



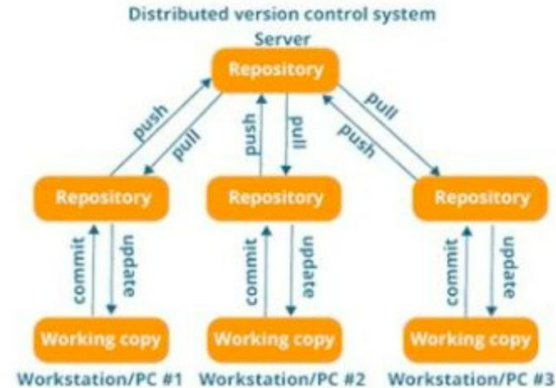
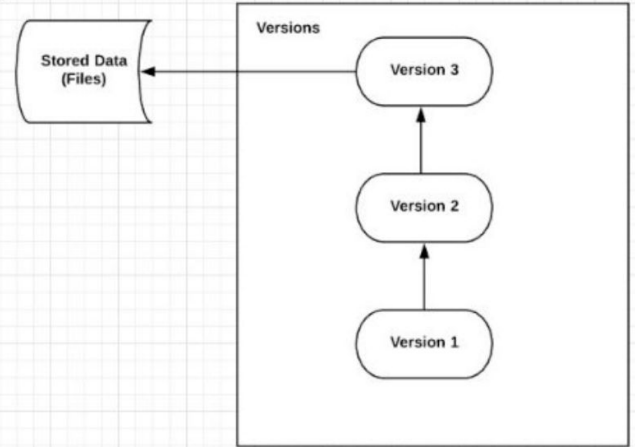
Lesson 37

22.04.2024

Types of Version Control System

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

Local Computer

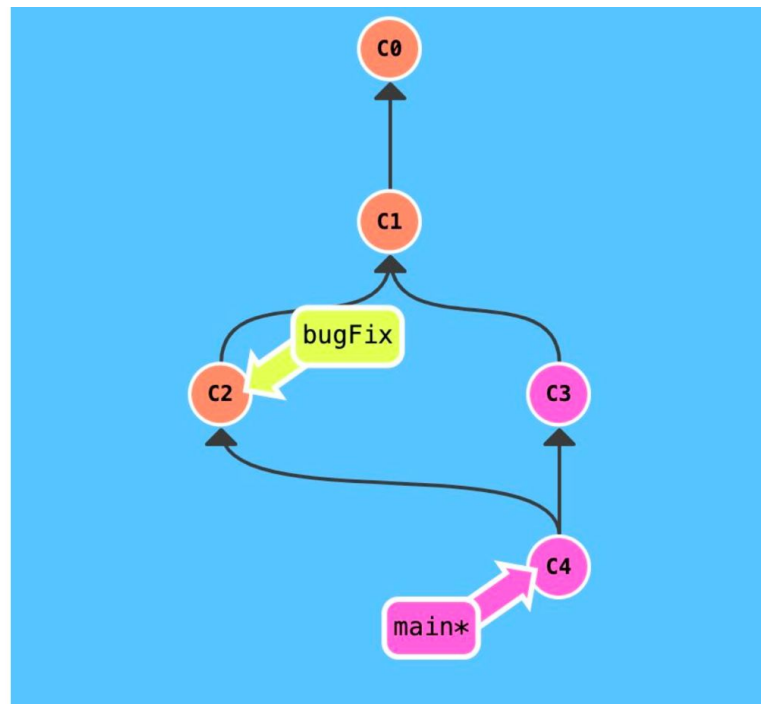
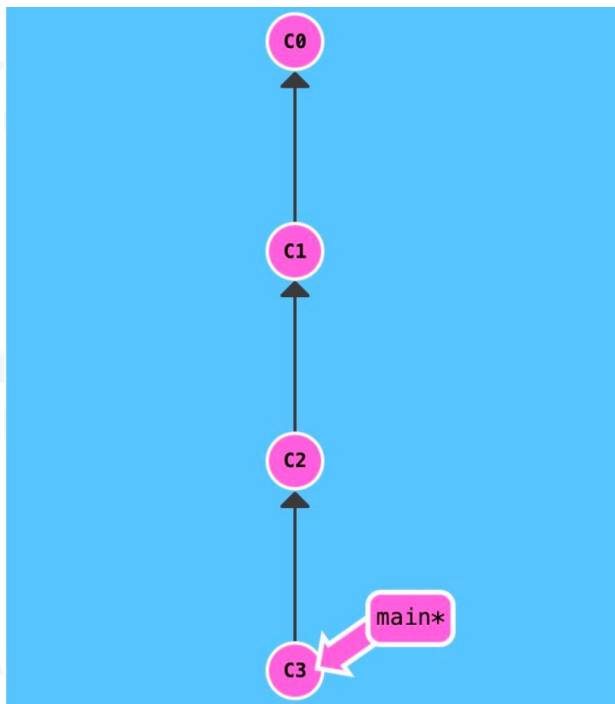


Git – розподілена система керування версіями. Проект був створений Лінусом Торвальдсом для управління розробкою ядра Linux, перша версія випущена 7 квітня 2005 року



Як працює

Якщо подивитися на картинку, то ставати трохи простіше з розумінням. Кожен гурток, це commit. Стрілки показують напрямок, з якого commit зроблений наступний. Наприклад C3 зроблено з C2 і т. д. Всі ці commit знаходяться у гілці під назвою main. Це основна гілка, найчастіше її називають master. Прямокутник main* показує в якому commit ми зараз знаходимося, вказівник.



Налаштування

Ви встановили собі Git та можете ним користуватися. Давайте тепер його налаштуємо, щоб, коли ви створювали commit, вказувався автор, хто його створив.

Відкриваємо термінал (Linux та MacOS) або консоль (Windows) та вводимо наступні команди.

```
#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>"
```



Створення репозиторію

Тепер ви готові працювати з Git локально на комп'ютері.

Створимо наш перший репозиторій. Для цього пройдіть до папки вашого проекту.

```
#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
```



Тепер Git відстежує зміни файлів вашого проекту. Але, тому що ви тільки створили репозиторій у ньому немає коду. Для цього потрібно створити 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>"
```



Гілка – це набір commit, які йдуть один за одним. Гілка має назву, основну гілку найчастіше називають master / main. Якщо говорити простими словами, то гілка master / main – це наш проект. Інші гілки – це окреме місце для реалізації нового функціоналу або виправлення багів (помилки) нашого проекту. Тобто, з окремою гілкою ви робите будь-що, а потім зливаєте ці зміни в основну гілку master / main.

Не рекомендую створювати commit безпосередньо в master / main. Краще для цього заводити нову гілку та всі зміни писати там.

При створенні нової гілки, намагайтеся називати її коротким і ємним ім'ям. Щоб одразу було зрозуміло, що саме змінювалося за проектом. Якщо ви використовуєте якусь систему для ведення завдань, то можете на початку назви гілки вказувати ID завдання, щоб можна було легко знайти, на основі якого завдання було створено гілку.

Для того, щоб створити нову гілку, вводимо:

```
git branch <название_ветки>
```

#или вот так

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




Перемикається між гілками можна такою командою:

```
git checkout <название_ветки>
```

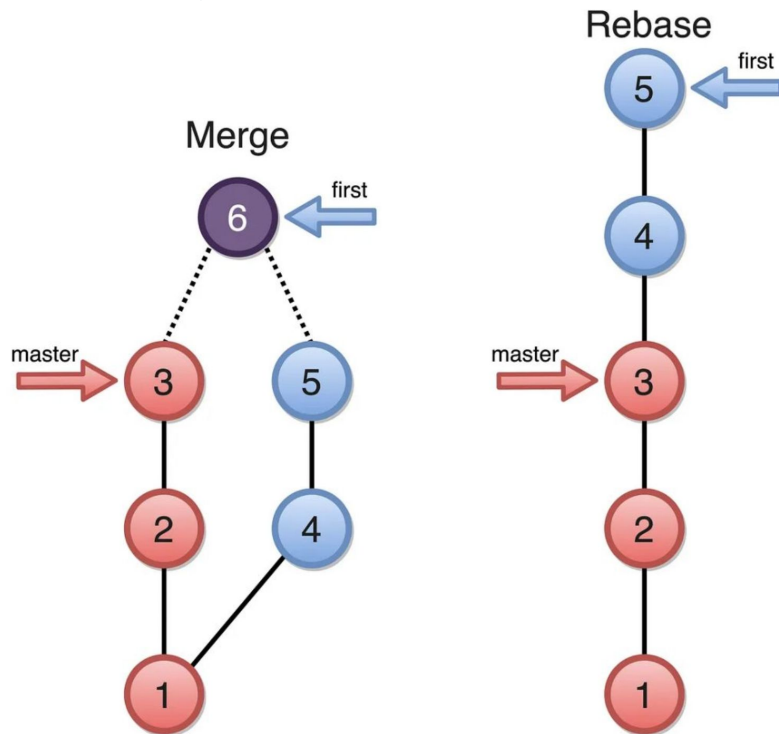
Після того, як ви завершили роботу над своїм завданням, гілку можна злити в master. Для цього потрібно перейти у гілку master і виконати наступну команду:

```
# 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>
```

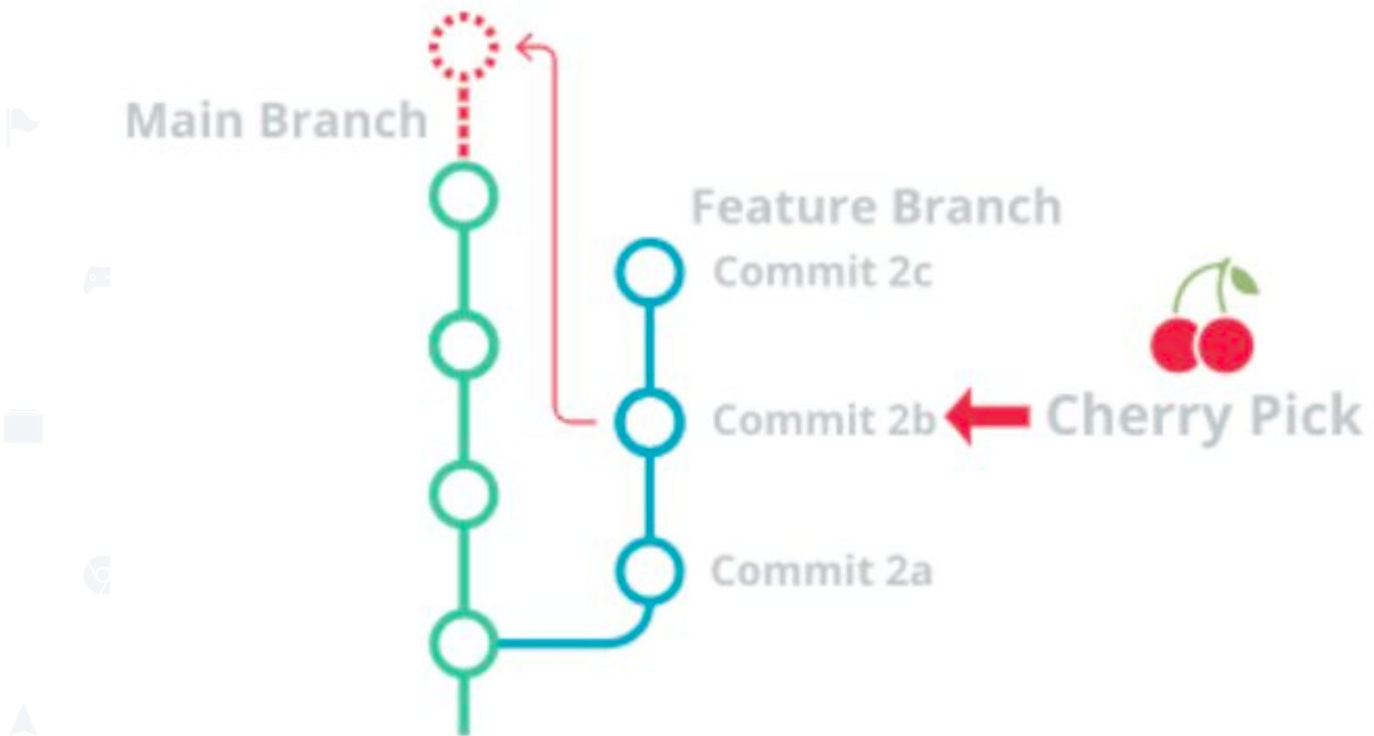



Rebase (перебазування) - один із способів у git, що дозволяє об'єднати зміни двох гілок. У цього способу є перевага перед merge (злиття) - він дозволяє переписати історію гілки, надавши той історії той вид, який нам потрібний.





Команда `git cherry-pick` бере зміни, що вносяться одним коммітом, і намагається повторно застосувати їх у вигляді нового комміту у поточній гілці. Ця можливість корисна в ситуації, коли потрібно забрати парочку коммітів з іншої гілки, а не зливати гілку повністю з усіма внесеними до неї змінами.



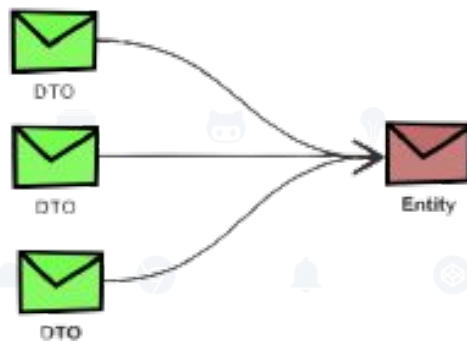
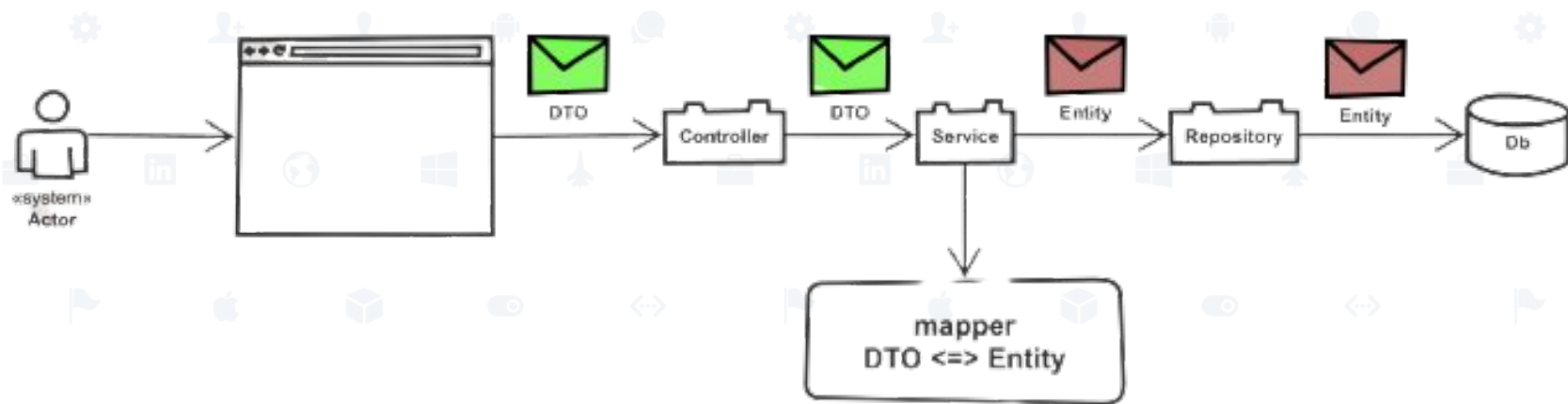


Команда **git commit --amend** - це зручний спосіб змінити останній коміт. Вона дозволяє об'єднати проіндексовані зміни із попереднім коммітом без створення нового комміту. Її можна використовувати для редагування коментаря до попереднього коміту без зміни стану коду в ньому.

Команда **git squash** це прийом, який допомагає взяти серію коммітів і ущільнити її. Наприклад, припустимо: у вас є серія з N коммітів і ви можете шляхом стиснення перетворити її на один-єдиний комміт. Стиснення через git squash в основному застосовується, щоб перетворити велику кількість малозначимих коммітів на невелике число значних. Так легше відстежувати історію Git.

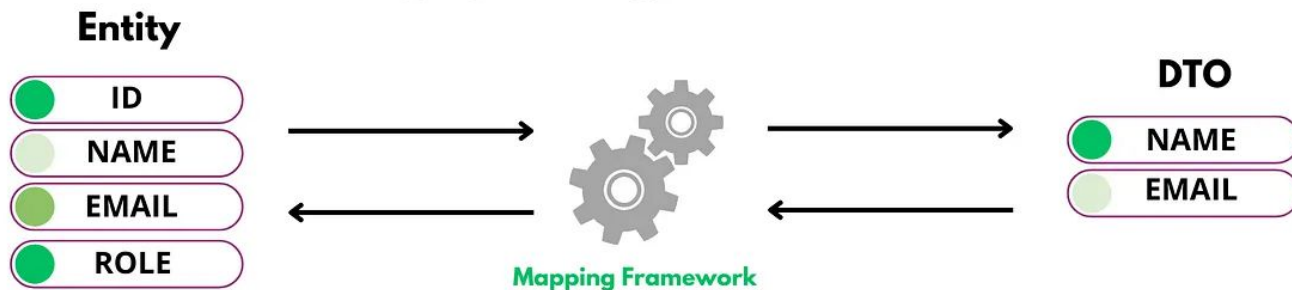
Команда **git reset** – це складний універсальний інструмент для скасування змін. Вона має три основні форми виклику, що відповідають аргументам командного рядка --soft, --mixed, --hard. Кожен із цих трьох аргументів відповідає трьом внутрішнім механізмам керування станом Git: дереву коммітів (HEAD), розділу проіндексованих файлів та робочого каталогу.

Команда **git stash** - ця команда використовується для збереження непідтверджених змін окреме сховище, щоб можна було повернутися до них пізніше. Самі файли повертаються до вихідного стану. Команда корисна, коли ви працюєте над однією гілкою, хочете перейти на іншу, але ви ще не готові зробити коміт у поточній гілці. Таким чином, ви ховаєте зміни в коді, перемикаєтеся на іншу гілку, повертаєтеся до вихідної гілки, а потім розархівуєте свої зміни.





Java Mapping Frameworks



ORIKA

