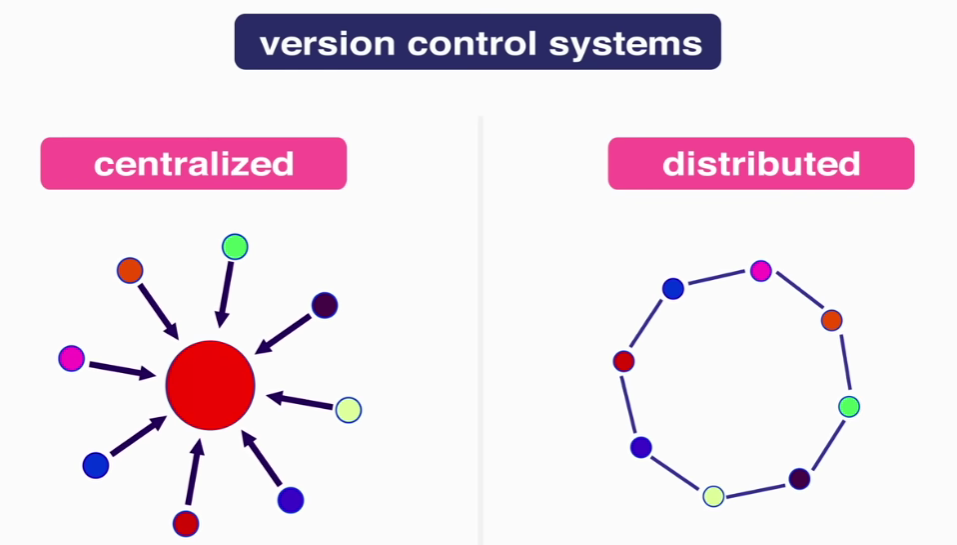
GIT



1. Один локальный сервер и только там есть история проекта, если один меняет чтото то у всех тоже меняется
2. У всех есть версия и история проекта, это и есть ГИТ

# Первоначальная настройка

У гита есть конфиг, Есть утилита прям такая гитКонфиг

И есть свои 3 уровня конфигурации:

1. все юзеры – git config –-system

2. к конкретному пользователю системы – git config --global

3. прям к репозиторию git config –local



# Ls -la

-l – вывести все папки в виде списка

-a – все скрытые файлы и каталоги

# Git log – история изменений

# Staged – подготовленный файл

# Git add

На самом деле сразу комит нельзя,надо сначала сделать add *file/dir*, тогда файлы станут ***staged*** и ихможно будет комиттить

Git commit -m “message”

То есть если ты много чего наделал и хочешь разделить все по коммитам, можно git add только первую часть и потом закомитить, а потом вторые

Git status – file states()

# Vim app.py открывает коммит

Чтобы выйти – esc ->

:x=save exit,

:q -> exit ;

:w -> save

Vim -> чтобы редачить файл можно нажать ‘a’

# Git restore app.cs – откат до последнего коммита

# Git diff – изменения с последнего коммита

# Git diff --staged – изменения которые находтятся в индексе(подготовлены для коммита)

# Git add+commit = git commit -am “comment”

-a = add ALL files

-m = message

# Git mv

Переименовывать или перемещать файлы

Git mv oldname.cs newname.cs

Move:

Git mv oldname.cs /some other dir/newname.cs

# .gitignore

Чтобы начал действовать надо закомитить его

# HEAD

Это указатель на последний коммит, или на место где вызываются команды

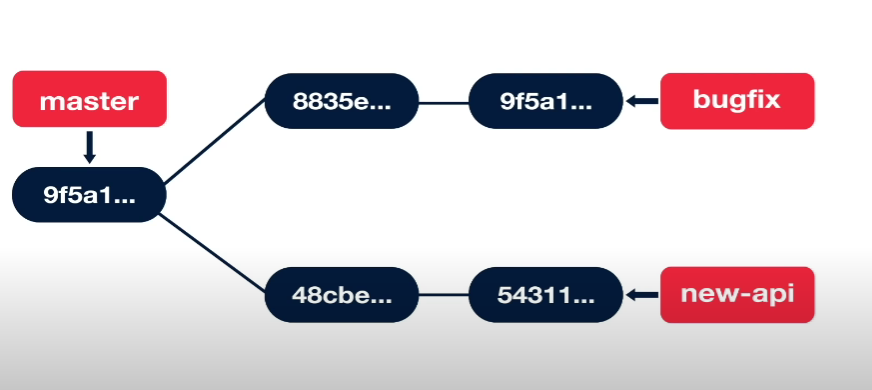
каждый раз когда делаем коммит он смещается на новый

# git log – история коммитов

# git branch newname – создать новую ветку из этого коммита

# git checkout newname – переключиться на ветку

# git merge

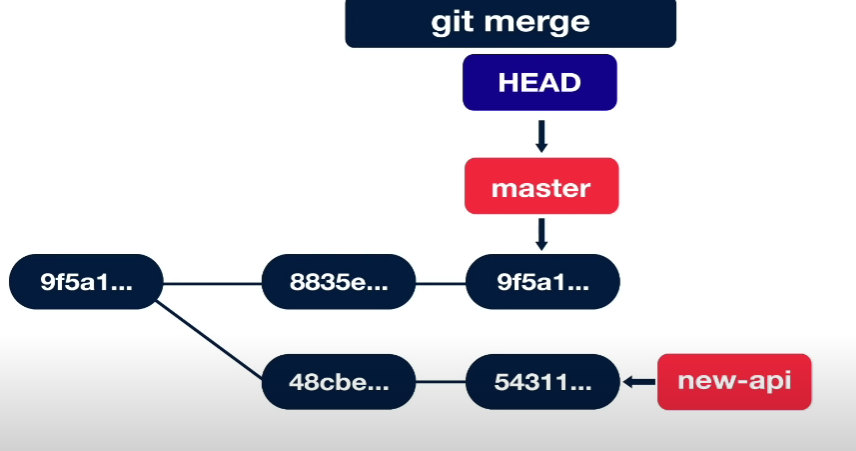
допустим у нас есть: 

и мы хотим все в мастер

переходим в мастер, и пишем *git merge bugfix***,** тампишется что за изменения, и еще пишется что это было **fast-forward**, это значит что мастер был прямым родителем для багфикс и можно просто перенести указатель мастера на последний commit bugfix

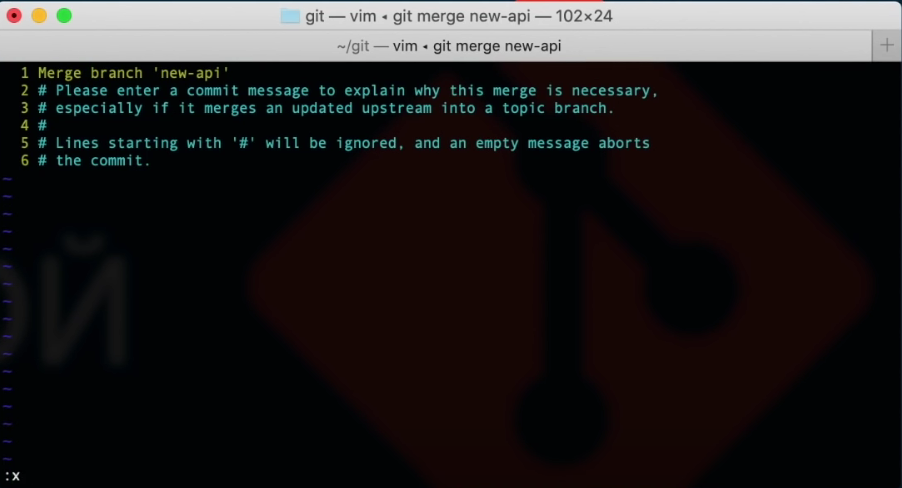
теперь можем удалить bugfix: *git branch -d bugfix*

теперь мы получили такую ситуацию:

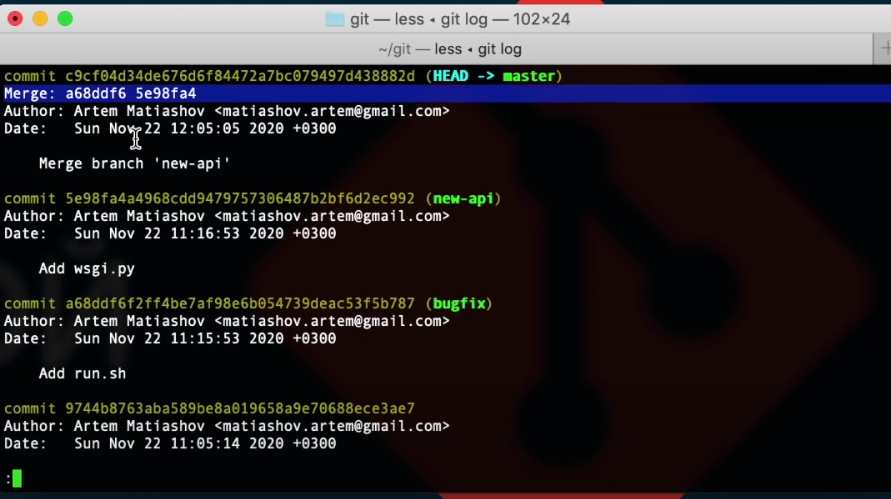


Master не прямой родитель для new-api, пишем *git merge new-api*

Git откроет vim

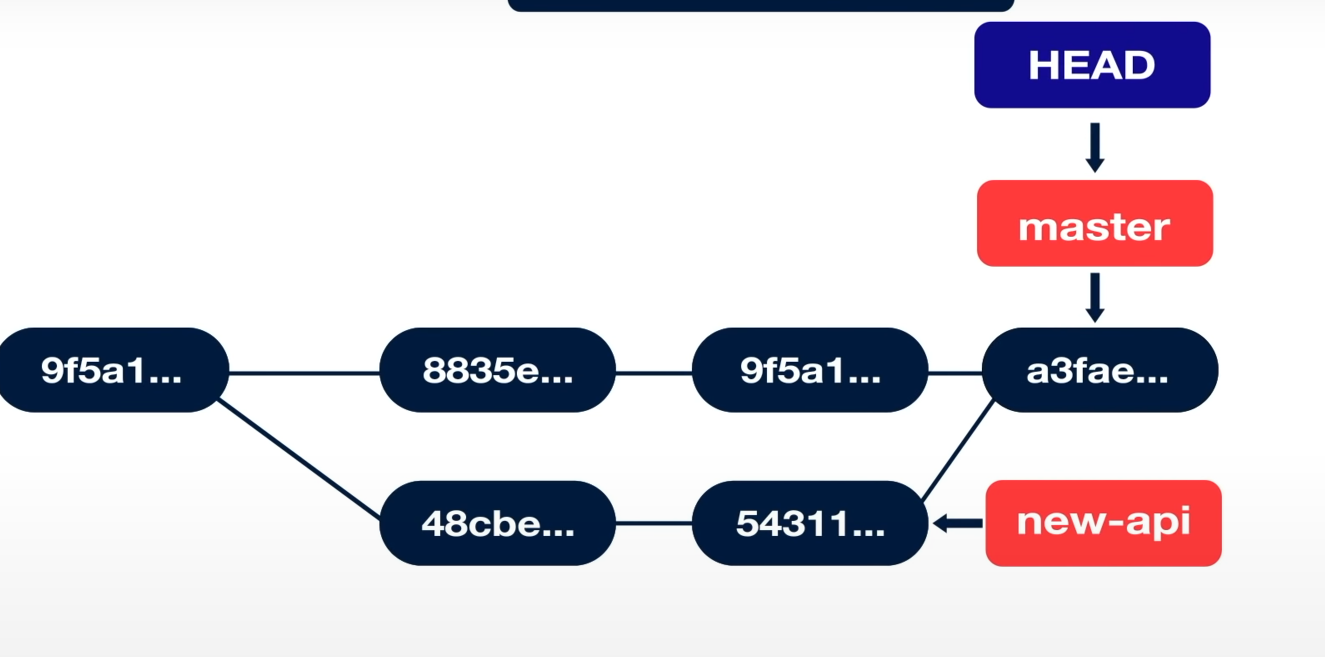


Ну окей сохраняем и нажимаем enter

Посмотрим шо произошло

Вот, выделено, там 2 хэш суммы new-api и bugfix

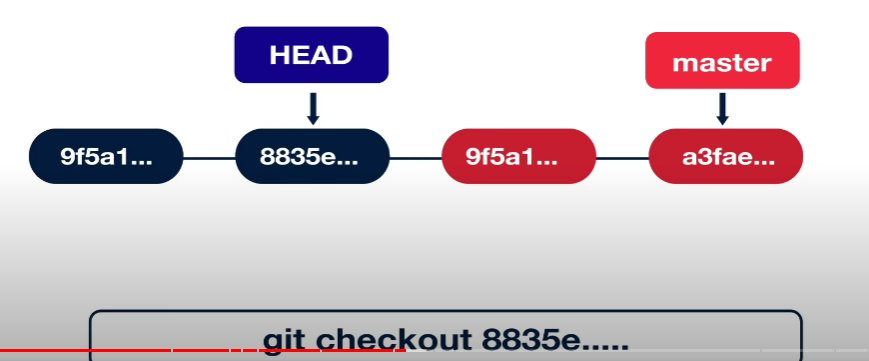
У нас собственно вот так все:



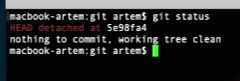
Тут все нормально но Часто бывает

# Merge conflict

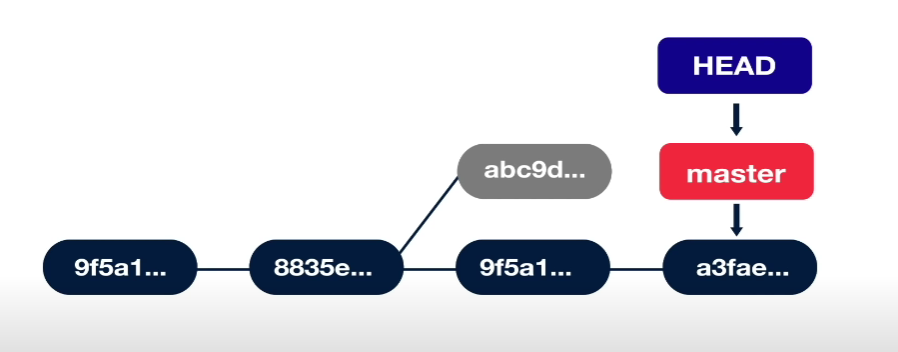
# Git checkout “commit hash”



Будет detached head

 рhead не указывает ни на одну из веток

И если сделать в этом состоянии коммит а потом переключиться на мастер



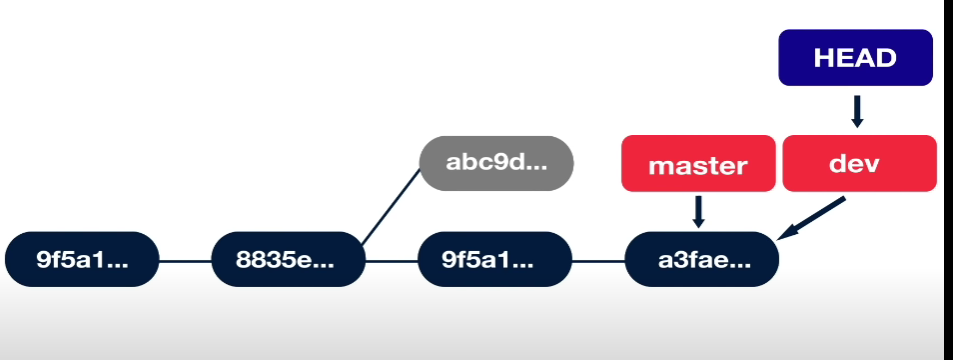
То этот комит потеряется и на него можно будет переключиться только зная хэш сумму

С помощью checkout можно создавать ветки

# Git checkout -b dev

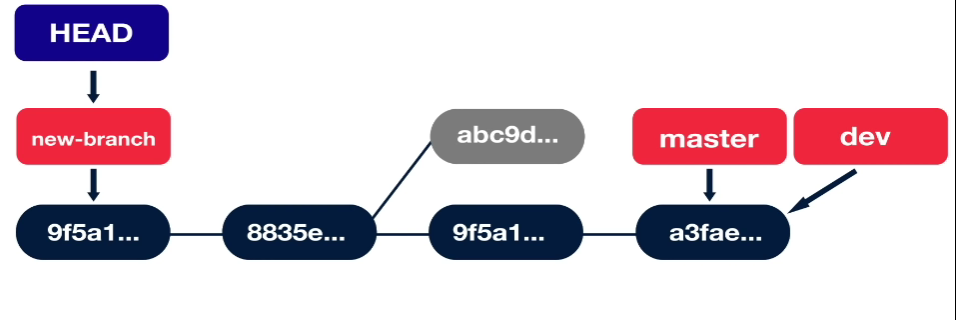


Тперь создастся ветка и хэд туда тоже уйдет



Если мы хотим отвести ветку не от этого, а от другого коммита то можно передать в конце хэш сумму коммита





# Git checkout “filename” – откат файла до последнего коммита

Аналог git reset, но мы всего лишь перемещаем head по истории, ну и файлы на компе соответственно тоже меняем

Мы не меняем указатели веток, они как были на своих коммитах, так и находятся там

# Git reset

Чек можно еще юзать

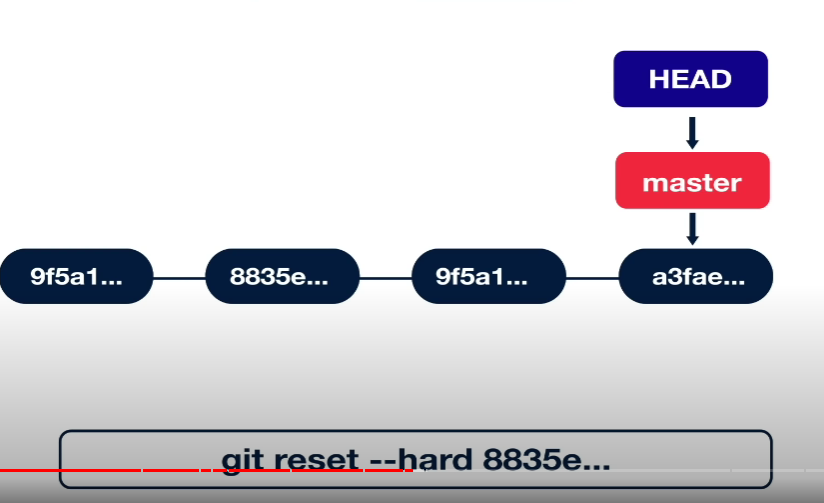
Git checkout “filename” для одного файла

Чтобы изменить из stage – git reset filename

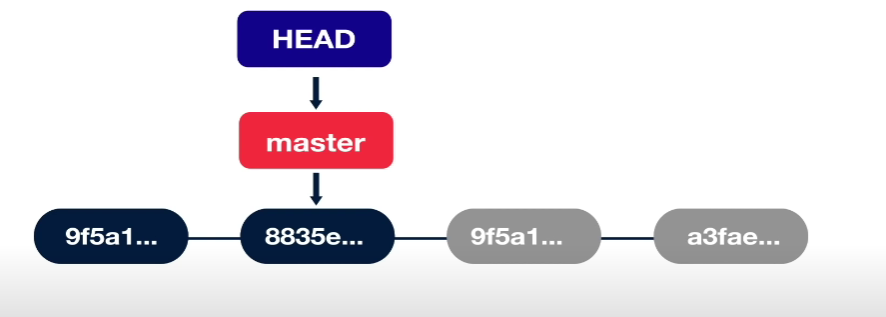
-----

Изменить указатели веток

--hard будет



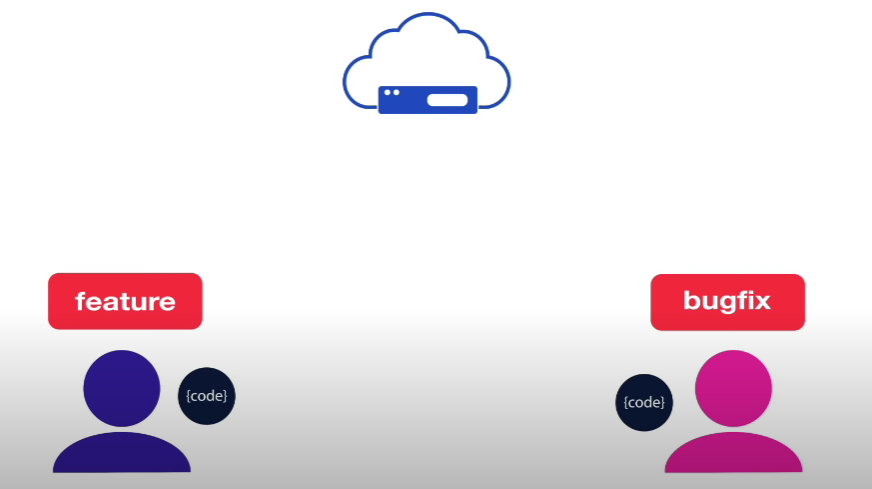
Станет



Белые не удалятся, просто не будет видно в истории, по хэшу можем прыгнуть туда и все восстановится

# Работа с удаленным репо

Допустим есть 2 типа и они забрали клон и сделали свою ветку



Потом один пушит, но один может смержить свою ветку с мастером и запушить на сервер

То у второго будет ветка мастер без коммитов второго

Чтобы получить коммиты второго разраба он может выполнить

## Git pull

И у них будут 2 одинаковые ветки мастера

# Git remote

Git remote -управление удаленные сервы, переименовывать, удалять

## Git remote add

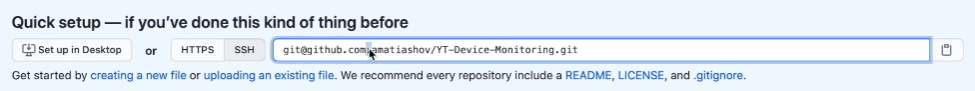
Пример:

У нас есть комит на компе



Мы сделали репо на гите и пишем

# Git remote add origin “ссылка ssh репо гита” //mb https



Теперь шо у нас есть:

Пишем

## Git remote -v



У нас есть один репо для чтения fetch и один для отправки push(локальный)

Мы хотим отправить тот что у нас есть на сервер, поэто му пушимм

## Git push origin master

Origin – server

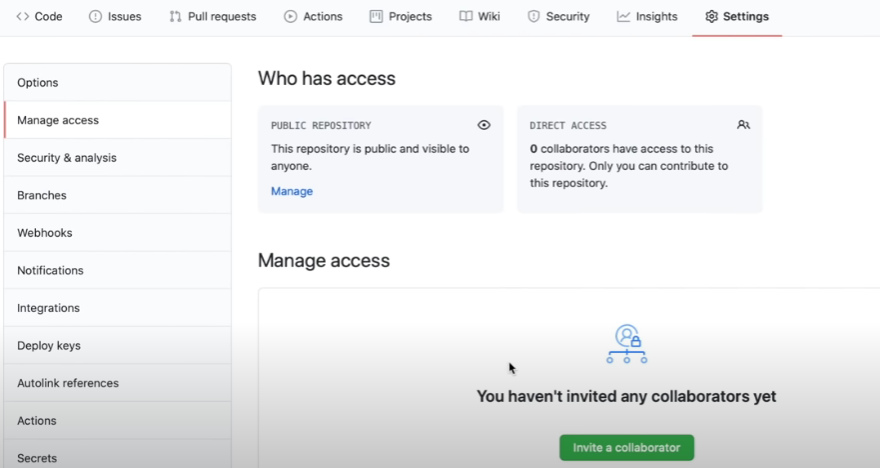
Master – в мастер на сервере

Из ветки где я сейчас нахожусь\

Теперь мы передали наш код на сервер

# private repo

Чтобы дать доступ другим типам к привату мы идем:



Invitе’им и усе

Потом другой чел чтото может менять ,

Он допустим делает git push origin master

Теперь как другому разрабу получить правки ?

# Делаем git pull origin master

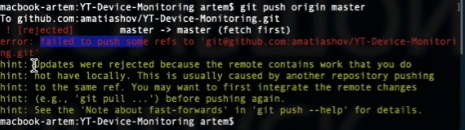
Она забирает изменения с удаленного сервера

И все как бы последовательно, а если параллельно работают, то могут одновременно и запушить правки на сервер:

У одного свои коммиты у другого другие

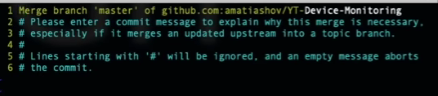
2й допустим сделал пуш, и второй тоже пытается, причем без get pull

Будет ошибка



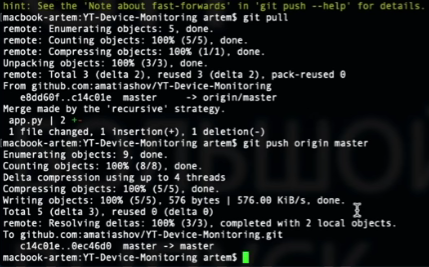
Запушить не получится, там желтым подсказка что отклонено тушта там уже другой мастер и надо сначала сделать git pull

Делаем git pull

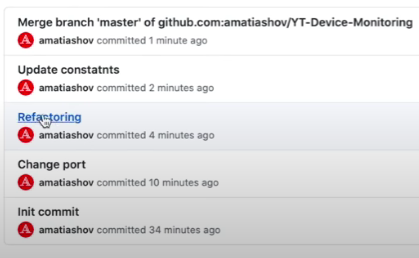


Git открыл vim чтобы сделать merge коммит, окей, пишем :х, выходим

Пишем git push origin master



Push прошел, там теперь 5 коммитов



Сверху

# merge commit

…. Отступление шобы понять как это работает

У нас есть репо и мы делаем remote add



И потом git push origin master



Теперь у нас появляетсся локальный указатель remote ref



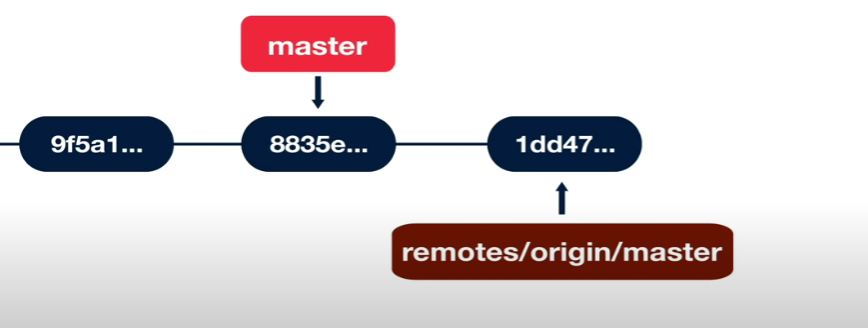
У нас потом меняется история коммитов и как он передвигается?

Чтобы получить актуалочку с удаленных веток мы можем сделать

# git fetch

если несколько удал репо то имя можно передать git fetch origin <- name

произойдет следующая штука:



Теперь чтобы получить то что получили удаленно, надо сделать

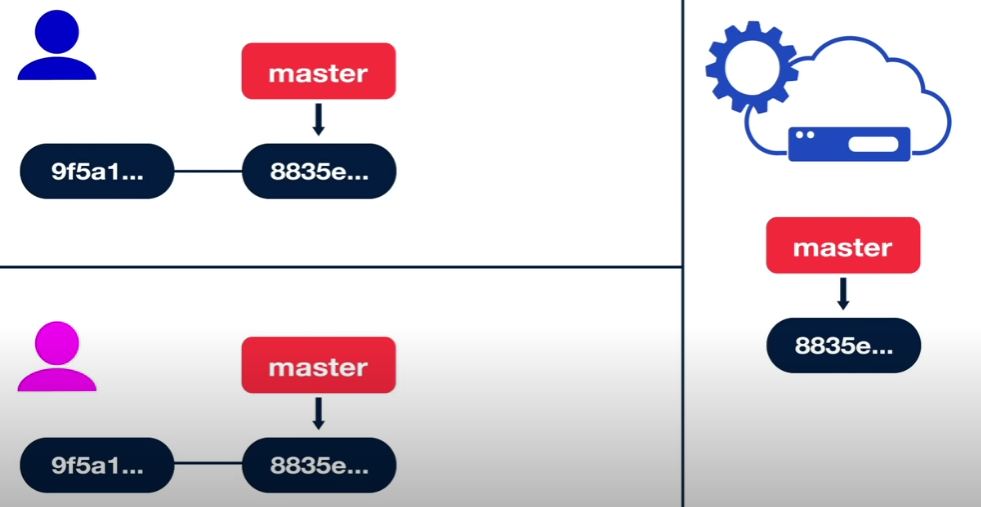
# Git merge origin/master

своей ветки с новой

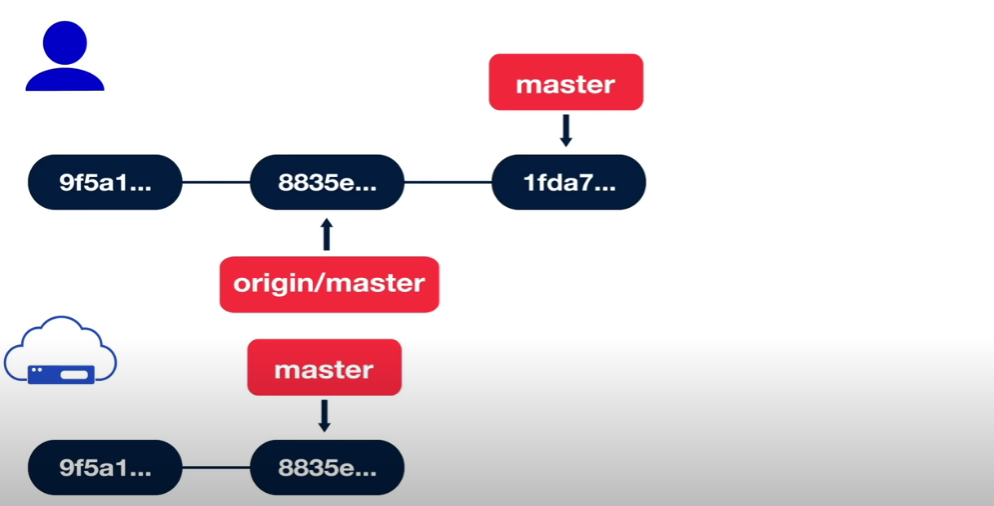
так вот git pull заменяет git fetch + git merge

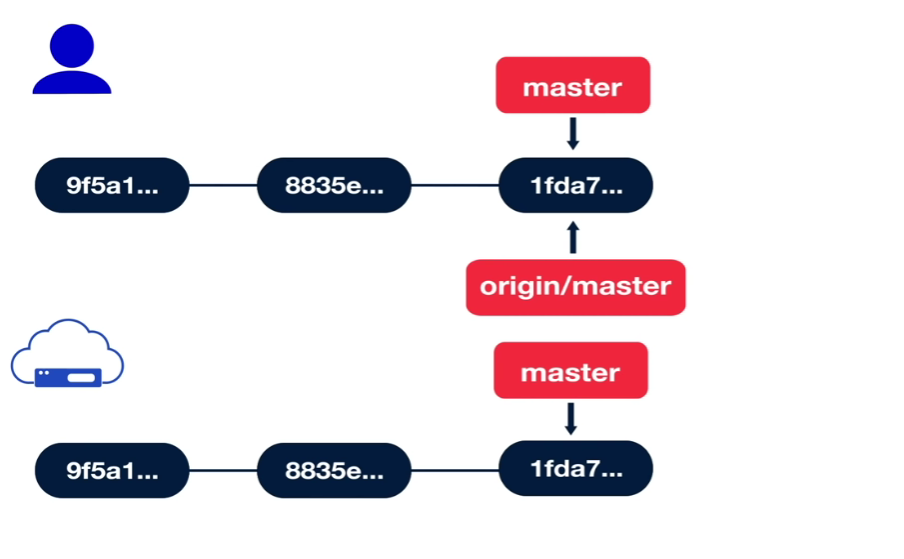
# что произошло с т.зрения 2х челов:

были 2 типа и серв на одном коммите



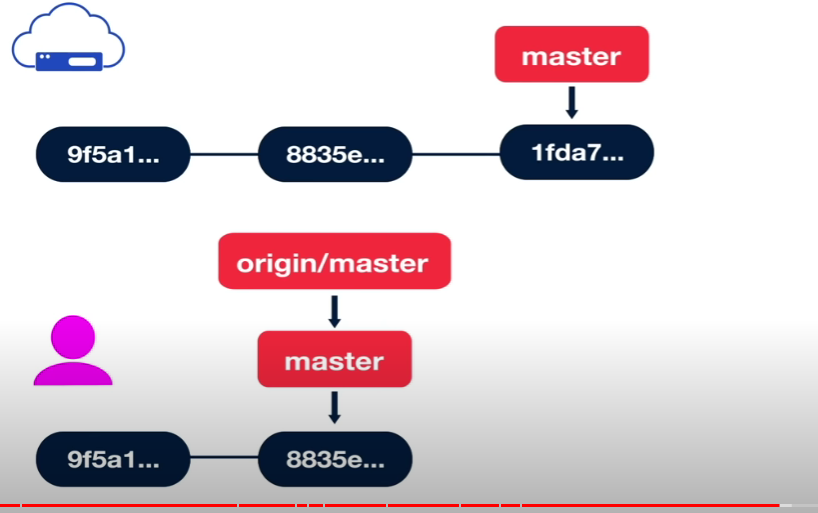
Первый делает коммит

 и после этого пушит

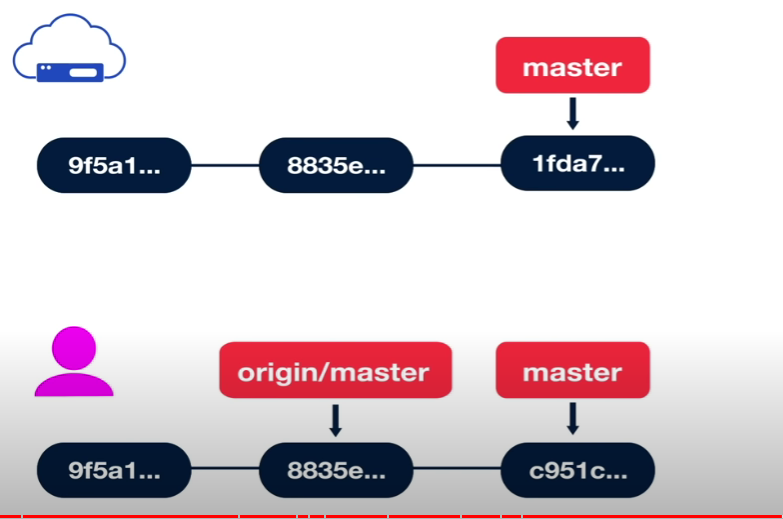


Серв принял правки разраба фаст форвардом

Переходим ко второму разрабу который ничо не знает об этом,



Он делает свой коммит

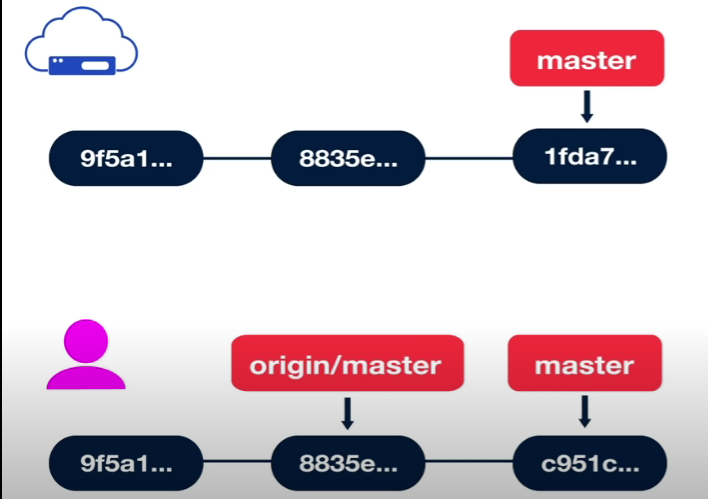


И потом гит пуш, серв откланяет

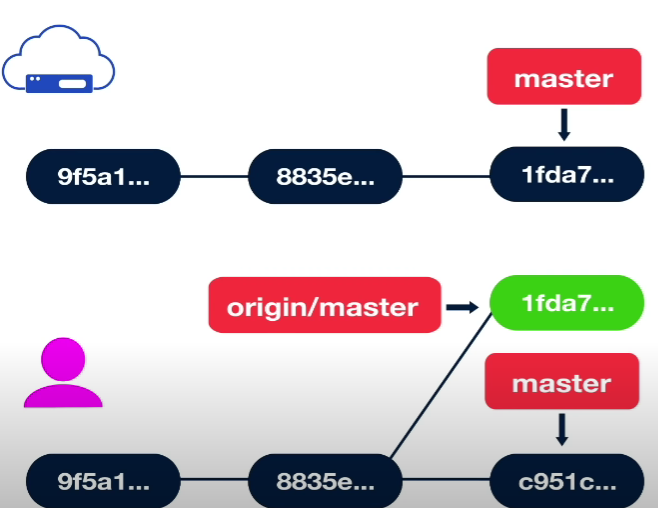
Делает пулл гит предлагает merge commit

# Если разложить pull в этот момент было:

Было:



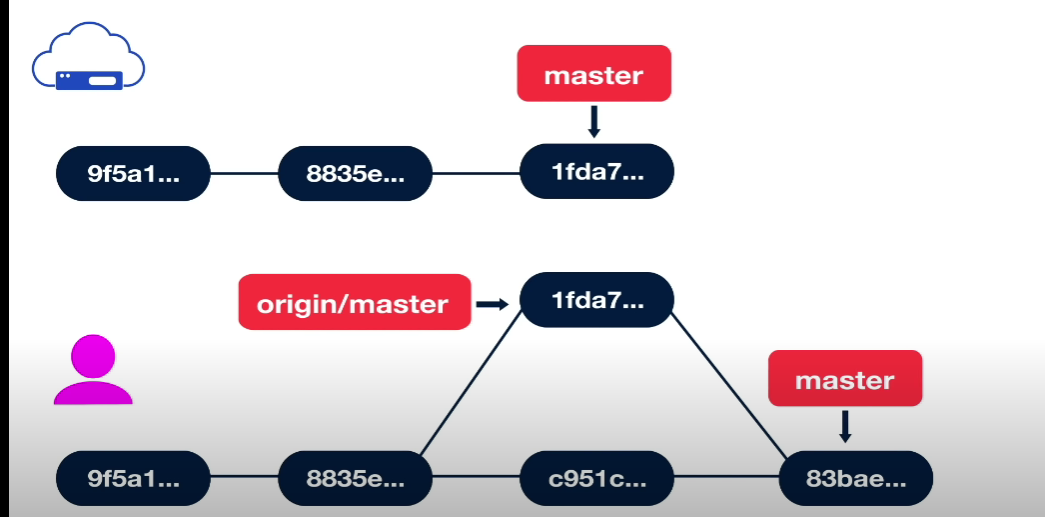
1. Git fetch



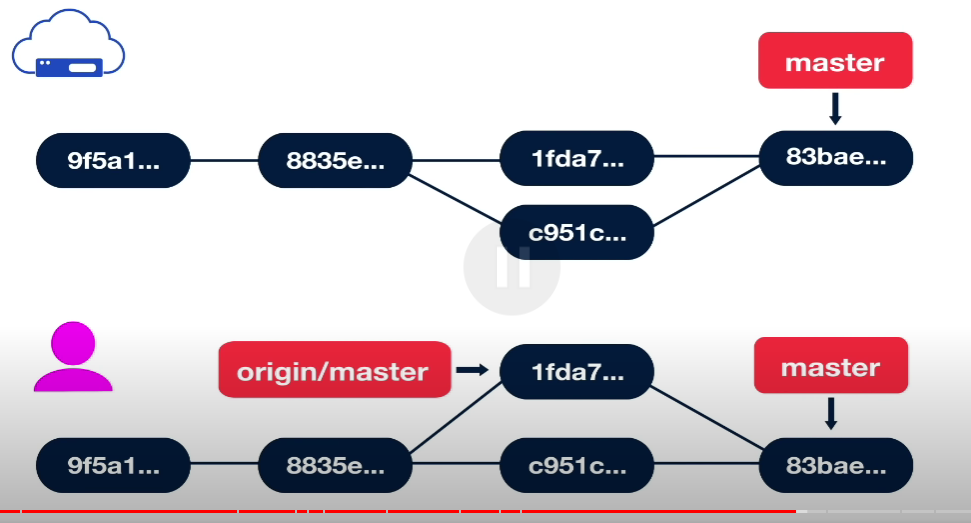
Зеленый, - коммит 1го разраба

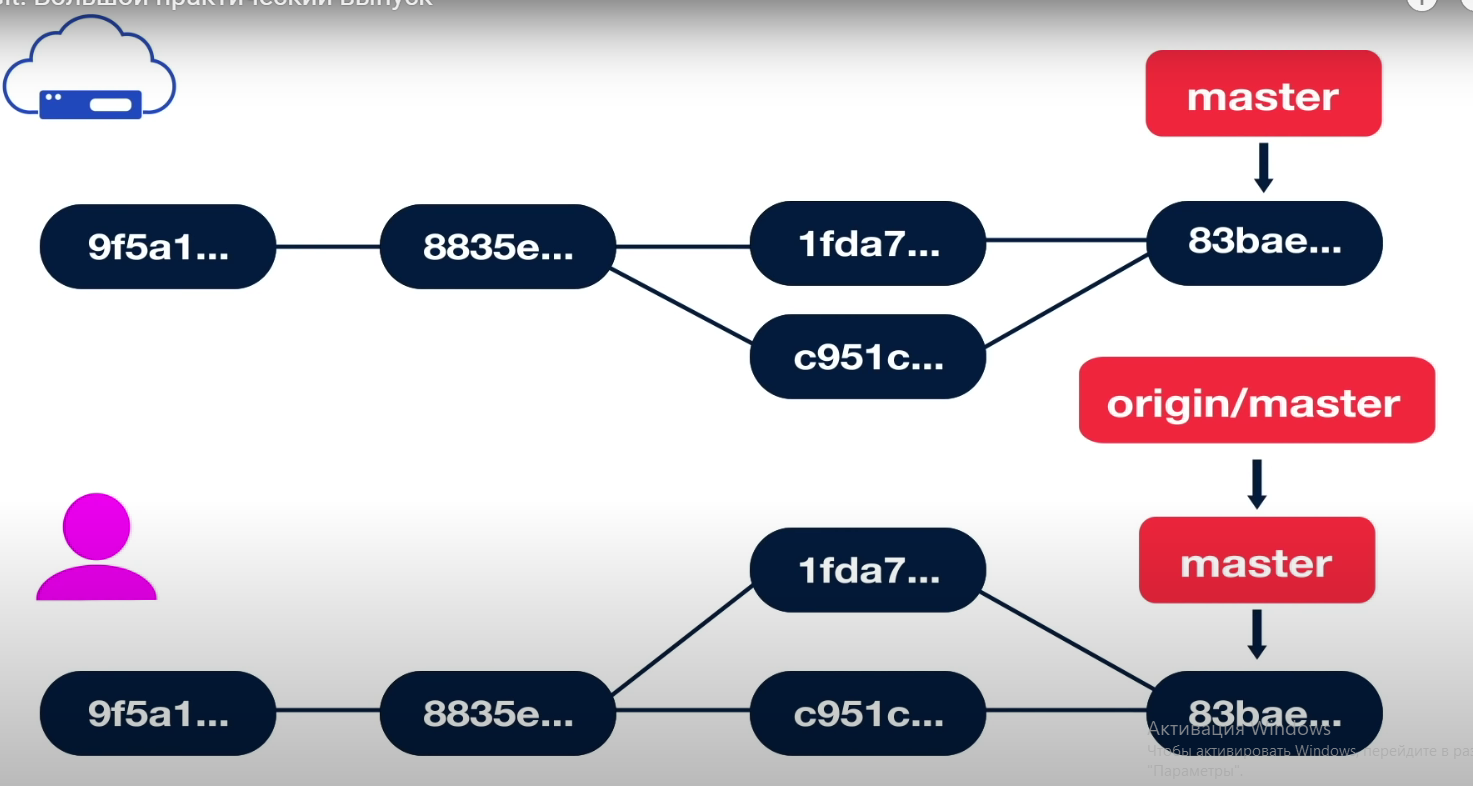
1. Git merge

Мы не можем передвинуть указатель ветки вперед, поэтому гит делает merge commit,



После этого чел делает git push и серв принимает тушта он может просто передвинуть указатель ветки вперед





Теперь гит может передвинуть указатель просто вперед

Кстати Чтобы каждый раз не писать git push origin master мы можем прописать 1 раз

# Git push --set-upstream origin master

Мы свяжем локальную ветку мастер с удаленной веткой origin

Мы можем это прописать и теперь можем просто git push

# Если ктото сделает какую то отдельную ветку dev2

И мы хотим подключиться к этой ветке

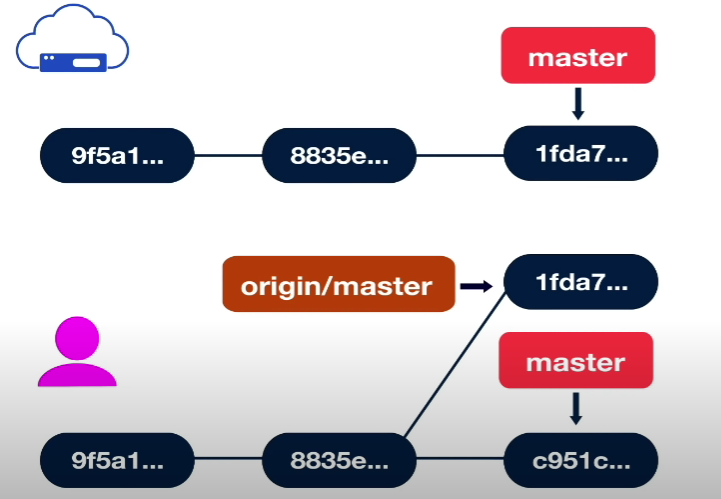
Когда мы делаем git pull гит запомнит что она есть но у нас не создать

Но Мы можем переключиться на нее git checkout dev2

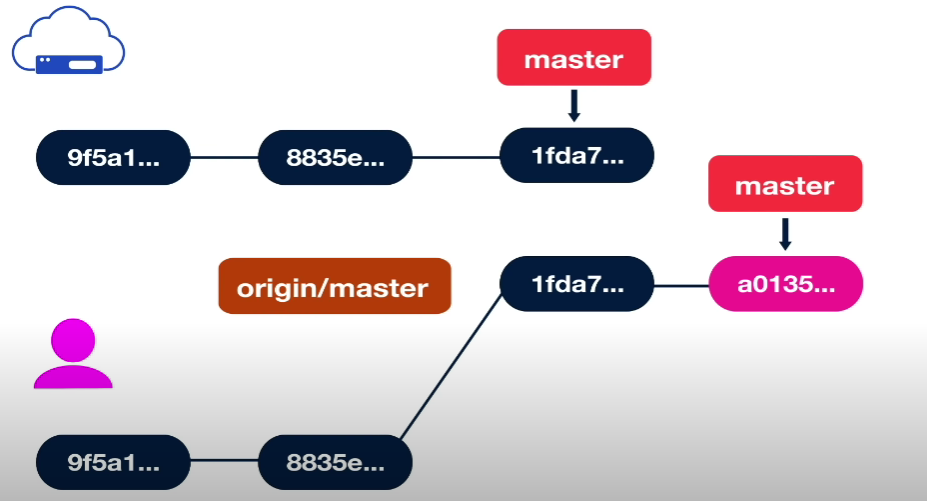
# Git rebase

Короче ветвление неприятно, поэтому мы можем сделать так:

У нас было после гит фетч(получили актуалочку ):

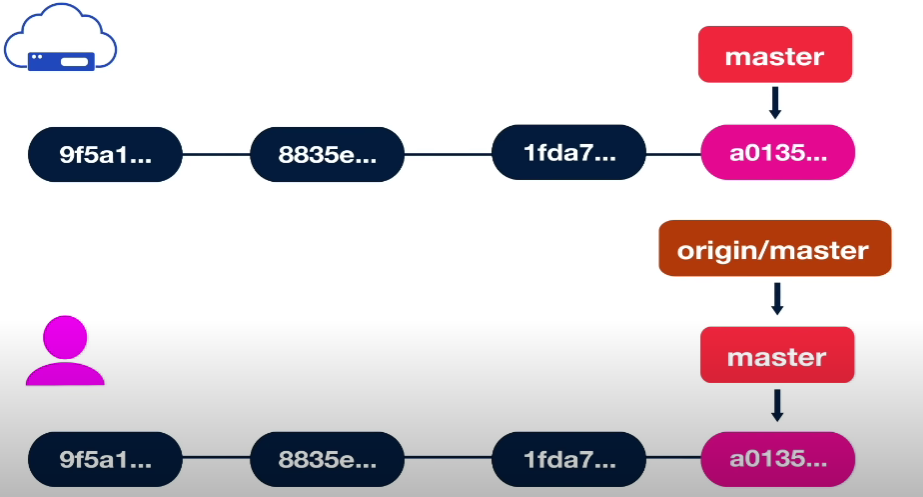


И мы хотим оторвать нижний коммит и приклеить его после актуалочки



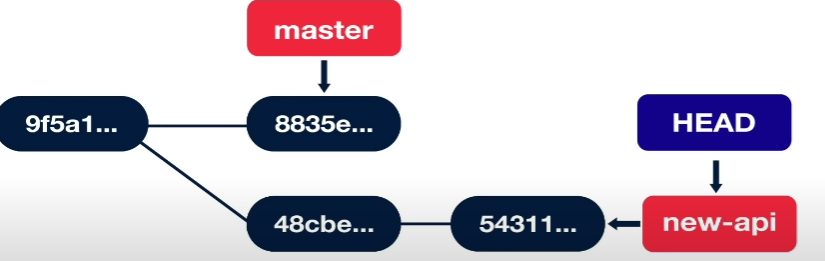
У него поменяется хэш конечно, тушта как минимум родитель другой.

Терь мы можем спокойно пушить это дело на сервер:

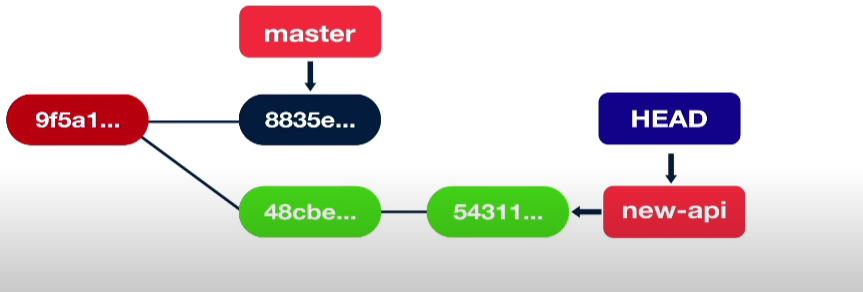


## Это можно сделать через git rebase

Допустим у нас есть 2 ветки, мастер и new api

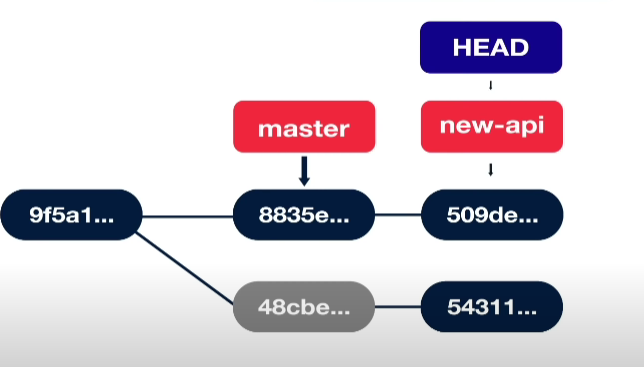


Находясь на new-api делаем rebase master, гит найдет общий коммит для обоих веток(красные)

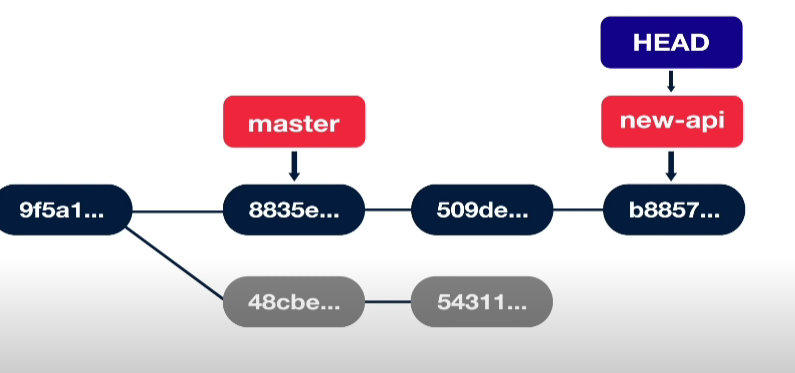
И коммиты которых не хватает в мастере(зеленые)

Он их начнет перемещать по очереди в мастер

Перемещает первый…. Сразу с хэд и new-api



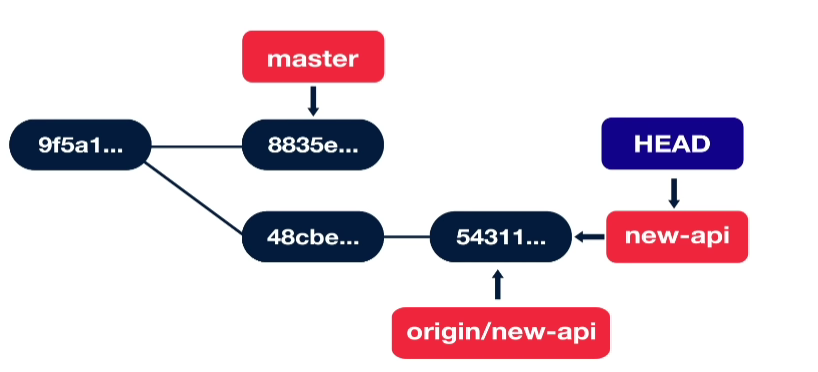
И потом следующий



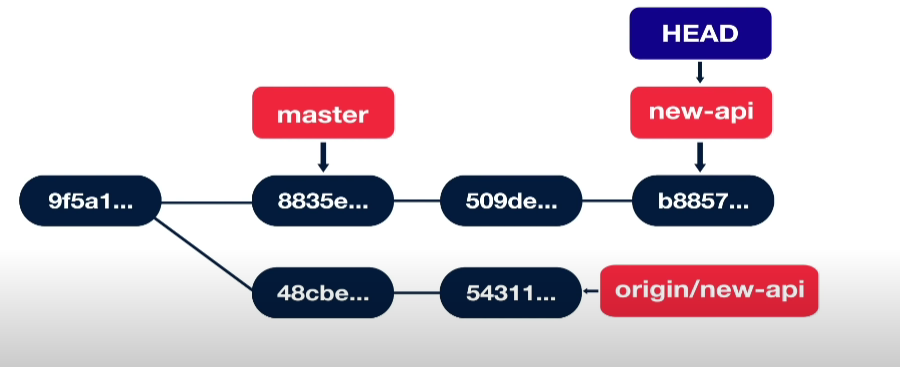
Серые коммиты не удалены, зная хэш можно прыгнуть

# У rebase есть и недостатки

Допустим мы работаем на своей ветке и запушили свой коммит на сервер, то есть там появился указатель origin/new-api



Делаем rebase, получаем

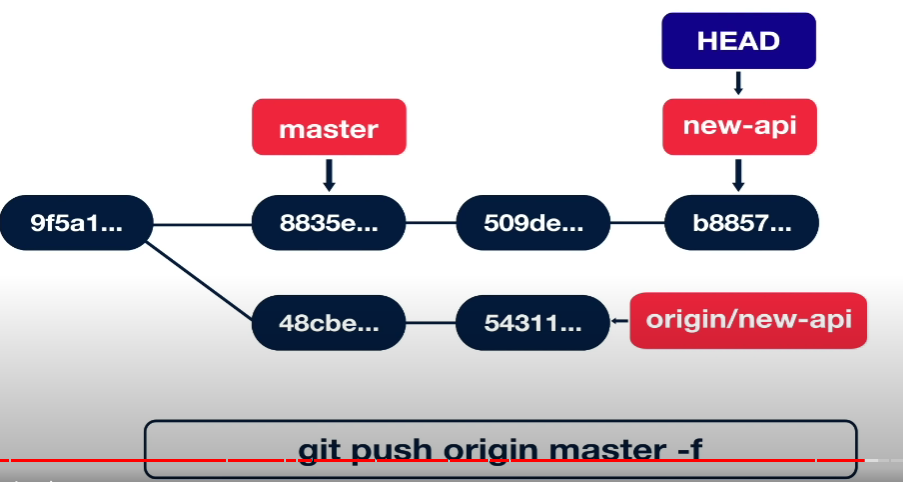


Если сделать git push то будет фэйл, ведь у него указатель стоит на 54311…

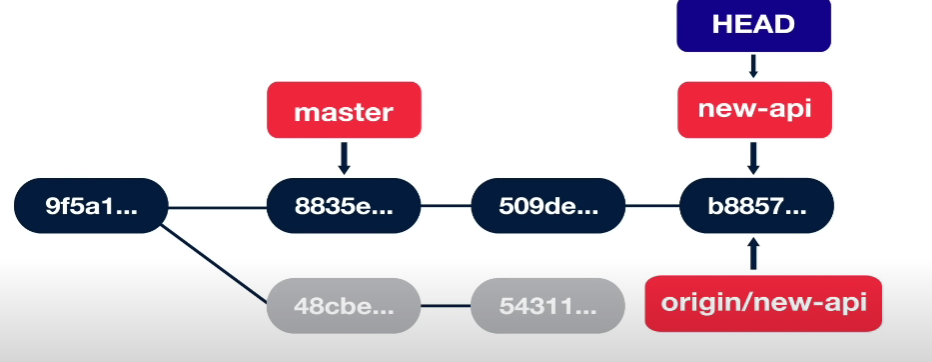
И приняв правки он не сможет перенести вперед

Можно сделать:

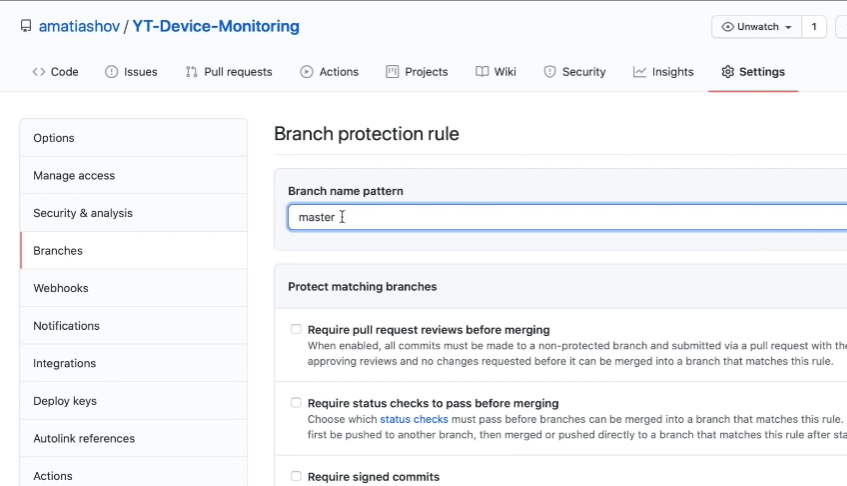
Git push origin master -f



И сервер перешет историю в этой ветке, и она будет повторять вашу ветку



Шобы джуны таким не занимаьться можно задавать правила для веток гита



Теперь нельзя

# Git tag

В гите можно ставить метки на коммиты

Tag -> легковесные(просто указатель с именем)

-> натированные – полноценные объекты с кучей инфы о коммите

Простое: git tag v1

Native: git tag -a NAMETAG -m “message”

Watch tags: git tag

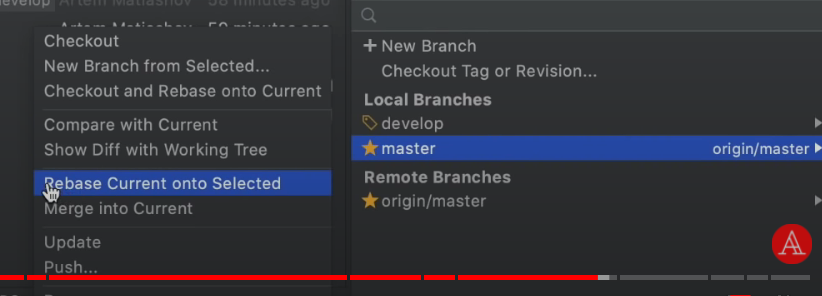
Чтобы отправить тэги:

Git push –tags -> ALL

Git push origin TAGNAME ->1 tag

# Rebase in ide

We on develop



# Merge conflicts

Branches: master + develop on same commit,

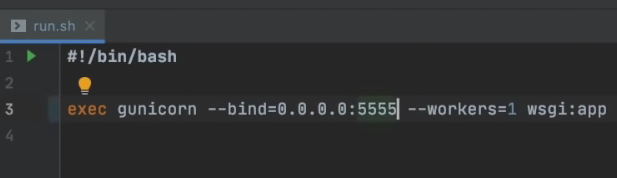
**На мастере меняем файл**



Делаем коммит

**На develop**

Меняем на другой

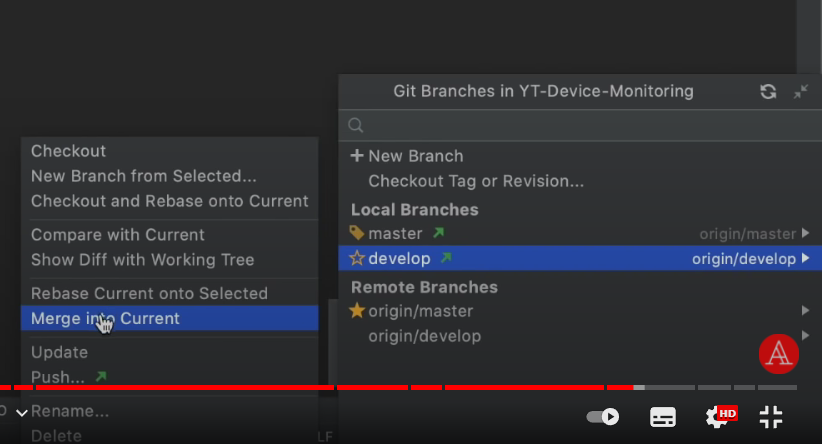


Тоже коммичу

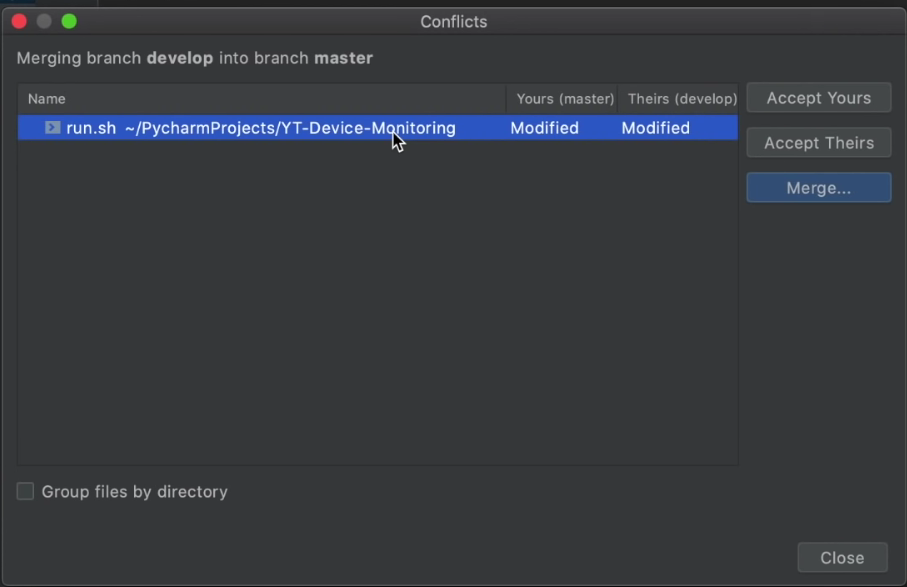
Теперь хотим мерж девелоп и мастер

**На мастере:**

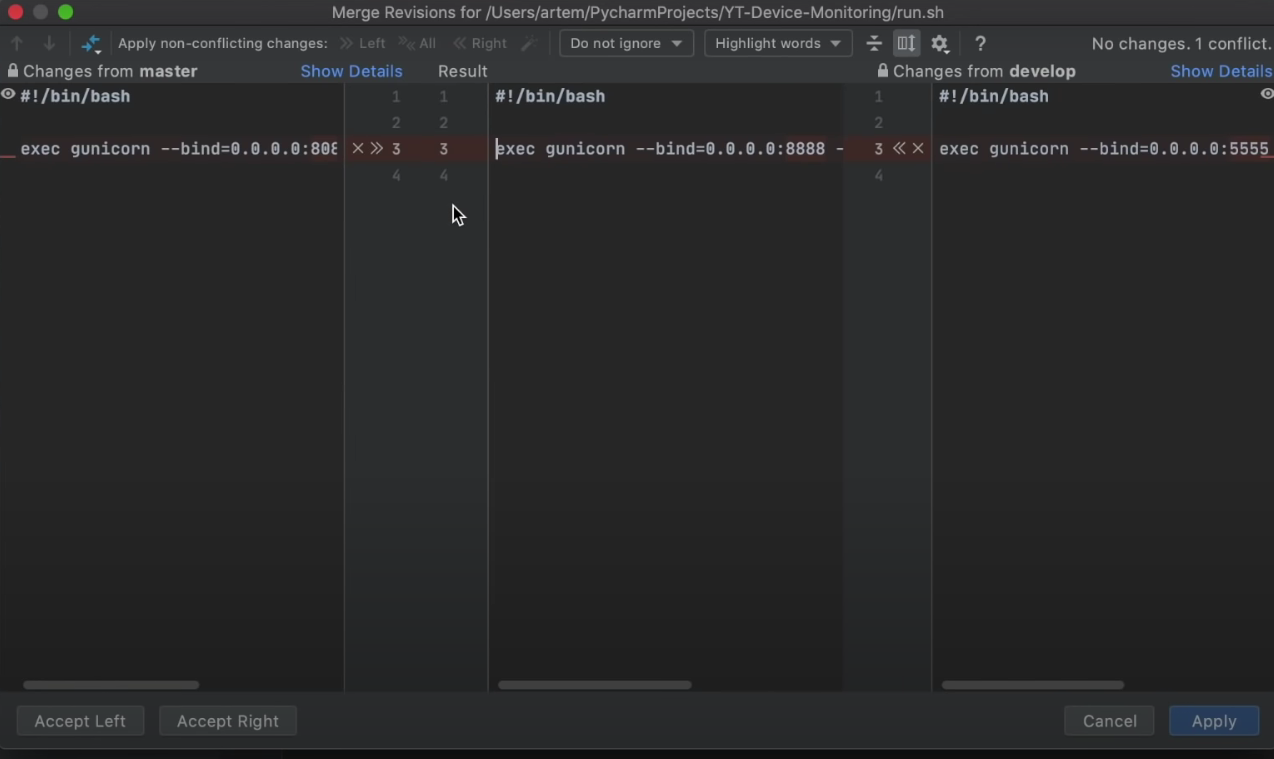
Merge develop in master



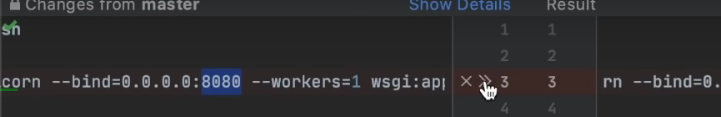
Jоткрывается



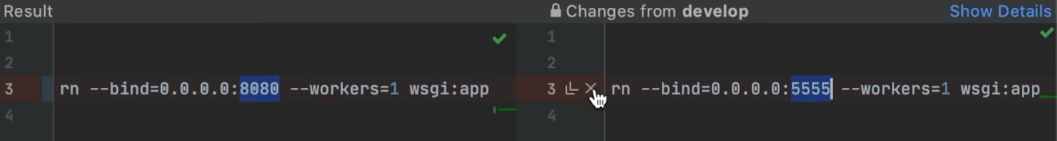
Jоткрываем



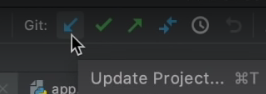
Мы можем взять из какой нибудь:



Можем отклонить какие то



# git pull in ide



# Ide colors

Green – file in index

Red – untracked

After commit = no color

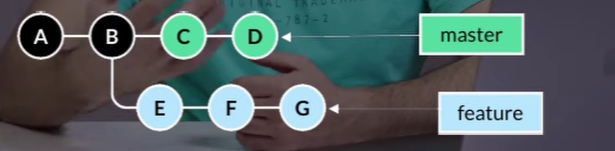
Blue - edited

# Cherry pick

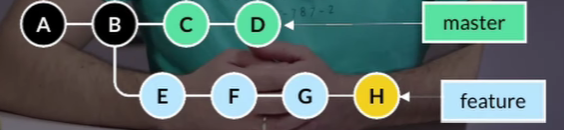
## Когда нужно юзать

1. Hot-fix

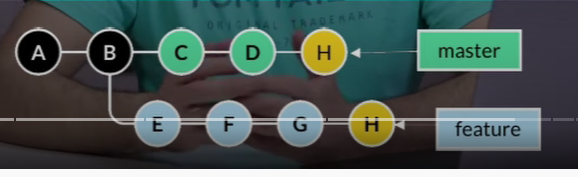
Делаем фичу и часть уже в продакшене



Мы делаем фикс



И мы можем его чисто кинуть на продакшн



1. Коммандная разработка

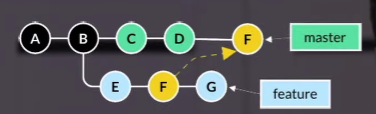
Фронт и бэк в разных брэнчах

Например бэк сделал фичу в апи и во фронте мы хотим юзать ее

1. Когда работаем над веткой и она заброшена, но нам надо некотерые коммиты

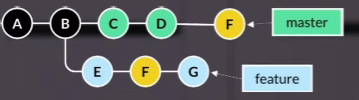
## Юзадж

Мы хотим кинуть в мастер F, нужен хэш



Мы идем в мастер и делаем git cherry-pick f-hash

Стало:



## -edit

Изменить коммит месадж

## --no-commit

Берем из feature изменения и помещает в workin dir

# Git stash

**Это как спрятать изменения как бы в карман**

Мы Допустим сделали изменения какие то, и нам говорят, давай другое делай, отвлекись от ветки

Но на коммит это не тянет все, мы не хотим коммит делать

* Мы делаем git stash

Тогда в git status все будет чисто и мы можем спокойно в другую ветку переключаться

**Также это можно делать когда мы хотим закомиттить в другую ветку:**

И мы хотим проверить что на другой ветке все будет работать ок

* Git stash apply – восстанавливает последний стэш
* Git stash pop stash@{1}- восстанавливает 1 стэш + удалить стэш
* git stash clear - очистить список стешей