



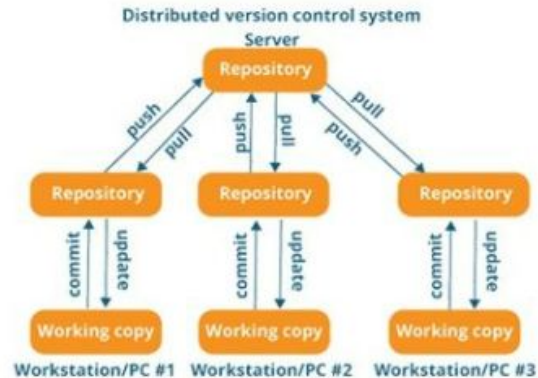
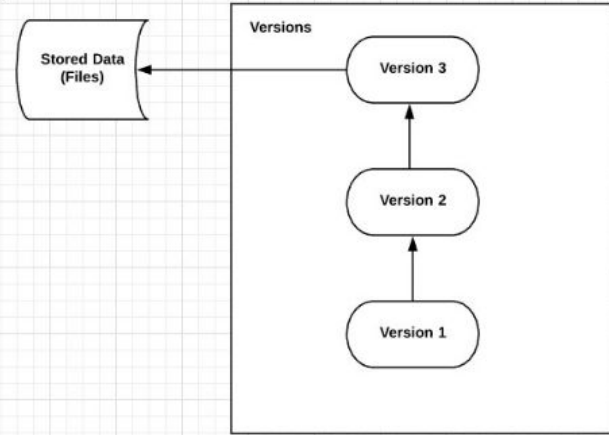
Lesson 36

02.03.2023

Types of Version Control System

- Local Version Control System
- Centralized Version Control System
- Distributed Version Control System

Local Computer

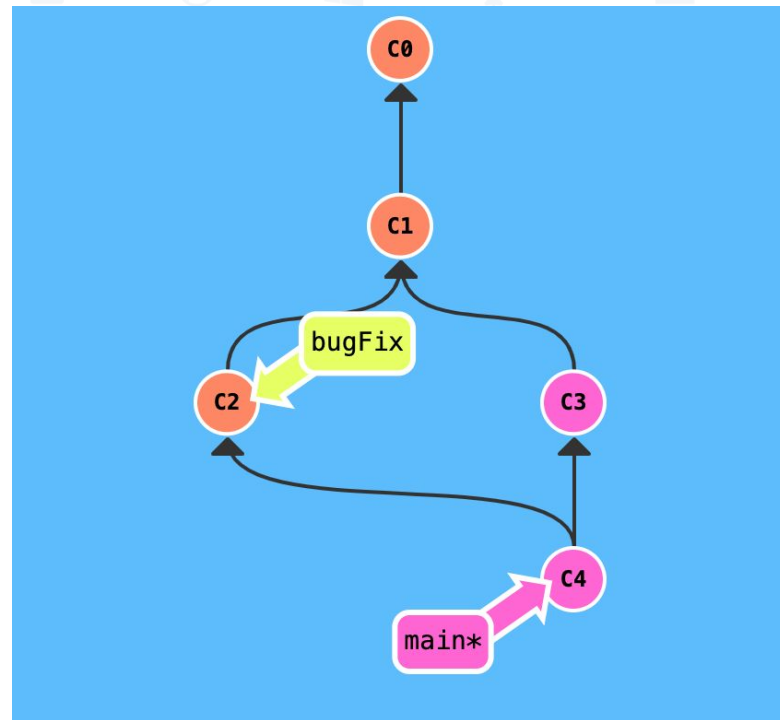
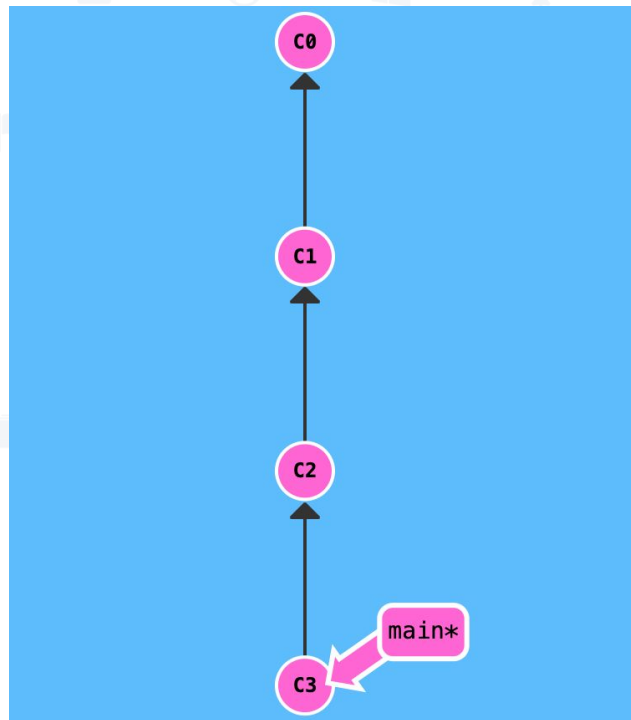


Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года



Как работает

Если посмотреть на картинку, то становится чуть проще с пониманием. Каждый кружок, это commit. Стрелочки показывают направление, из какого commit сделан следующий. Например C3 сделан из C2 и т. д. Все эти commit находятся в ветке под названием main. Это основная ветка, чаще всего ее называют master. Прямоугольник main* показывает в каком commit мы сейчас находимся, проще говоря указатель.



Настройка

Вы установили себе Git и можете им пользоваться. Давайте теперь его настроим, чтобы когда вы создавали commit, указывался автор, кто его создал.

Открываем терминал (Linux и MacOS) или консоль (Windows) и вводим следующие команды.

```
#Установим имя для вашего пользователя
```

```
#Вместо <ваше_имя> можно ввести, например, Grisha_Popov
```

```
#Кавычки оставляем
```

```
git config --global user.name "<ваше_имя>"
```

```
#Теперь установим email. Принцип тот же.
```

```
git config --global user.email "<адрес_почты@email.com>"
```



Создание репозитория

Теперь вы готовы к работе с Git локально на компьютере.

Создадим наш первый репозиторий. Для этого пройдите в папку вашего проекта.


#Для Linux и MacOS путь может выглядеть так `/Users/UserName/Desktop/MyProject`

#Для Windows например `C://MyProject`

`cd <путь_к_вашему_проекту>`

#Инициализация/создание репозитория

`git init`



Теперь Git отслеживает изменения файлов вашего проекта. Но, так как вы только создали репозиторий в нем нет вашего кода. Для этого необходимо создать commit.

#Добавим все файлы проекта в нам будущий commit

```
git add .
```

#Или так

```
git add --all
```


#Если хотим добавить конкретный файл то можно так

```
git add <имя_файла>
```

#Теперь создаем commit. Обязательно указываем комментарий.

#И не забываем про кавычки

```
git commit -m "<комментарий>"
```



Ветка - это набор commit, которые идут друг за другом. У ветки есть название, основную ветку чаще всего называют master (на картинках будет называться main) . Если говорить простыми словами, то ветка master - это наш проект.

Другие ветки - это отдельное место для реализации нового функционала или исправление багов (ошибок) нашего проекта. То есть, с отдельной веткой вы делаете что угодно, а затем сливаете эти изменения в основную ветку master.

Не рекомендую создавать commit напрямую в master . Лучше для этого заводить новую ветку и все изменения писать там.

Для того, чтобы создать новую ветку вводим:

```
git branch <название_ветки>
```

#или вот так

```
git checkout -b <название_ветки>
```

При создании новой ветки, старайтесь называть ее кратким и ёмким именем. Чтобы сразу было понятно, что именно изменялось по проекту. Если вы используете, какую-нибудь систему для ведения задач, то можете в начале названия ветки указывать ID задачи, чтобы можно было легко найти, на основе какой задачи была создана ветка.



Переключаться между ветками можно такой командой:

```
git checkout <название_ветки>
```

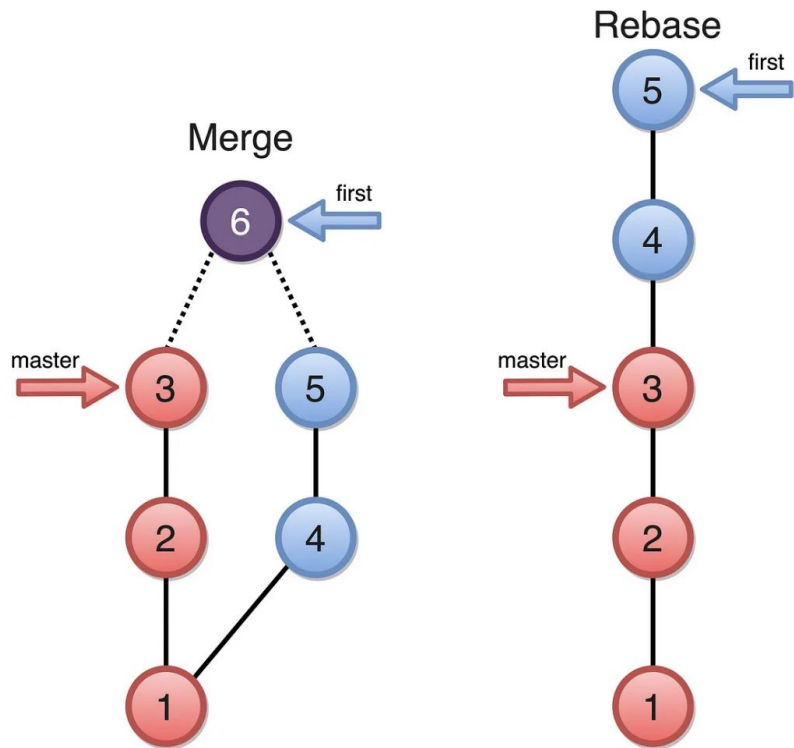
После того, как вы завершили работу над своей задачей, ветку можно слить в master . Для этого нужно переключиться в ветку master и выполнить следующую команду:

```
# Переключаемся в master
git checkout master

# Обновляем локальную ветку с сервера
git pull origin master

# Делаем merge вашей ветки, в ветку в которой вы находитесь
# В данном примере это master
git merge <название_ветки>
```


Rebase (перебазирование) — один из способов в git, позволяющий объединить изменения двух веток. У этого способа есть преимущество перед merge (слияние) — он позволяет переписать историю ветки, придав тот истории тот вид, который нам нужен.





Команда `git cherry-pick` берёт изменения, вносимые одним коммитом, и пытается повторно применить их в виде нового коммита в текущей ветке. Эта возможность полезна в ситуации, когда нужно забрать парочку коммитов из другой ветки, а не сливать ветку целиком со всеми внесенными в нее изменениями.





Команда **git commit --amend** — это удобный способ изменить последний коммит. Она позволяет объединить проиндексированные изменения с предыдущим коммитом без создания нового коммита. Ее можно использовать для редактирования комментария к предыдущему коммиту без изменения состояния кода в нем.

Команда **git squash** — это прием, который помогает взять серию коммитов и уплотнить ее. Например, предположим: у вас есть серия из N коммитов и вы можете путем сжатия преобразовать ее в один-единственный коммит. Сжатие через **git squash** в основном применяется, чтобы превратить большое число малозначимых коммитов в небольшое число значимых. Так становится легче отслеживать историю Git.

Команда **git reset** — это сложный универсальный инструмент для отмены изменений. Она имеет три основные формы вызова, соответствующие аргументам командной строки `--soft`, `--mixed`, `--hard`. Каждый из этих трех аргументов соответствует трем внутренним механизмам управления состоянием Git: дереву коммитов (HEAD), разделу проиндексированных файлов и рабочему каталогу.

Команда **git stash** - эта команда используется для сохранения неподтверждённых изменений в отдельном хранилище, чтобы можно было вернуться к ним позже. Сами файлы возвращаются к исходному состоянию. Команда полезна, когда вы работаете над одной веткой, хотите переключиться на другую, но вы ещё не готовы сделать коммит в текущей ветке. Таким образом, вы прячете изменения в коде, переключаетесь на другую ветку, возвращаетесь к исходной ветке, а затем разархивируете свои изменения.









