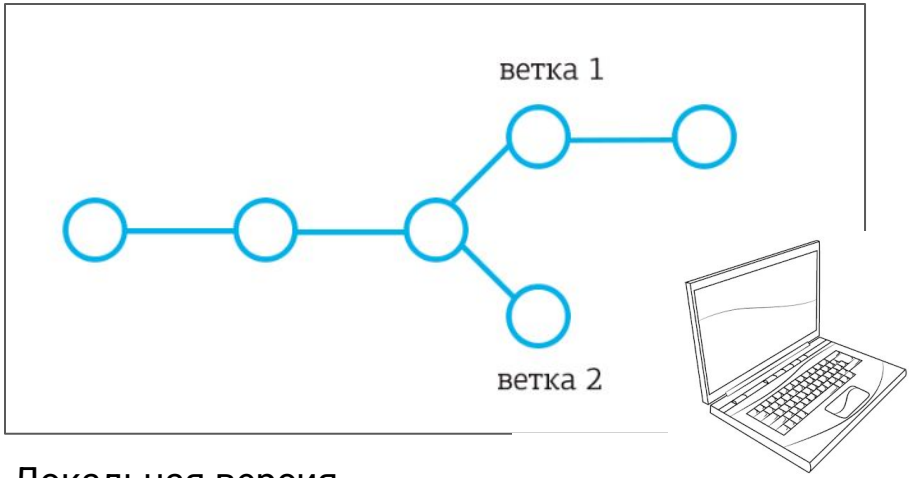


Локальная версия пользователя

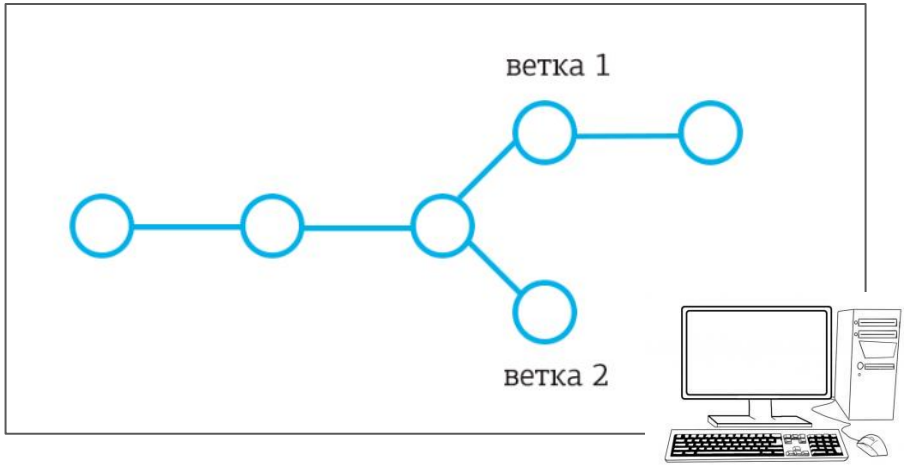


Общая версия на сервере

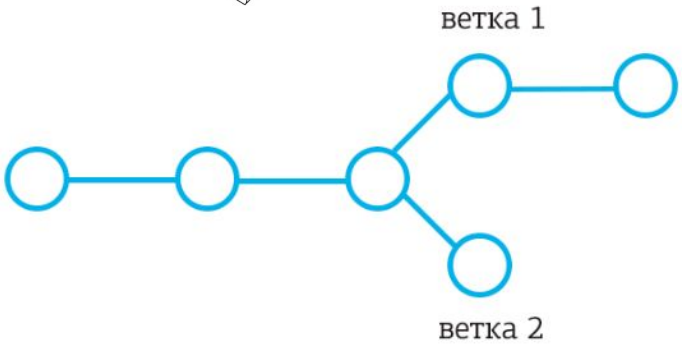
Основные команды



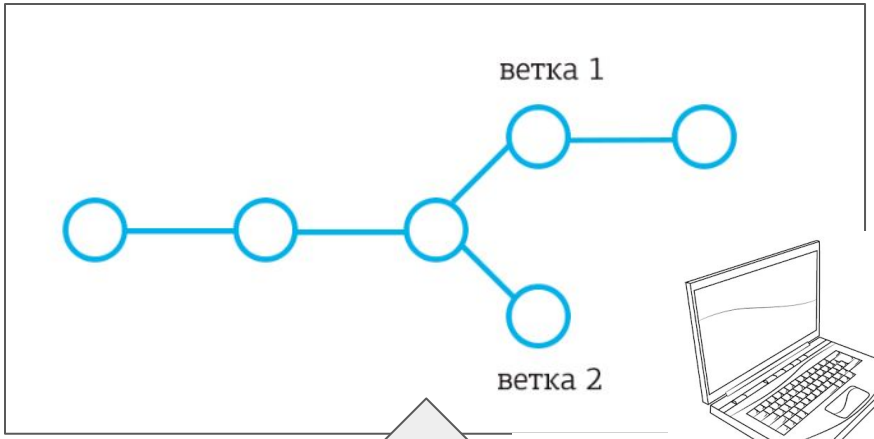
Локальная версия пользователя



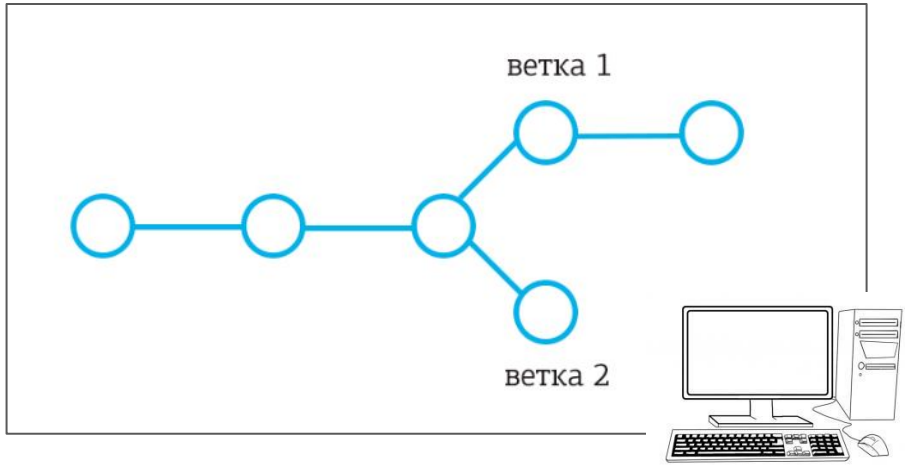
Локальная версия пользователя



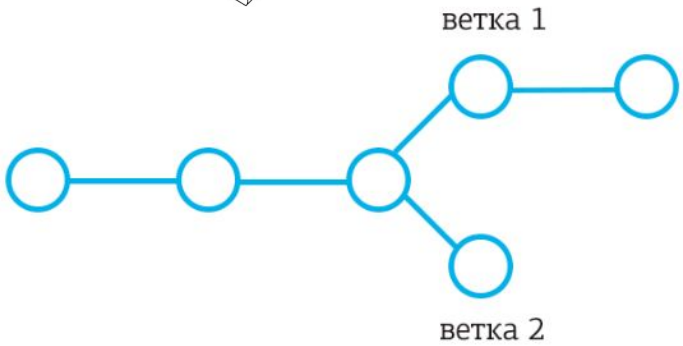
Основные команды



Локальная версия пользователя

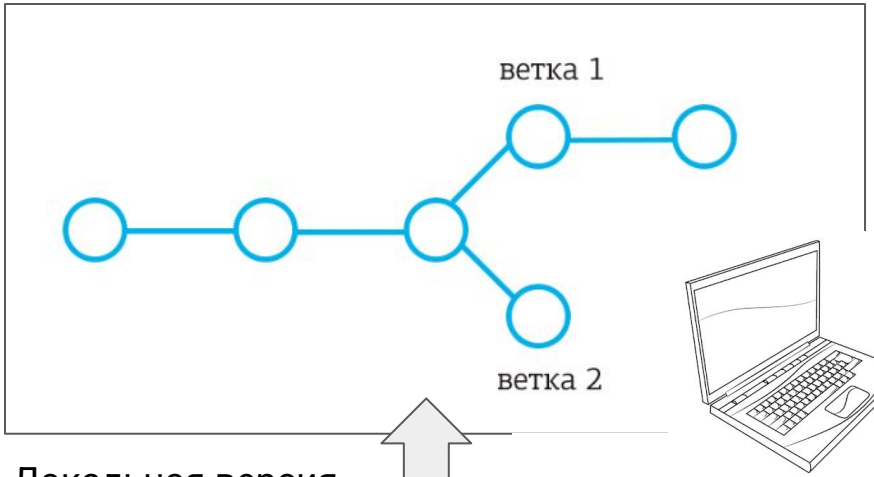


Локальная версия пользователя

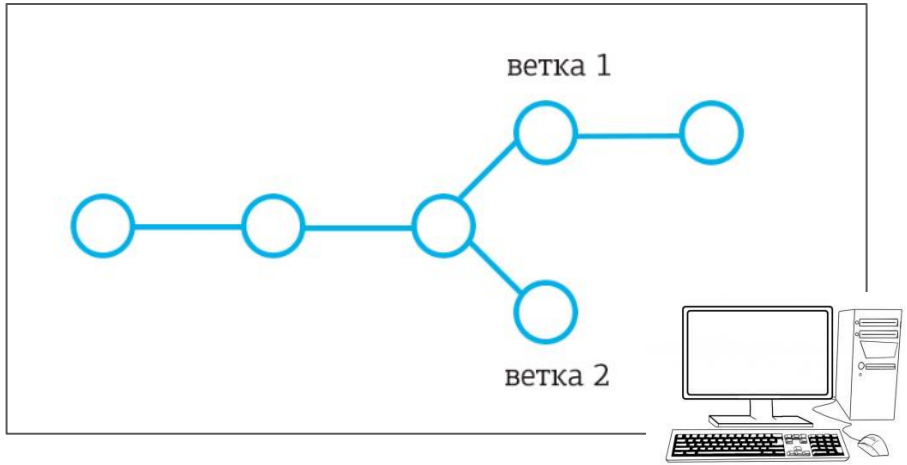


Мы будем пользоваться хостингом “github”

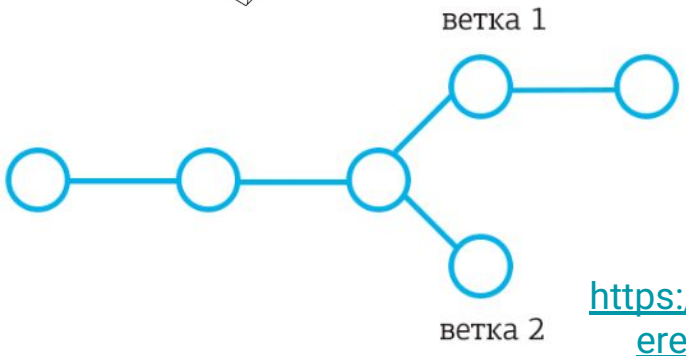
Основные команды



Локальная версия пользователя

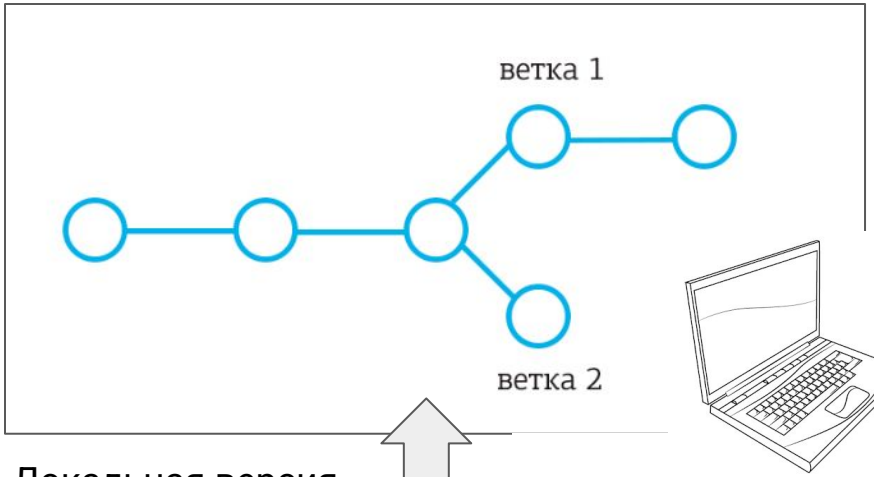


Локальная версия пользователя

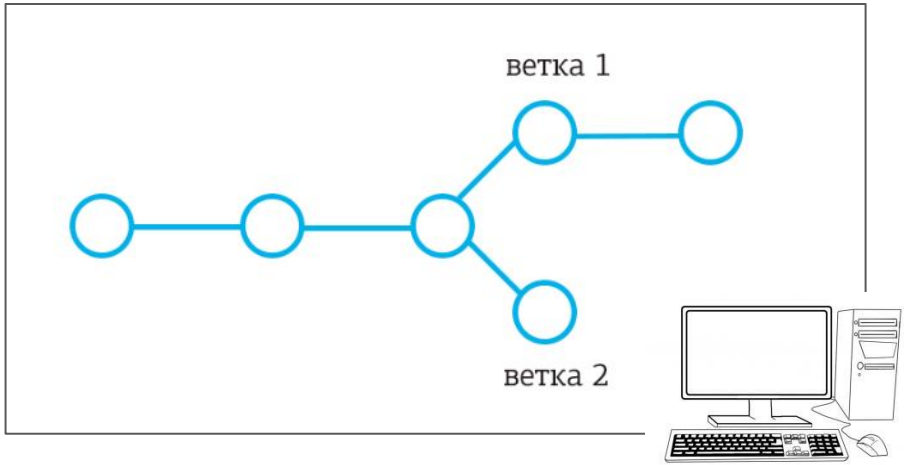


Мы будем пользоваться хостингом “github”
https://tproger.ru/translations/difference-between-git-and-github/

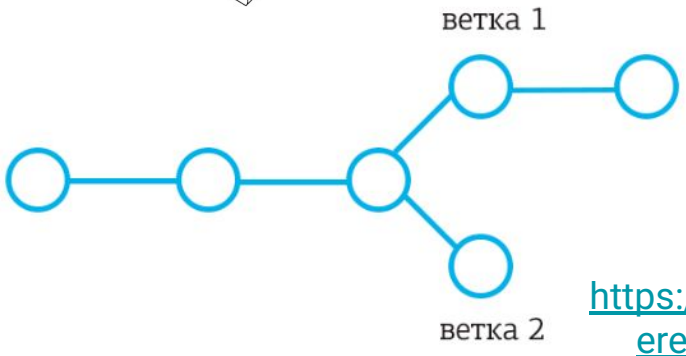
Основные команды



Локальная версия пользователя



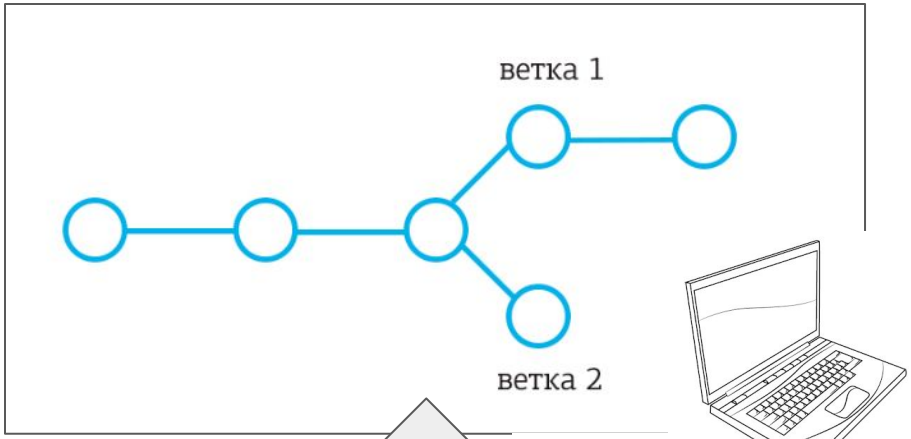
Локальная версия пользователя



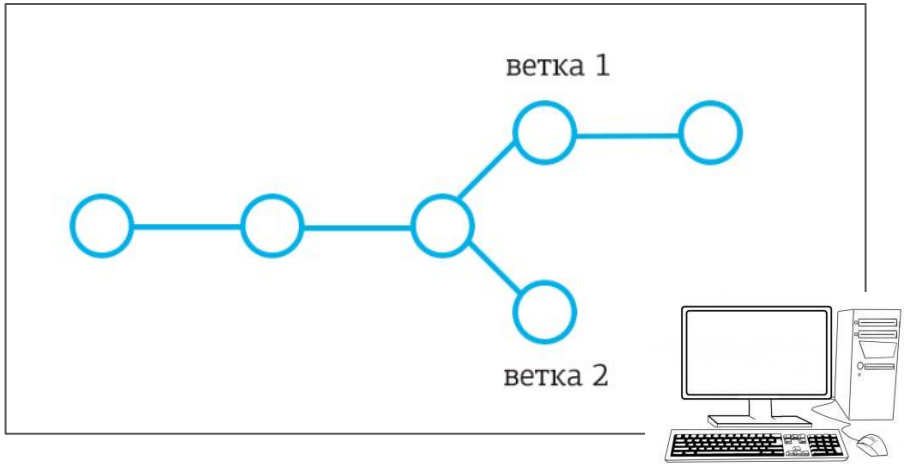

```
5. pull <источник>
```

Мы будем пользоваться хостингом “github”
https://tproger.ru/translations/difference-between-git-and-github/

Основные команды



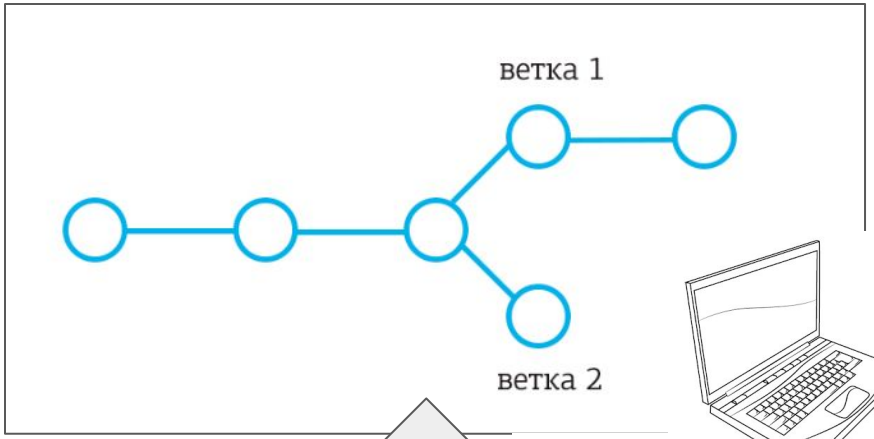
Локальная версия пользователя



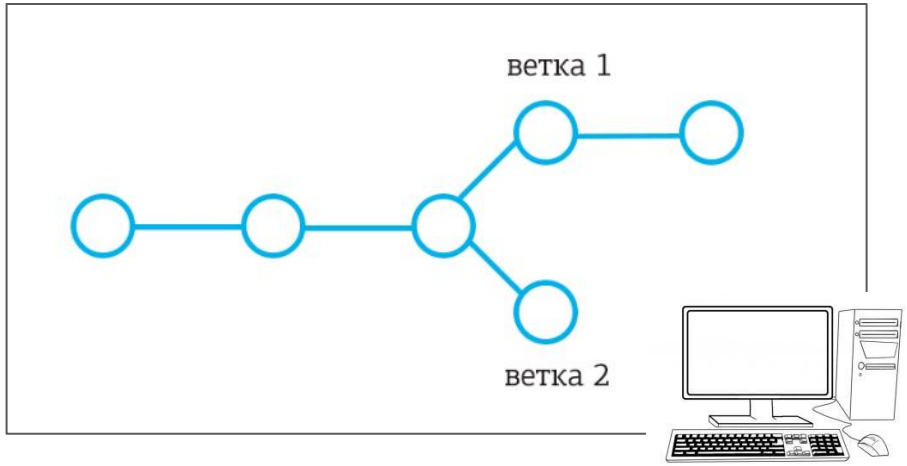
Локальная версия пользователя

5. pull <источник>
Копирует изменения с хостинга на локальную машину

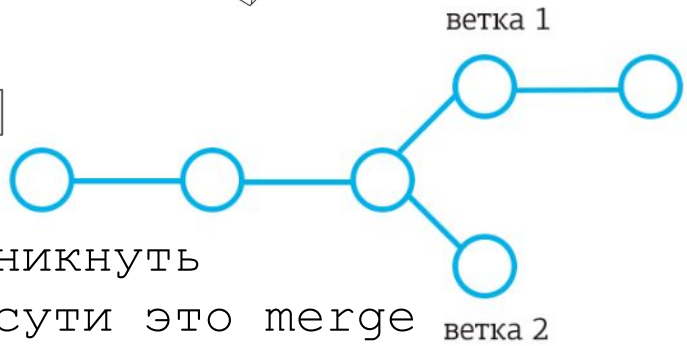
Основные команды



Локальная версия пользователя



Локальная версия пользователя



5. pull <источник>
При этому могут возникнуть конфликты. Т.к. по сути это merge

Основные команды

А в чем тогда разница между **pull** и **merge**?

Основные команды

А в чем тогда разница между **pull** и **merge**?

merge видит только локальные ветки

Основные команды

А в чем тогда разница между **pull** и **merge**?

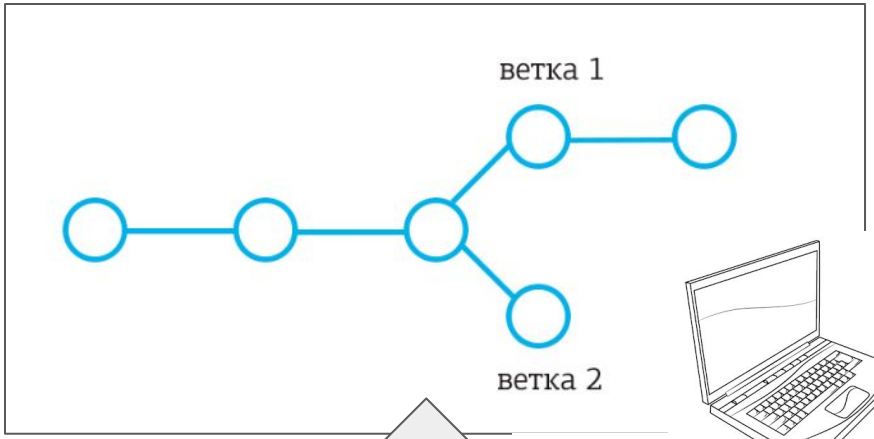
merge видит только локальные ветки
pull = **fetch** + **merge**

Основные команды

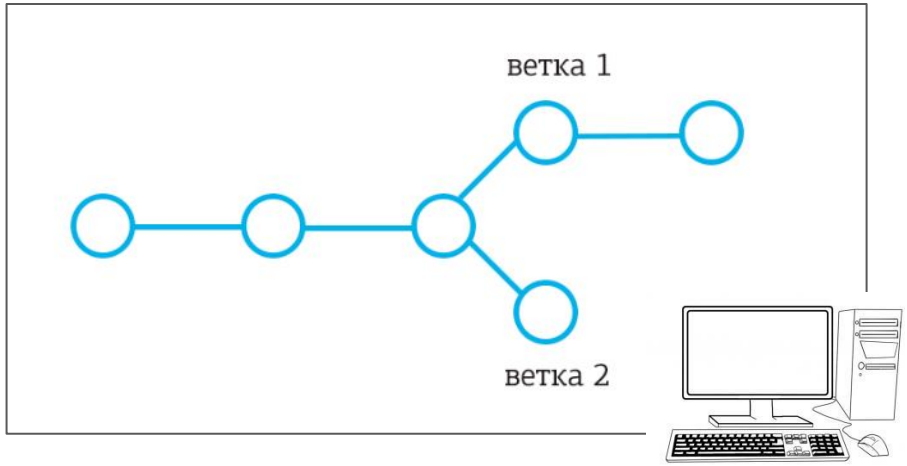
А в чем тогда разница между **pull** и **merge**?

merge видит только локальные ветки
pull = **fetch** (скачать с сервера все изменения) + **merge**

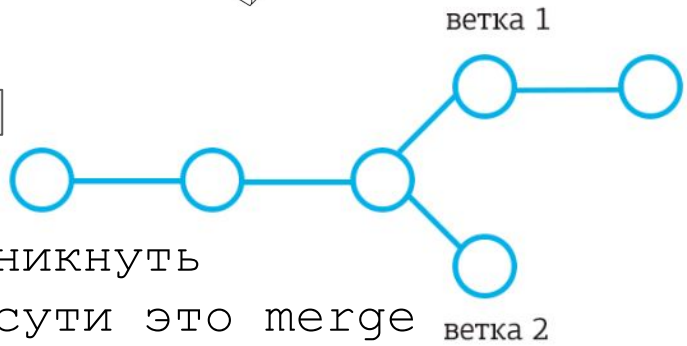
Основные команды



Локальная версия пользователя

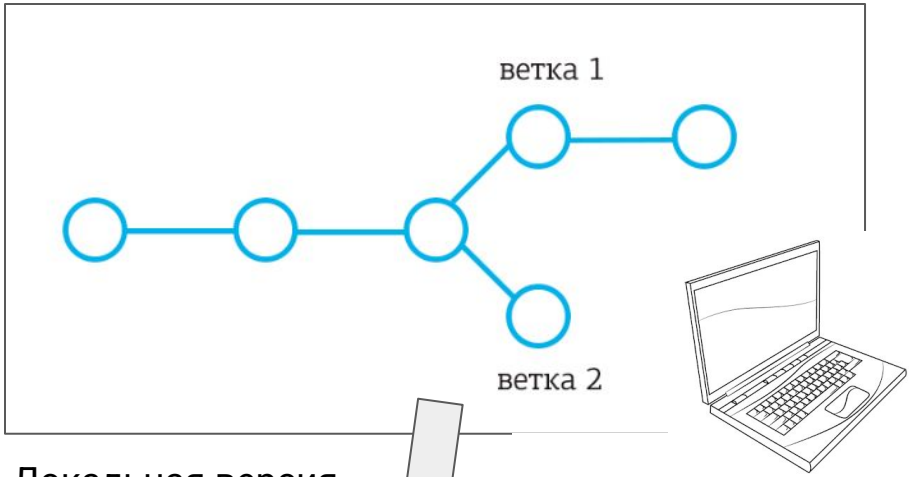


Локальная версия пользователя

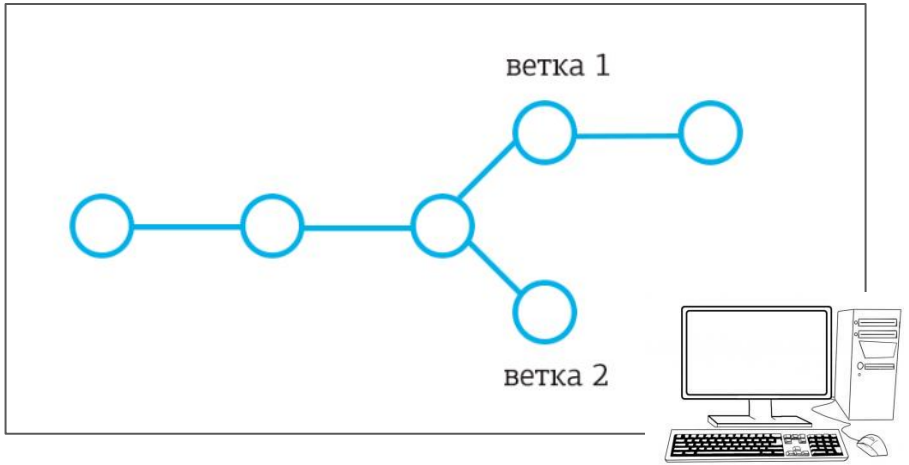


5. pull <источник>
При этому могут возникнуть конфликты. Т.к. по сути это merge

Основные команды



Локальная версия пользователя



Локальная версия пользователя

6. push <источник>
Отправить все изменения на сервер

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов
2. **`Commit`** для всех измененных файлов

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов
2. **`Commit`** для всех измененных файлов
3. **`Pull`** – вливаем все изменения с сервера (что там понаделали другие участники в нашей ветке?)

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов
2. **`Commit`** для всех измененных файлов
3. **`Pull`** – вливаем все изменения с сервера
4. **`Push`** – заливаем наши изменения на сервер

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов
2. **`Commit`** для всех измененных файлов
3. **`Pull`** – вливаем все изменения с сервера
4. **`Push`** – заливаем наши изменения на сервер

Исключение для ветки master.

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов
2. **`Commit`** для всех измененных файлов
3. **`Pull`** – вливаем все изменения с сервера
4. **`Push`** – заливаем наши изменения на сервер

Исключение для ветки master. Пункт 4 делает всегда один человек или группа людей на ревью.

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов
2. **`Commit`** для всех измененных файлов
3. **`Pull`** – вливаем все изменения с сервера
4. **`Push`** – заливаем наши изменения на сервер

Исключение для ветки master. Пункт 4 делает всегда один человек или группа людей на ревью. Чтобы он узнал о необходимость соединения веток, надо дать специальную команду на github (Pull request)

Итого

Вы сделали изменения в текущей ветке. Как сделать их доступными для всех?

1. **`Add`** для всех новых файлов
2. **`Commit`** для всех измененных файлов
3. **`Pull`** – вливаем все изменения с сервера
4. **`Push`** – заливаем наши изменения на сервер

Исключение для ветки master. Пункт 4 делает всегда один человек или группа людей на ревью. Чтобы он узнал о необходимость соединения веток, надо дать специальную команду на github (Pull request - по сути это push request, но исторически сложилось название pull request)

Основные команды

7. А с чего начать?

Основные команды

7. А с чего начать?

Вариант 1: склонировать имеющийся проект с сервера себе

```
clone <адрес репозитория>
```

Основные команды

7. А с чего начать?

Вариант 1: склонировать имеющийся проект с сервера себе

```
clone <адрес репозитория>
```

Вариант 2: наоборот, создать на сервере новый проект из своего локального

```
init <адрес репозитория>
```

Итого

Основное:

1. git add
2. git commit
3. git checkout
4. git merge
5. git branch

Взаимодействия с удаленным репозиторием:

1. git clone
2. git pull
3. git fetch
4. git push
5. git init

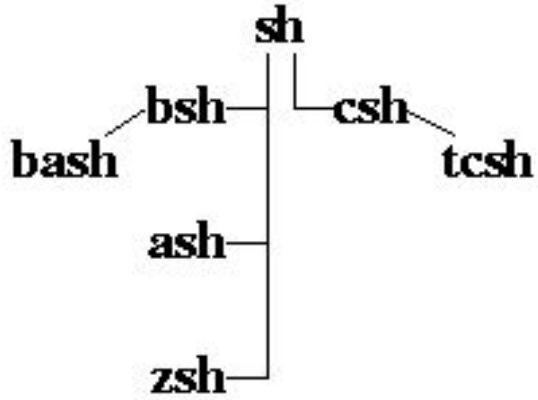
Самопроверка

1. Чем Git отличается от GitHub?
2. Чем отличается `commit` от `add`?
3. Чем отличается `clone` от `pull`?
4. Чем отличается `merge` от `pull`?
5. Как переводится “Ветка” на английский язык?

Shell

Shell

- **Unix shell** («sh») — командный интерпретатор, используемый в операционных системах семейства Unix
- **Bash** (Bourne again shell) — это стандартная командная оболочка в большинстве дистрибутивов Linux и macOS, а также язык для этой оболочки.



Примеры команд

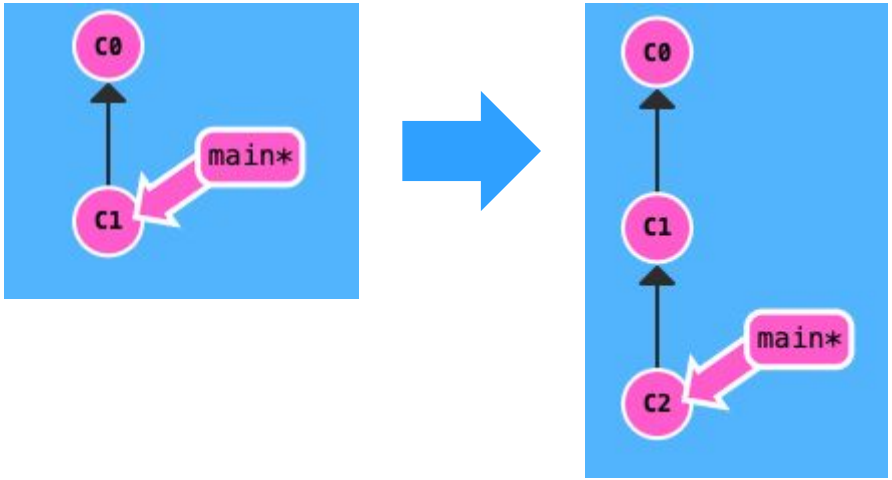
- `ls`
- `cd` **`name_of_dir`**
  - `cd ..`
- `mkdir` **`name_of_dir`**
- **<ЧЕМ ЗАПУСКАЕМ?> <ЧТО ЗАПУСКАЕМ?> <ПАРАМЕТРЫ>**
  - `python script.py 100 500`

Бренчинг

Коммит

`git commit`

(англ.) совершить, зафиксировать

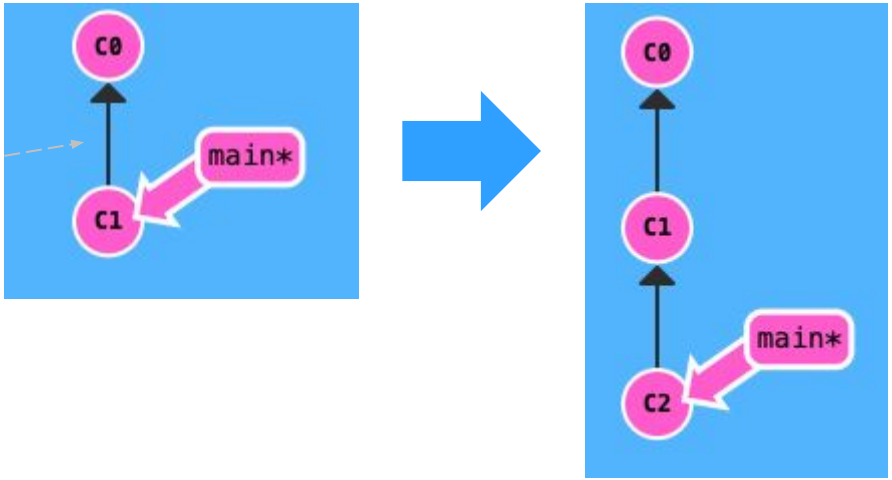


Коммит

```
git commit
```

Сохранить набор изменений (дельту)

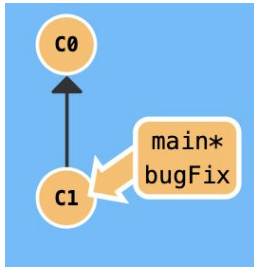
Поэтому всегда важно понимать, кто ваш родитель?



Создание веток

```
git commit
```

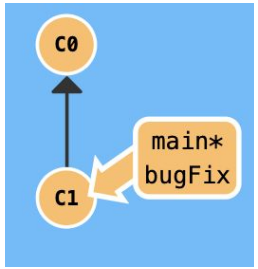
```
git branch bugFix
```

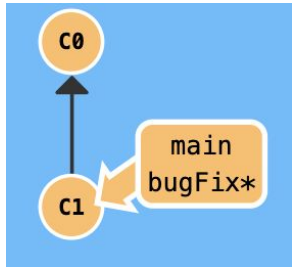


Создание и переключение веток

```
git commit

git branch bugFix
git checkout bugFix
```



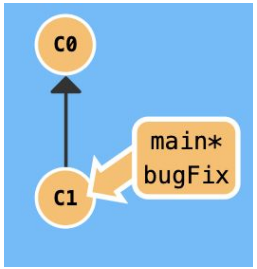


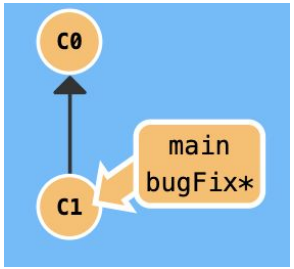
Создание и переключение веток

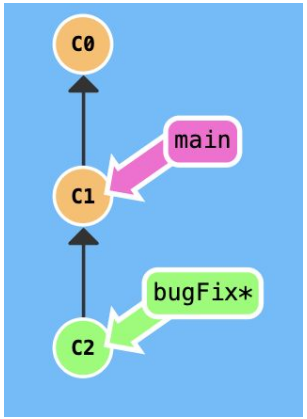
```
git commit

git branch bugFix
git checkout bugFix

git commit
```







Создание и переключение веток

```
git commit
```

```
git branch bugFix
git checkout bugFix
```

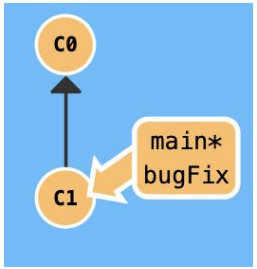
} `git checkout -b bugFix`

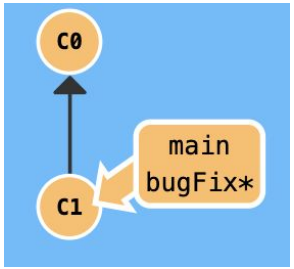
Создание и переключение веток

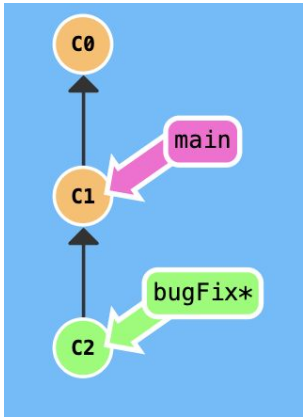
```
git commit

git branch bugFix
git checkout bugFix

git commit
```







Способы слияния

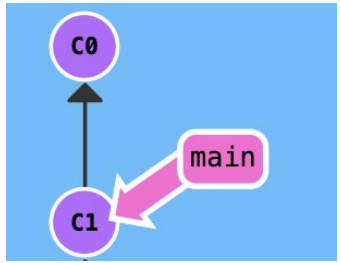
1: Merge

- Особый вид коммита, который имеет сразу двух родителей

`git merge bugFix` = "вмержи в меня `bugFix`"

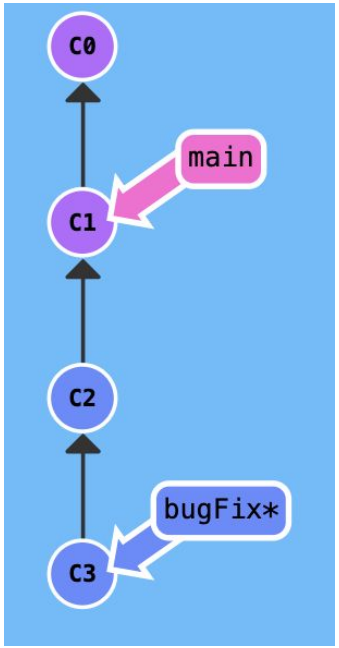
1: Merge

```
git checkout -b bugFix
git commit
git commit
```



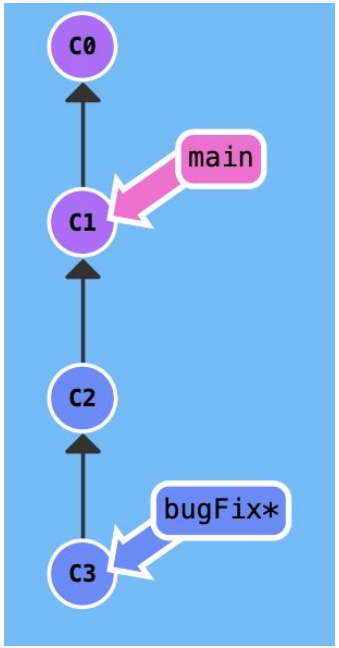
1: Merge

```
git checkout -b bugFix
git commit
git commit
```



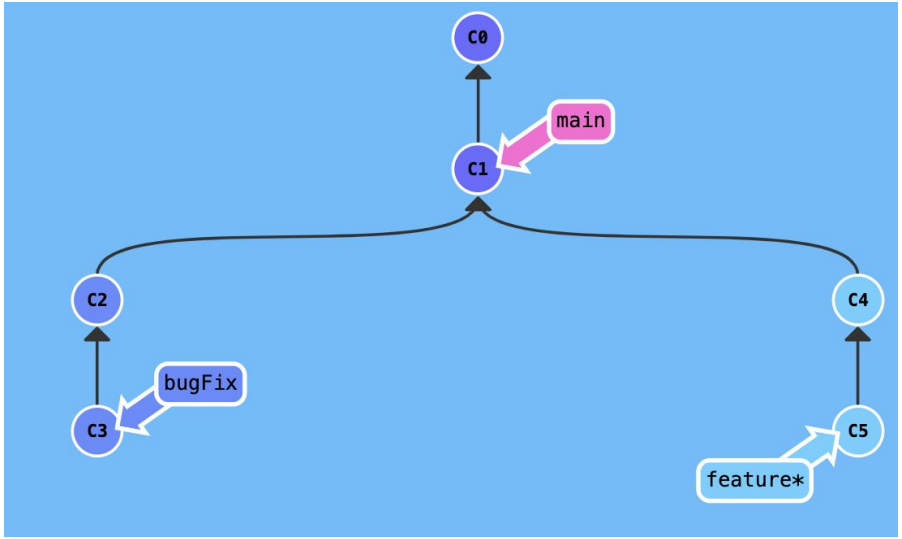
1: Merge

```
git checkout -b bugFix
git commit
git commit
git checkout main
git checkout -b feature
git commit
git commit
```



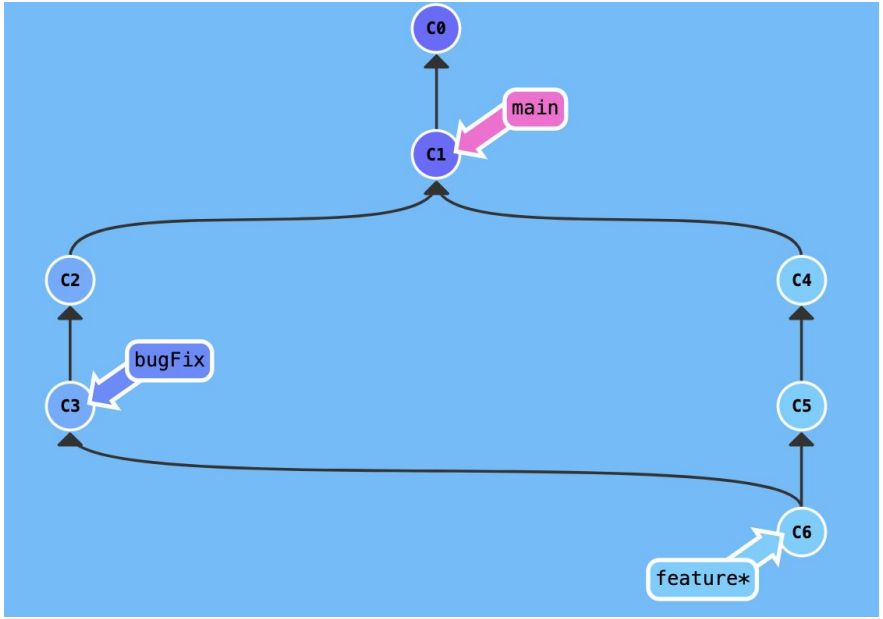
1: Merge

```
git checkout -b bugFix
git commit
git commit
git checkout main
git checkout -b feature
git commit
git commit
```



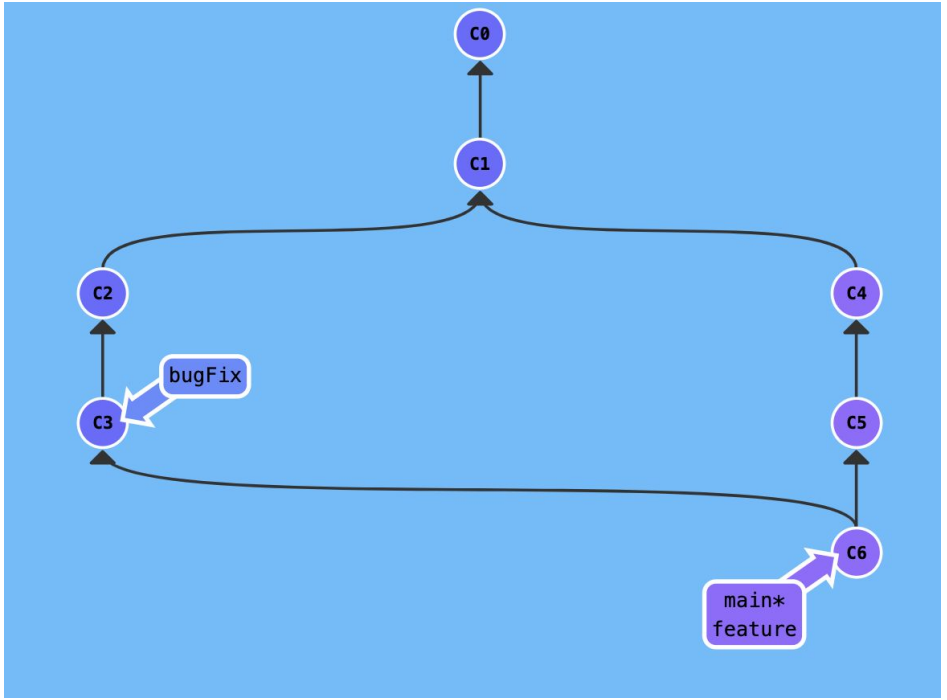
1: Merge

```
git checkout -b bugFix
git commit
git commit
git checkout main
git checkout -b feature
git commit
git commit
git merge bugfix
```



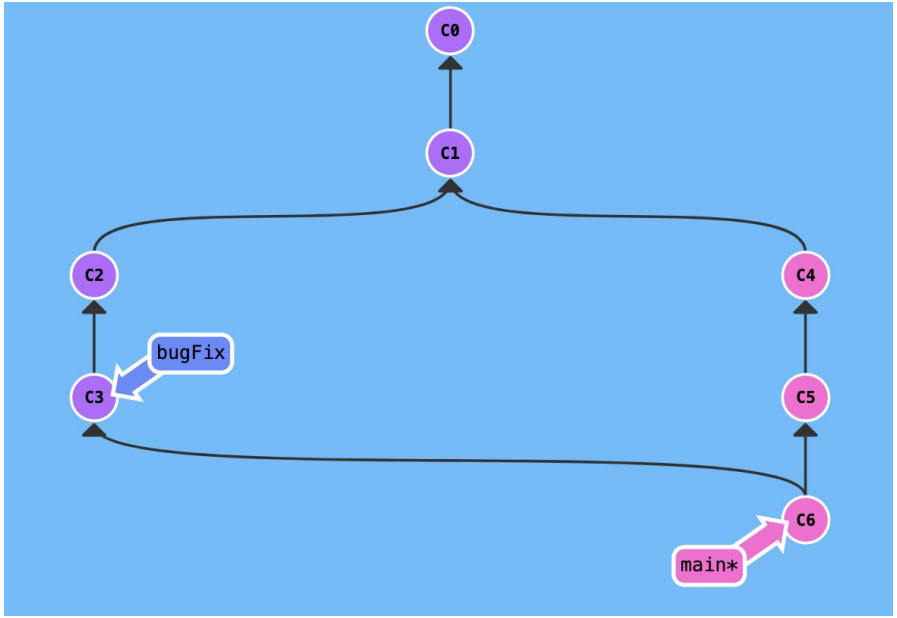
1: Merge

```
git checkout -b bugFix
git commit
git commit
git checkout main
git checkout -b feature
git commit
git commit
git merge bugfix
git checkout main
git merge feature
```



Удаление

```
git checkout -b bugFix
git commit
git commit
git checkout main
git checkout -b feature
git commit
git commit
git merge bugfix
git checkout main
git merge feature
git branch -d feature
```



Что уже знаем?

1. `git commit`
2. `git branch bugFix`
3. `git branch -d bugFix`
4. `git checkout main`
5. `git checkout -b bugFix`
6. `git merge bugFix`

2: Rebase

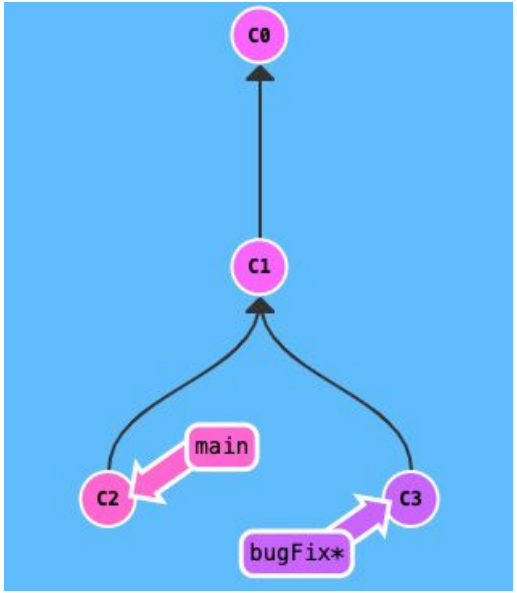
- Копирует набор коммитов и переносит их в другое место.
- **Преимущество** – можно делать чистые и красивые линейные последовательности коммитов.

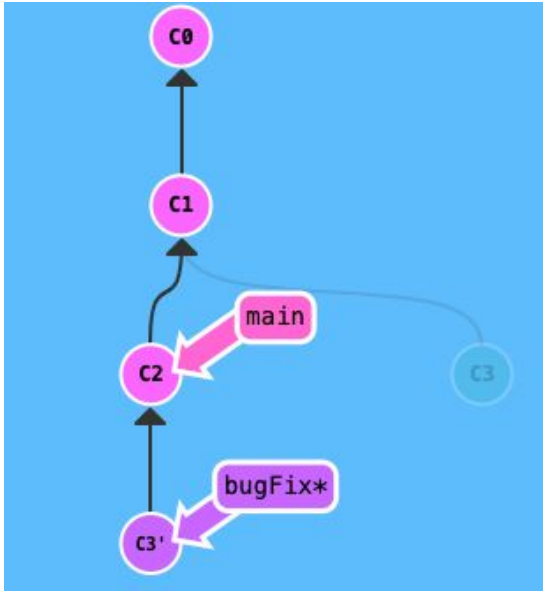
`git merge bugFix` = "вмержи в меня `bugFix`"

`git rebase bugFix` = "перенеси меня на `bugFix`"

2: Rebase

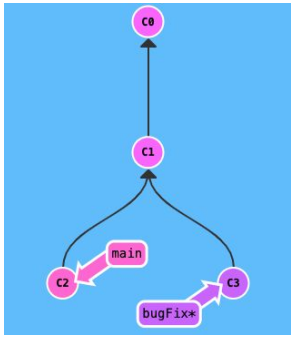
```
git rebase main
```

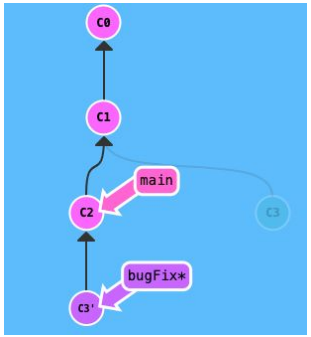


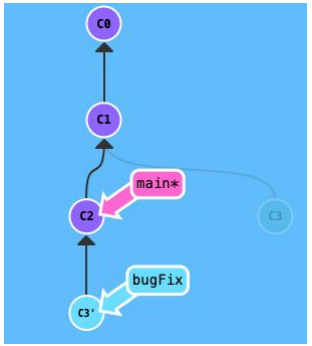


2: Rebase

```
git rebase main
git checkout main
```

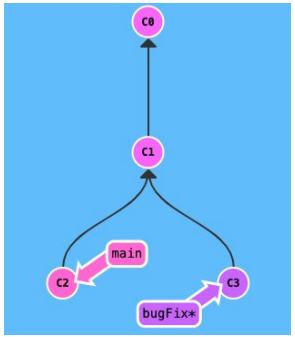


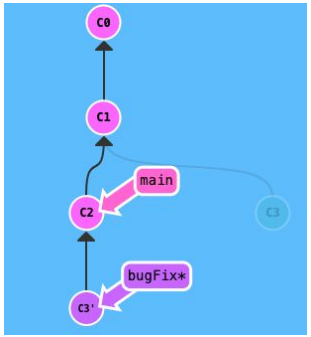


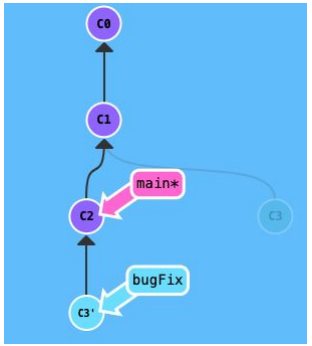


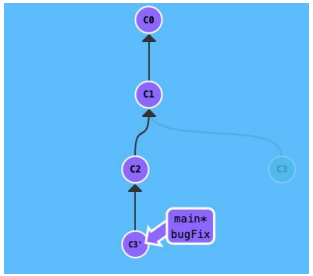
2: Rebase

```
git rebase main
git checkout main
git rebase bugFix
```









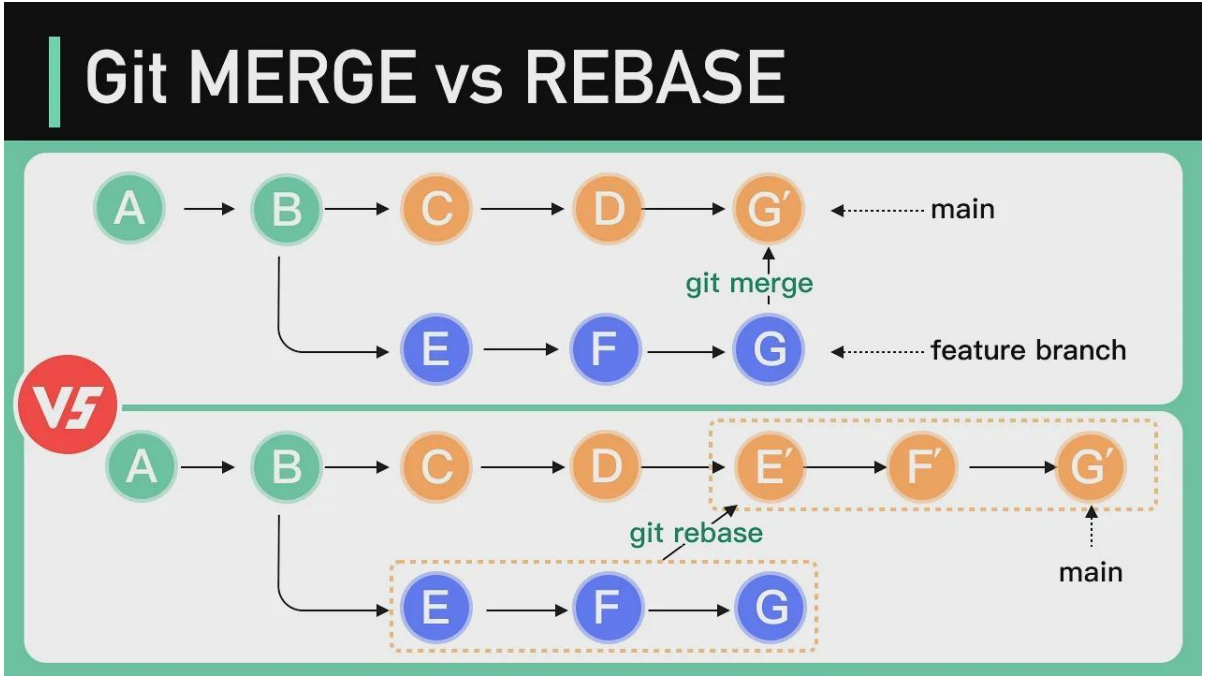
Merge vs. Rebase

- Особый вид коммита, который имеет сразу двух родителей

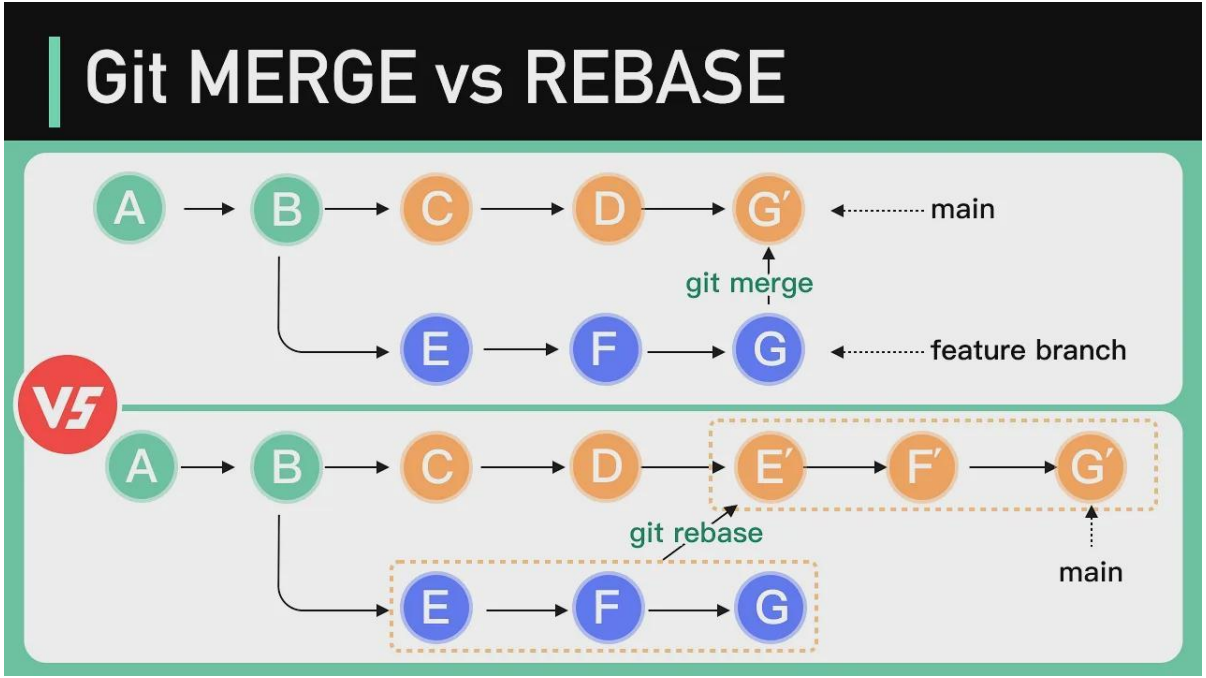
`git merge bugFix` = "вмержи в меня `bugFix`"

`git rebase bugFix` = "перенеси меня на `bugFix`"

Merge vs. Rebase



Merge vs. Rebase



Head

HEAD

- **HEAD** - это символическое имя текущего выбранного коммита (тот коммит, над которым мы в данный момент работаем)
- HEAD всегда указывает на последний коммит из вашего локального дерева.
- **Отделение** (detaching) HEAD означает лишь присвоение его не ветке, а конкретному коммиту.

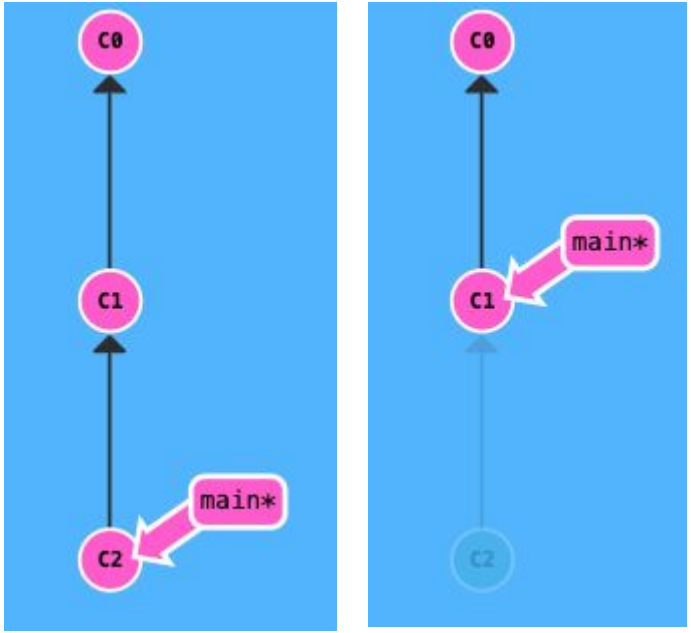
Reset

Для локальной ветки

```
git reset HEAD^
```

или

```
git reset HEAD~1
```



Для удаленной ветки

```
git revert HEAD
```

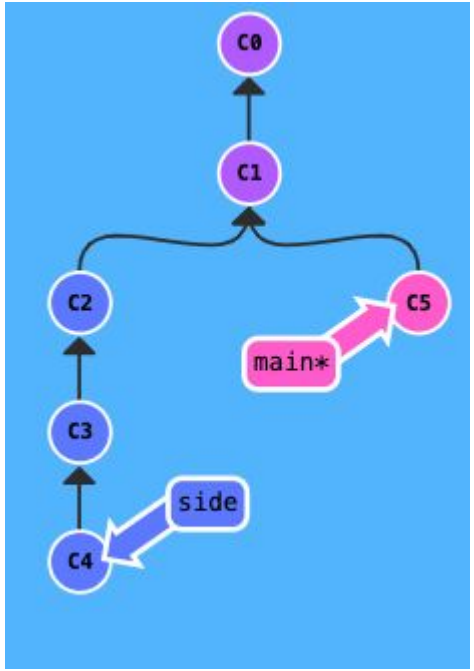
Cherry pick

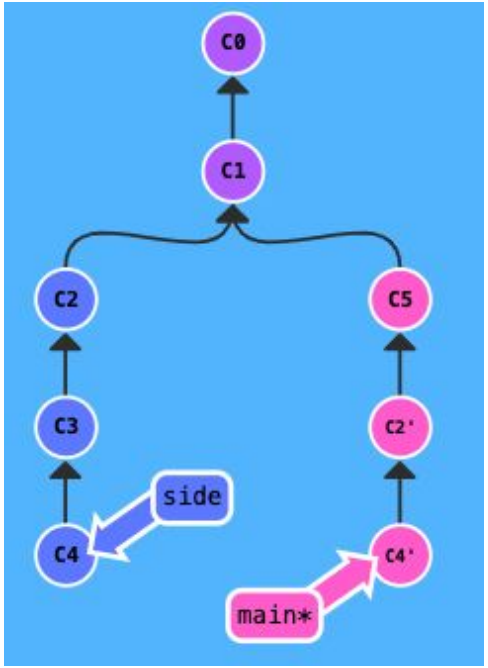
Перенос изменений

```
git cherry-pick <Commit1> <Commit2> <...>
```

Перенос изменений

```
git cherry-pick C2 C4
```





Домашняя практика

Вспомним

Можно ли пушить в main?

Вспомним

А как надо поступать?

Использованные материалы

- При создании презентации были использованы материалы Школы “ДИО-ГЕН”