-------------https://www.youtube.com/playlist?list=PLDyvV36pndZFHXjXuwA\_NywNrVQO0aQqb

**sudo apt install git** - установить Git

**git --version** - версия Git

**git config --global init.defaultBranch main** - установить имя основной ветви по умолчанию "main" при создании Git репозитория

**git branch -m newName** - переименовать текущую ветку (move)

**git init** - создать Git репозиторий в текущей директории

**git init -b branchName** - создать Git репозиторий с заданным именем ветки

Уровни настройки Git:

**--system** - системные (etc/gitconfig) - изначально отсутствует, директория системная (sudo)

**--global** - глобальные (на уровне текущего пользователя системы /home/anton/.gitconfig)

**--local** - локальные (на уровне проэкта .git/config)

**git config --list** - перечень настроек всех уровней

**git config --list --system**

**git config --list --global**

**git config --list --local**

**git config --settingLevel setting.name "value"** - установить настройку уровня

**git config --global user.name "Anton Yurchenco"**

**git config --global user.email satnetuser001@gmail.com**

**git config user.name "Anton Yurchenco"** - без приставки = локально

**git config user.email satnetuser001@gmail.com**

**git config --settingLevel --unset setting.name** - удалить настройку уровня

**git config --global --unset user.name**

**git config --unset user.name** - без приставки = локально

**git config merge.ff false** - запретить слияние fast-forvard по умолчанию

Настройки можно посмотреть\отредактировать через текстовый редактор (видно наименование секций настроек [core], [user]), из папки проекта это можно сделать:

**subl /etc/gitconfig** - посмотреть в sublime файл с системными настройками

**subl ~/.gitconfig** или **subl /home/anton/.gitconfig**- посмотреть в sublime файл с глобальными настройками

**subl .git/config** - посмотреть в sublime файл с локальными настройками

**git config --settingLevel --remove-section sectionName** - удалить секцию уровня

**git config --global --remove-section user**

**git config --remove-section user**

**git config --global core.editor "subl -n -w"** - текстовый редактор для Git по умолчанию:

https://docs.github.com/en/get-started/getting-started-with-git/associating-text-editors-with-git

**git config --global commit.verbose true** - перед коммитом в редакторе сообщения будет видно изменения Index относительно HEAD

**git config --global merge.conflictSSStyle diff3** - по умолчанию отображать в конфликтных файлах слияния 3 варианта (base, ours, theirs)

---alias---

**git config --pseudonymSaveLevel alias.commandPseudonym commandName**

**git config --global alias.conf config** - создать alias

**git conf --list** - использовать alias

**git config --global --unset alias.conf** - удалить alias

**git config --global alias.c 'config --global'** - создать alias с флагом

**sudo git config --system alias.fconflist '!echo "---System---"; git config --system --list; echo "---Global---"; git config --global --list; echo "---Local---"; git config --local --list;'** - создать alias с несколькими командами (выводит полный список настроек)

**git command -h** - короткое описание Git команды и её ключей

**git help command** - более подробное описание Git команды и её ключей в "читалке" LESS

---LESS---

**/word** - поиск по описанию команды:

**n** - следующее найденное

**Shift**+**n** - предыдущее найденное

**q** - выход из "читалки"

---принцип работы Git--

**git status** - текущий статус репозитория

**---Work Directory** - папка с проектом (все входящие в него файлы)

**.git** - репозиторий git (находится в папке проекта)

**---Index** - новые файлы проекта и изменения сначала необходимо проиндексировать. На этапе индексирования можно отредактировать список файлов которые будут включены в коммит (убрать ненужные и\или добавить игнорируемые).

**git add .** - проиндексировать все файлы

**git add fileName** - проиндексировать файл с именем fileName

**git add -f fileName** - проиндексировать игнорируемый файл

**git restore --stage fileName** - убрать файл из списка проиндексированных

**git reset fileName** - убрать файл из списка проиндексированных(М ? при использовании reset git сигнализирует М, как при ссылке двух веток на один коммит)

**git add -p fileName** - построчная индексация файла (можно добавить в индекс не все изменения в файле)

**git rm --cached fileName** - удалить из индекса но не удалять сам файл (команда применяется, если в предыдущий коммит попал файл который не нужно отслеживать)

**---Repository** - хранилище файлов и их изменений (для удобства навигации по версиям в репозиторий добавляются завершенные фичи или версии проекта)

**git commit** - сохранить проиндексированные файлы в репозиторий (сделать коммит). Git откроет текстовый редактор по умолчанию для описания коммита. Коммит совершается после сохранения сообщения и выхода из редактора (Ctrl+s, Ctrl+w, Ctrl+w).

**git commit -m "change description"** - сделать коммит с его описанием в терминале, без открытия текстового редактора Git-ом

**git commit -v** - перед коммитом в редакторе сообщения будет видно изменения Index относительно HEAD

---команды нескольких действий---

**git commit -a** или **git commit --all** - проиндексировать и закоммитеть ВСЕ ИЗМЕНЕНИЯ, НЕ КОММИТЕТ НОВЫЕ ФАЙЛЫ

**git commit filename** - проиндексировать и закоммитеть ИЗМЕНЕНИЯ в файле filename, НЕ КОММИТЕТ НОВЫЕ ФАЙЛЫ

**git mv oldFileName newFileName** - переименовать файл и проиндексировать

**git rm -r dirName** - удалить директорию и проиндексировать (без коммита; не удаляет файлы с изменениями)

**git rm -f fileName** - принудительное удаление файла и индексирование

---show---

**cat fileName** или **subl fileName**- посмотреть НА ДИСКЕ заданный файл

**git show :fileName** - посмотреть В ИНДЕКСЕ заданный файл

**git show @:fileName** - посмотреть в последнем КОММИТЕ заданный файл

**git show** или **git show HEAD** или **git show @**или **git show fileName** - посмотреть ИЗМЕНЕНИЯ

в коммите (файле)

**git show --pretty=fuller** - посмотреть изменения в коммит с инф. об авторе и коммитере

**git show --quiet** - сокращенная информация о коммите

**git show branchName** - посмотреть последний коммит в заданной ветке

**git show commitID** (min 4 symbol) - посмотреть коммит c ID

**git show @~** - посмотреть предыдущий коммит от HEAD

**git show @~~~** или **git show @~3** - посмотреть третий коммит назад от HEAD

**git show branchName~3** - посмотреть третий коммит с конца в заданной ветке

**git show branchName~5:fileName** - посмотреть в заданной ветке 5 коммитов назад заданный файл

---

**git commit --author='Another Author <exemple@meil.com>'** - коммит от имени Author

**git commit --date'...'** - коммит датой (используется вместе с author)

**.gitkeep** - Git не вносит в коммит пустые папки, если это необходимо, по соглашению, в ней создают пустой файл .gitkeep (можно любой другой файл)

---gitignore---

**git config --global core.excludesFile ~/.gitignore** - добавить в глобальный файл настроек путь к глобальному файлу .gitignore (файл не создается, путь - рядом с глобальным .gitconfig)

**.gitignore** - должен находится в корне репозитория и содержать названия папок и файлов не подлежащих сохранению (подключения к БД с паролем)

---branches---

**git branch** - список ветвей

**git branch -v** - список ветвей, короткий идентификатор последнего коммита и его описание

**git branch nameBranch** - создать новую ветку с именем nameBranch

**git checkout nameBranch** - переключится на ветку

**git checkout nameBranch -f** - переключится на ветку принудительно (без сохранения не закоммиченных изменений)

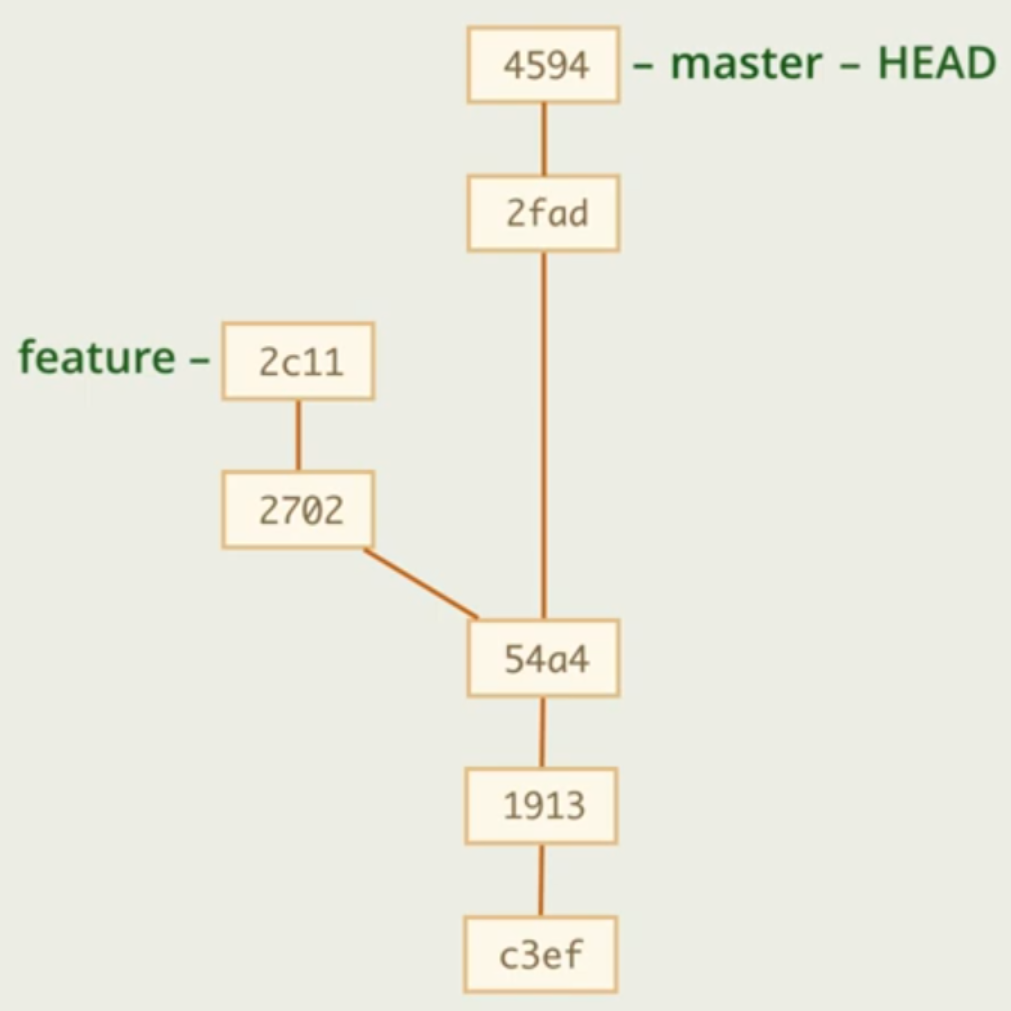
**git checkout -f** - в текущей ветке принудительно вернутся к последнему коммиту (без сохранения не закоммиченных изменений)

**git checkout -b nameBranch** - создать новую ветку и переключится на неё. Чаще используется если небольшие изменения кода в основной ветке переросли в большие и требуют вынесения в отдельную ветку (в основной ветке останется коммит с которого начинали делать изменения, а в новую ветку уйдут текущие изменения). Внимание!!! При использовании команды checkout -b исходная и созданная ветки до создания коммита для git будут равнозначными и при переходе между ними ошибки о не сохраненных данных возникать не будет и git без ошибки может закоммитеть новый код в основную ветку).

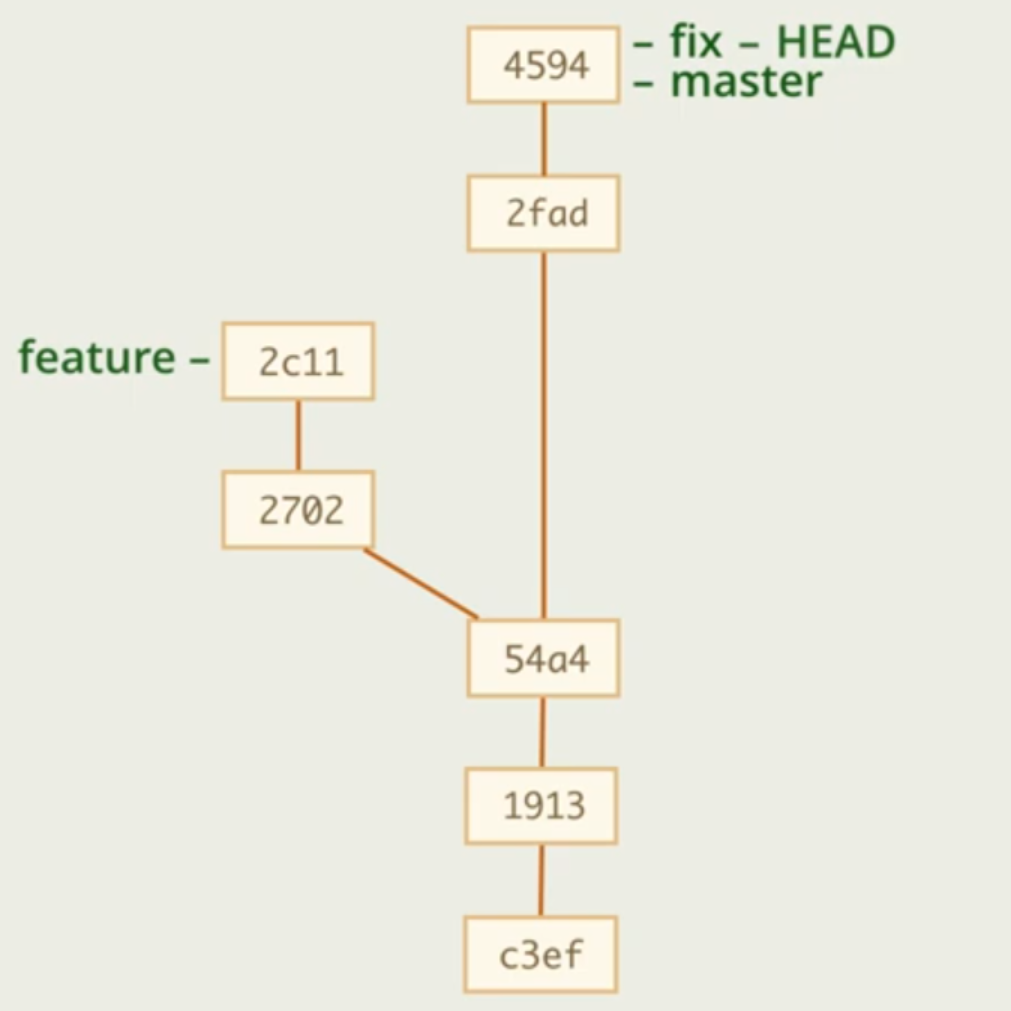
**git stash** - сохранит не закомиченные изменения в "тайник" и вернутся к последнему коммиту (для перехода на другую ветку)

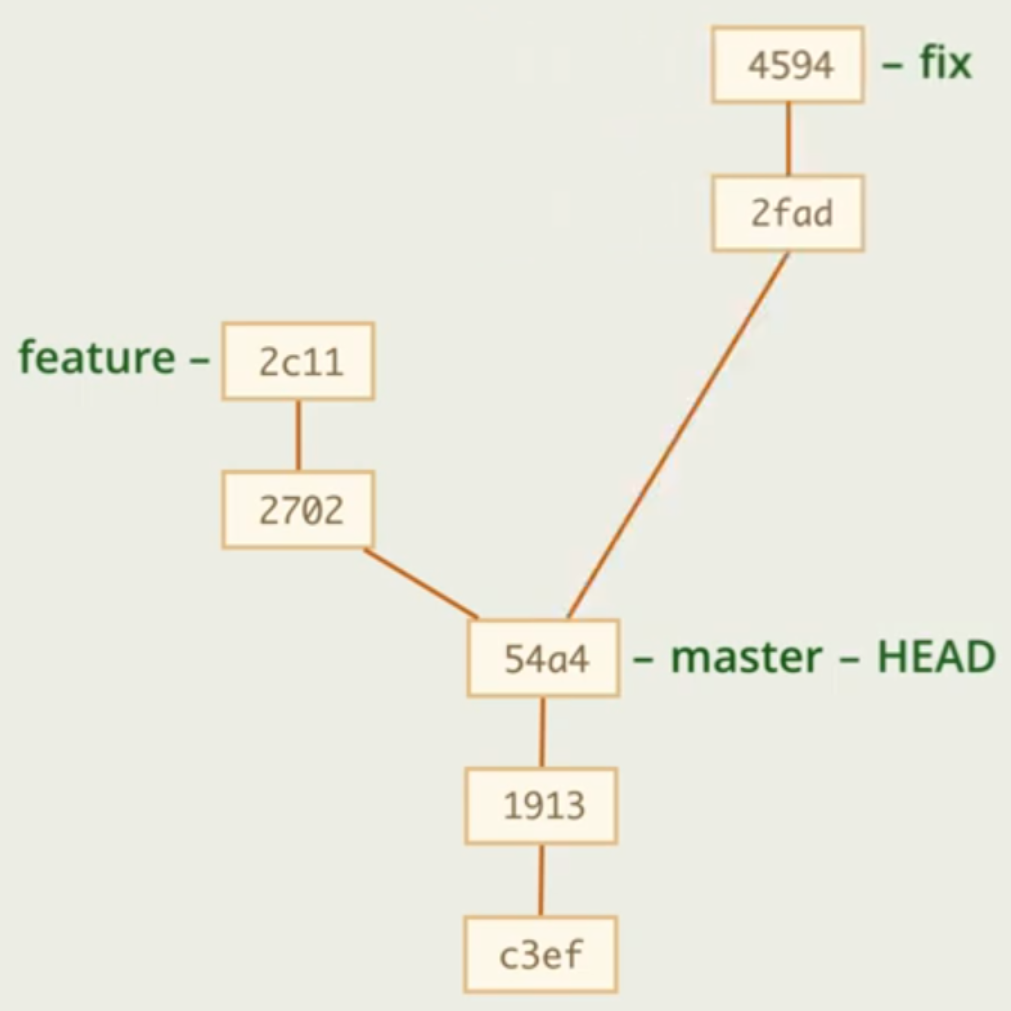
**git stash pop** - восстановить из "тайника" **В ТЕКУЩУЮ ВЕТКУ** (можно переместить изменения в другую ветку)

---перенос нескольких последних коммитов из основной ветки в новую---

**git branch fix** - делаем новую ветку fix, по сути теперь последний коммит это две ветки master и fix.

**git checkout fix** - переходим на новую ветку (git не даёт переименовывать текущую ветку)

**git branch -f master 54a4** или **git checkoute -B master 54a4**- принудительно передвинуть ветку master на коммит 54а4

**git branch -f master fix** - передвигать ветки можно не по ID коммита и на другую ветку (если передумали; так-же нельзя стоять на передвигаемой ветке)

---

**git restore fileName/dirName** или **git checkout fileName/dirName** - восстановить измененный файл/директорию до состояния его предыдущего индексирования. Если имя директории совпадает с именем ветки нужно использовать формат "**-- name**"

**git checkout HEAD fileName/dirName** - восстановить измененный не проиндексированный/ проиндексированный файл/директорию до состояния предыдущего коммита

**git checkout commitID/branch fileName/dirName** - в текущую ветку копировать/заменить файл/директорию из коммитаID/веткиName и проиндексировать его

---просмотр дерева коммитов---

**git log** - отобразить текущую ветку от HEAD с подробной информацией (не очень удобно)

**git log --oneline** - отобразить текущую ветку от HEAD, один коммит в одну строку(короткий ID)

**git log branchName --oneline** - отобразить заданную ветку

**git log --pretty=medium** - формат вывода информации по ветке = стандартный (git log)

**git log --pretty=oneline** - отобразить текущую ветку от HEAD, один коммит в одну строку(длинный ID)

**git log --no-decorate** - без декоратора (где HEAD, вершина других веток ...)

**git log -p** или **git log --patch** - текущая ветка с добавлением к каждому коммиту diff

кастомизация git log - https://www.youtube.com/watch?v=Oim9dbpbCMc&list=PLDyvV36pndZFHXjXuwA\_NywNrVQO0aQqb

**git log branchesNames --oneline --graph** - отобразить перечисленные ветви с консольным деревом (очень наглядно)

**git log --all --oneline --graph**- отобразить все ветви с консольным деревом

**git gui** - графические утилиты git

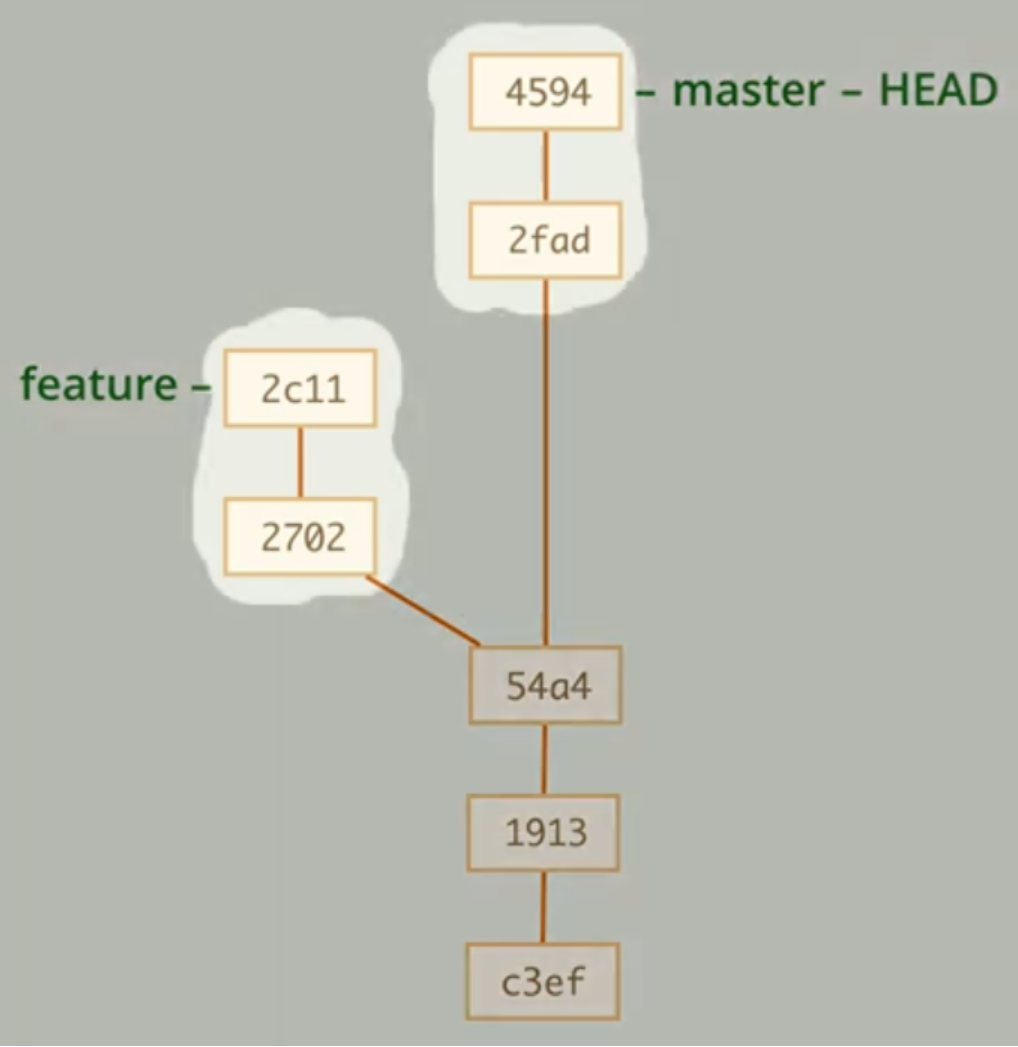
**git log feature ^master** или **git log feature..master** - вывести все коммиты ветки feature с ответвления от master

**git log HEAD..feature** или **git log ..feature**

**git log feature..HEAD** или **git log feature..**

**git log feature..master --boundary** - вывести все коммиты ветки feature с ответвления от master с пограничным коммитом

**git log feature...master** - вывести все коммиты ветки feature и master с ответвления без пограничного коммита

**git log feature...master --boundary** - вывести все коммиты ветки feature и master с ответвления и с пограничным коммитом

**git log fileName** - выведет историю изменения файла в текущей ветке (--oneline)

**git log dirName, git log file1 file2 .. fileN**

**git log fileName --follow** - выведет историю изменения файла с переименованиями

**git log master../...feature fileName** - вывести изменения файла по веткам

**git log -- fileName** - если имя файла/директории совпадает с веткой, двойной дефис укажет

git явно, что это файл

поиск в коммитах по регулярным выражениям

https://www.youtube.com/watch?v=lhrchh5dqH0&list=PLDyvV36pndZFHXjXuwA\_NywNrVQO0aQqb

**git blame fileName** - посмотреть автора каждой строки в файле

**git log main --oneline --first-parent** - у коммита слияния посмотреть дерево только первого родителя

---reflog---

**git reflog** или **git log --oneline -g** - (reference log) журнал ссылок для HEAD (история перемещения по коммитам и веткам)

**git reflog --date=iso** - журнал ссылок с датами

**git reflog branchName** - журнал ссылок по заданной ветке

**git branch oldOrNewBranchName refLogName/reflogDate** (git branch fourth HEAD@{1}) или **git branch oldOrNewBranchName idHeadBranch** - восстановить ветку по reflog или id

**git checkout @{-1}** или **git checkout -** или **git checkout @{-3}** - переключится на предыдущую ветку или на 3 ветки назад по ссылке из reflog

**git branch --merged** - список веток объединенных с HEAD

**git branch --no-merged** - список веток не объединенных с HEAD

---приведение директории проекта к состоянию коммита---

**git reset --hard** или **git checkout -f** - вернет отслеживаемые файлы/папки к состоянию HEAD и очистит Index

**git checkout -f fileName** - вернет fileName к состоянию HEAD

**git clean -dxf** - удалить не отслеживаемые файлы и **-d** директории, **-x** игнорируемые

---исправление ошибок в предыдущем коммите (+ commit --amend)---

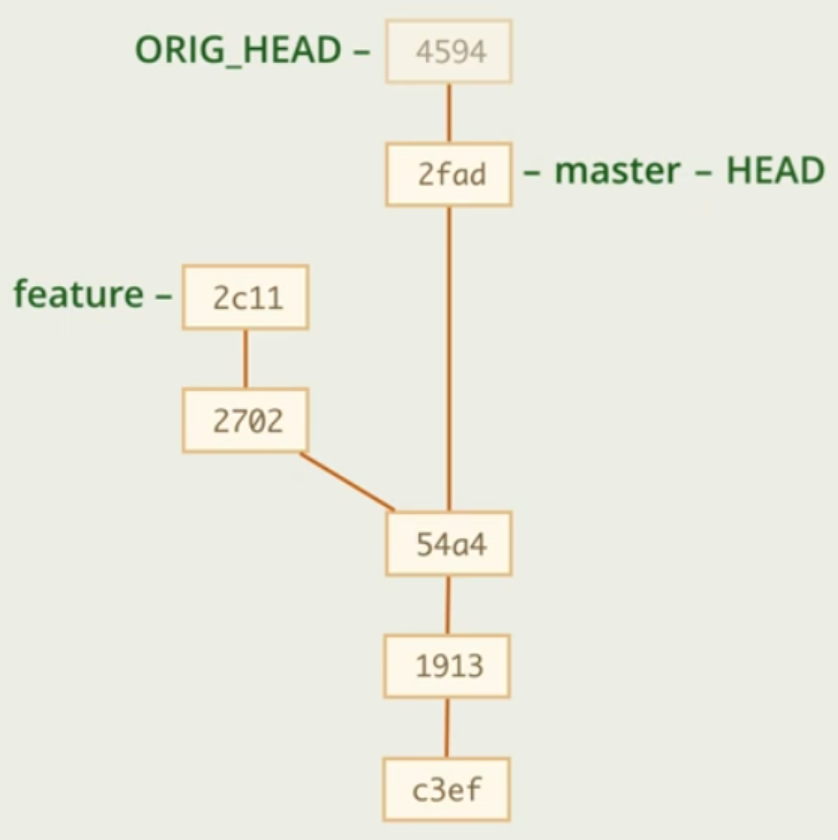
red - приводится к состоянию коммита; black - остается в исходном состоянии

суть: вернутся в коммит где была допущена ошибка, исправить её и сохранить изменения (создастся новая ветка HEAD, а прошлая станет недостижимой ORIG\_HEAD)

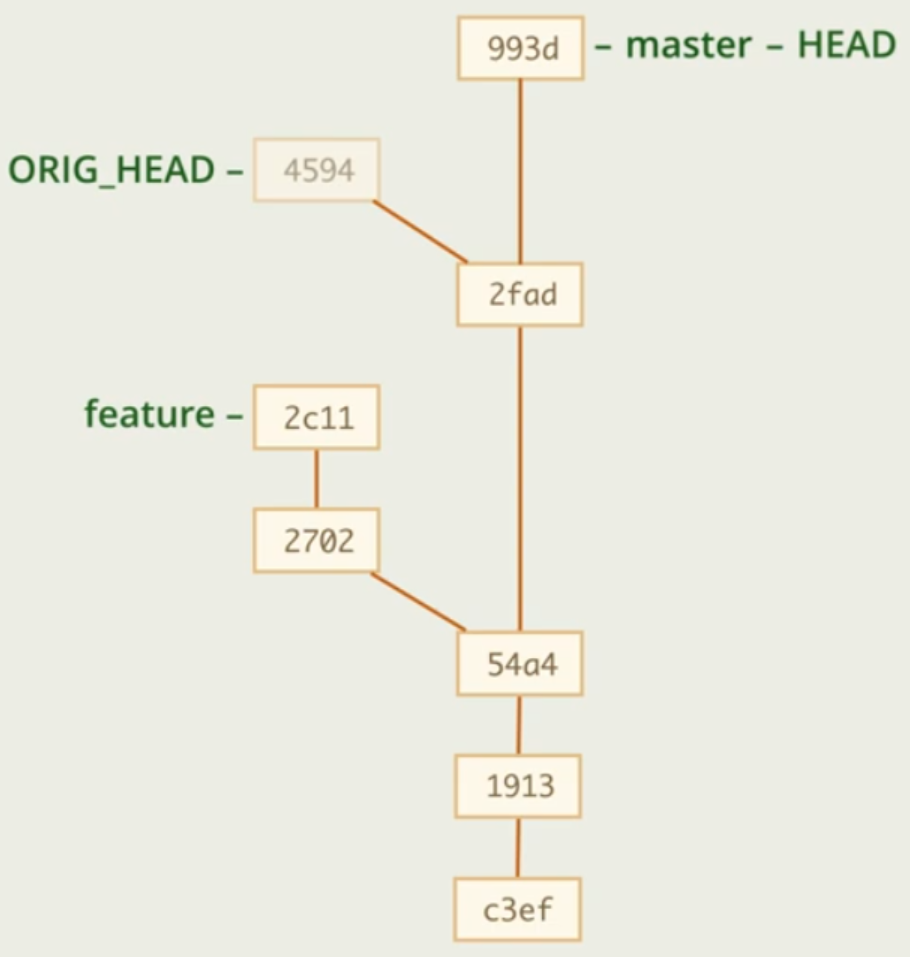
--reset --hard--

переходит на указанный коммит и обновляет до его состояния Work Directory и Index

**git reset --hard @~** или **git reset --hard 2fad** - переносим HEAD на один коммит назад

вносим изменения

**git commit -a** - индексируем и коммитем изменения (создастся новая ветка)

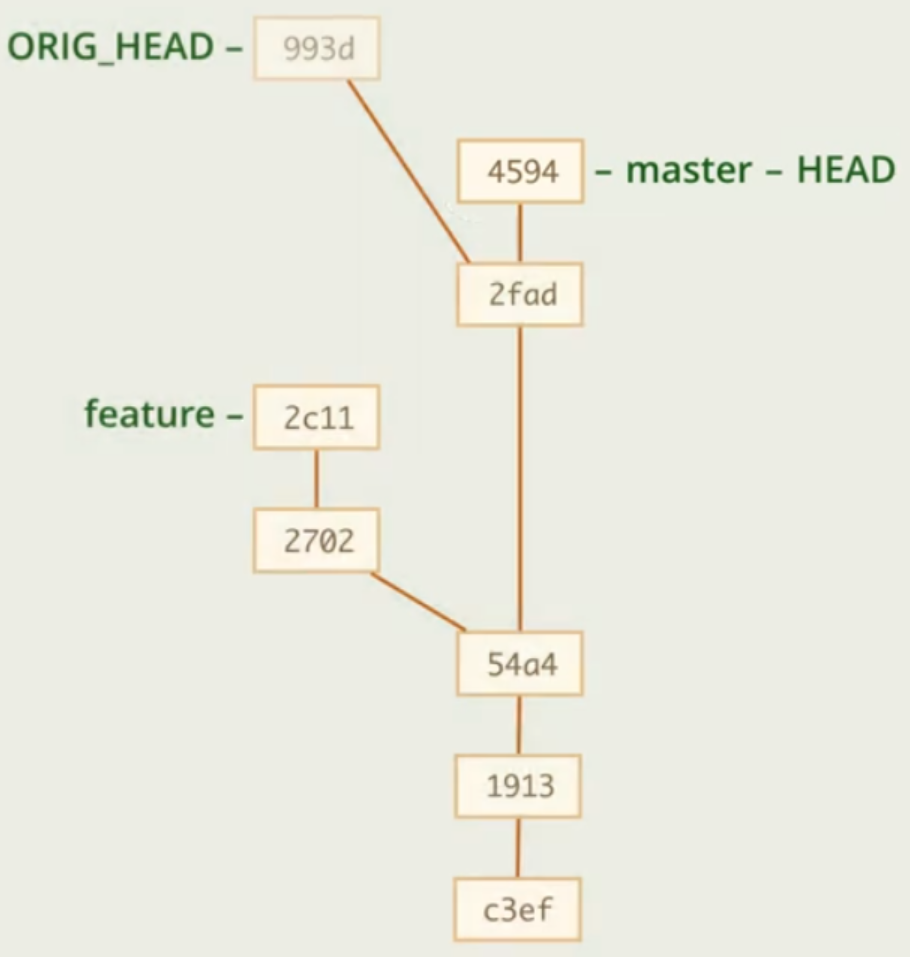


теперь старая вершина ветки master становится "недостижимой" и автоматически удалится через некоторое время, но пока доступна по ссылке ORIG\_HEAD или по ID

Если нужно все поменять обратно

**git reflog master** - посмотреть журнал ссылок для определения ID недостижимой ветки

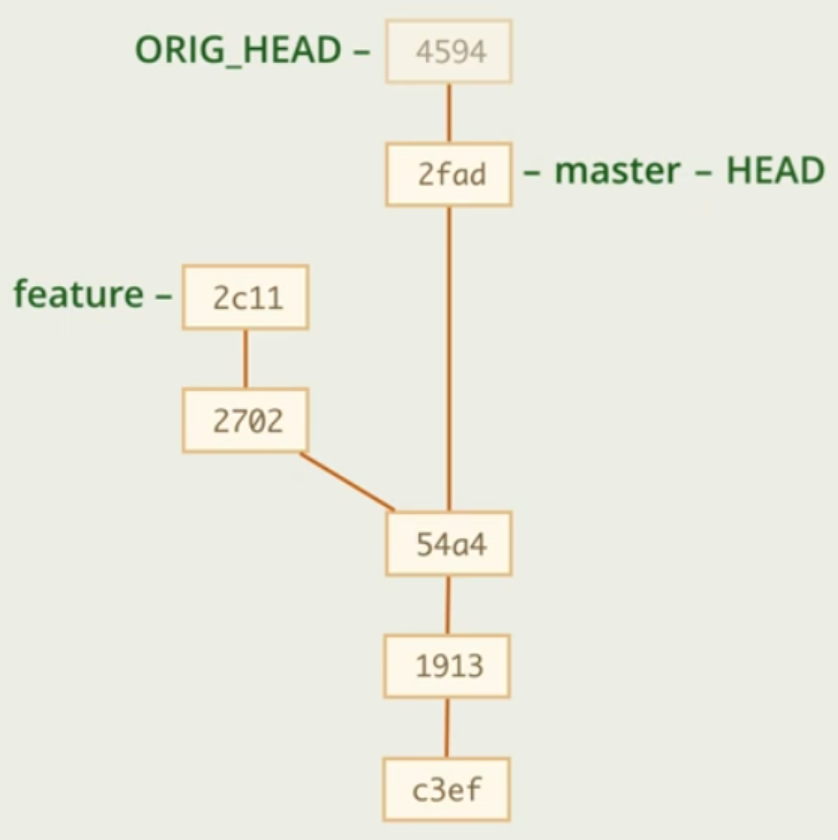
**git reset --hard ORIG\_HEAD** или **git reset --hard 4594** - устанавливаем ветку master как было



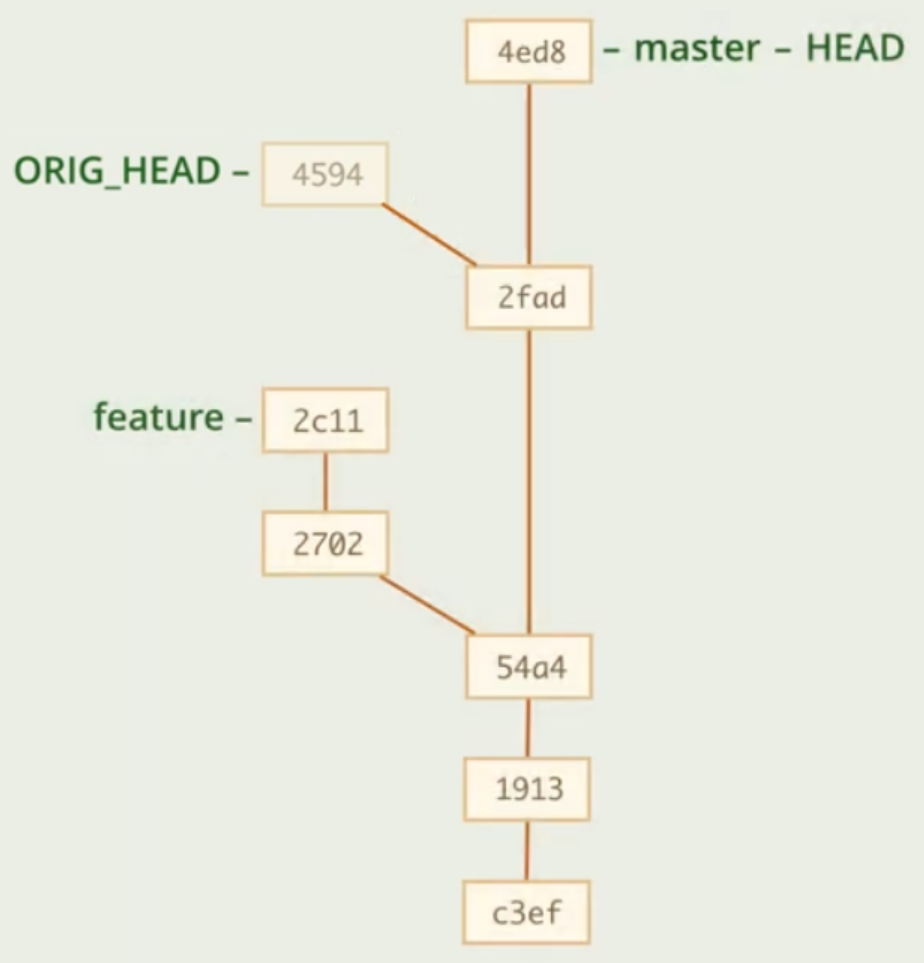
--reset --soft--

переходит на указанный коммит и оставляет в том-же состоянии Work Directory и Index

**git reset --soft @~** или **git reset --soft 2fad** - переносим HEAD на один коммит назад

вносим изменения и индексируем их

**git commit -c ORIG\_HEAD** - закоммитеть изменения взяв описание, дату создания, автора из orig\_head и отредактировать описание

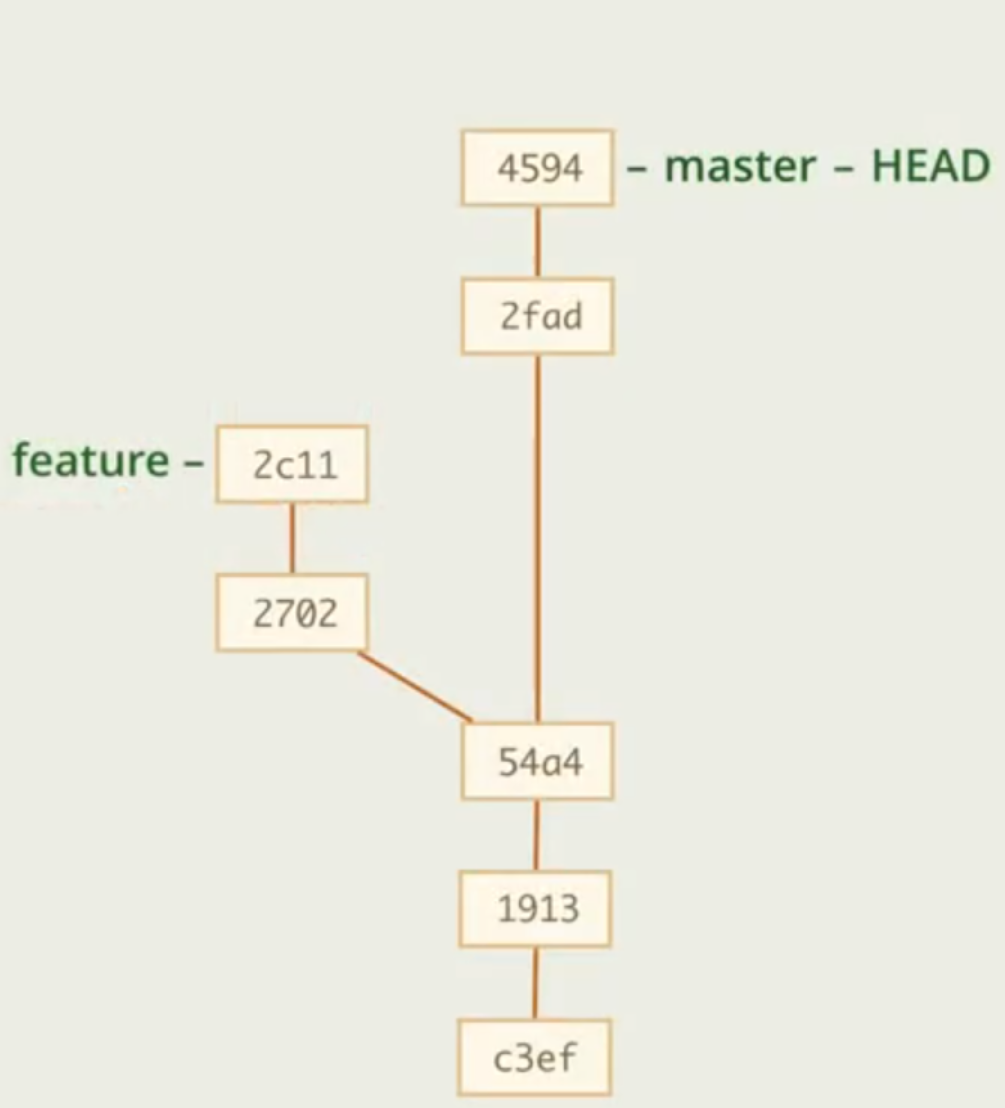
**git commit -C ORIG\_HEAD** - закоммитеть изменения взяв описание, дату создания, автора из orig\_head и не редактировать ничего

По умолчанию у HEAD и ORIG\_HEAD в описании будут отличатся только Коммитер и Дата коммита

**git commit -C ORIG\_HEAD --reset-author** - закоммитеть изменения взяв только описание из orig\_head, остальное установит на текущие значения

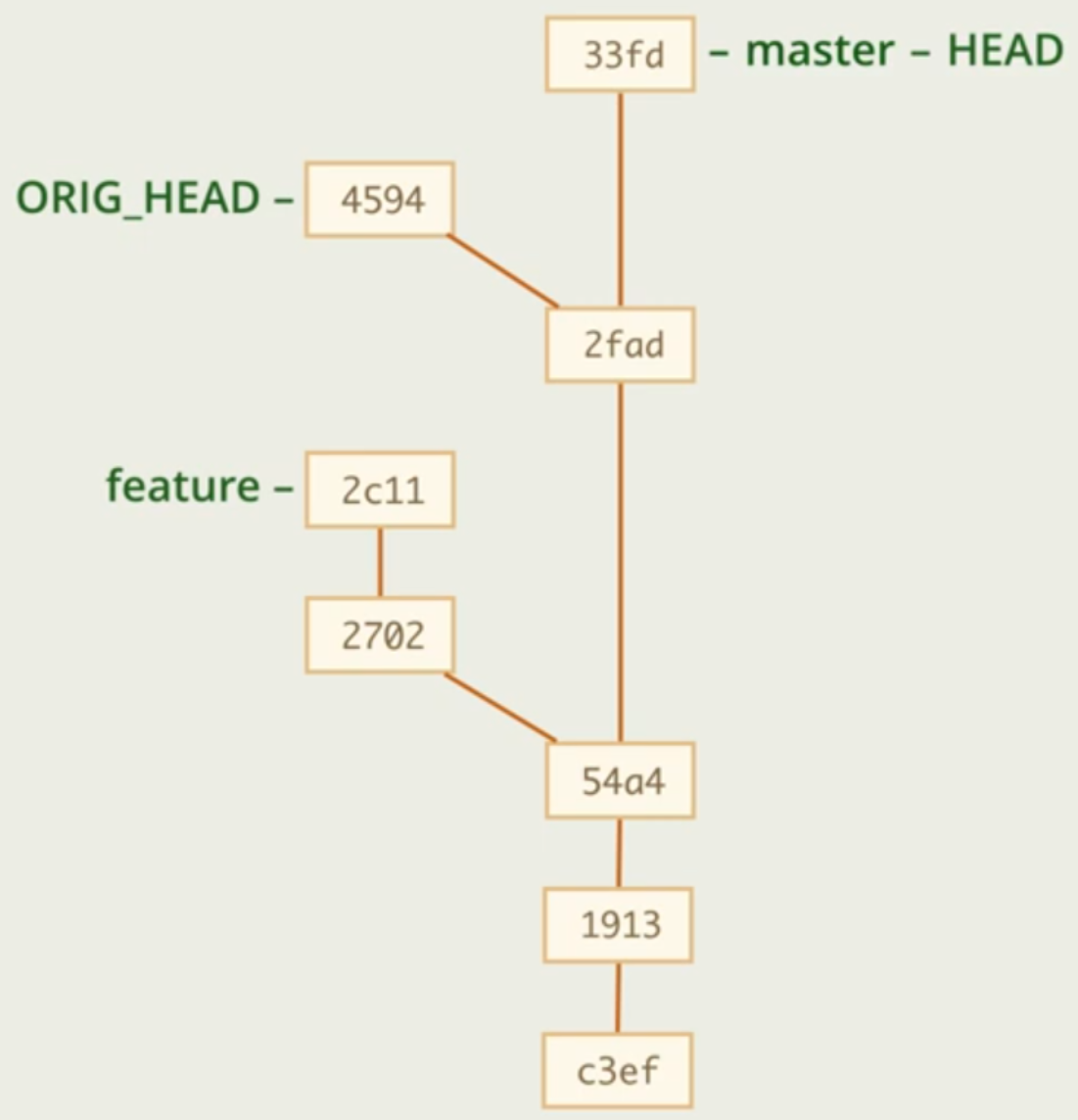
--commit --amend--

отредактировать последний коммит, выполняет те-же действия что и resset --soft, но одной командой



вносим изменения и индексируем их

**git commit --amend** - внести изменения в последний коммит (даже в описание)

--amend переносит HEAD на один коммит назад, оставляя рабочую директорию и индекс неизменными, создает новый коммит, взяв информацию об авторе, времени и описание из заменяемого и предложит отредактировать описание

**git commit --amend --reset-author** - изменить автора на текущего

**git commit --amend --no-edit** - без вызова редактора (оставить описание неизменным)

--reset --mixed--

переходит на указанный коммит и оставляет в том-же состоянии Work Directory но обновляет до состояния коммита Index

**git reset --mixed @~** или **git reset @~** - переносим HEAD на один коммит назад

**git reset HEAD** или **git reset** - сбросит все изменения в Index, но оставит изменения в файлах

**git reset fileName** - сбросить изменения в Index fileName

---diff--- (игнорирует новые\неотслежуемые файлы)

**git diff startCommit finishCommit** - как изменился finishCommit относительно startCommit

**git diff @~ @** или **git diff @~** - как изменился HEAD относительно прошлого коммита

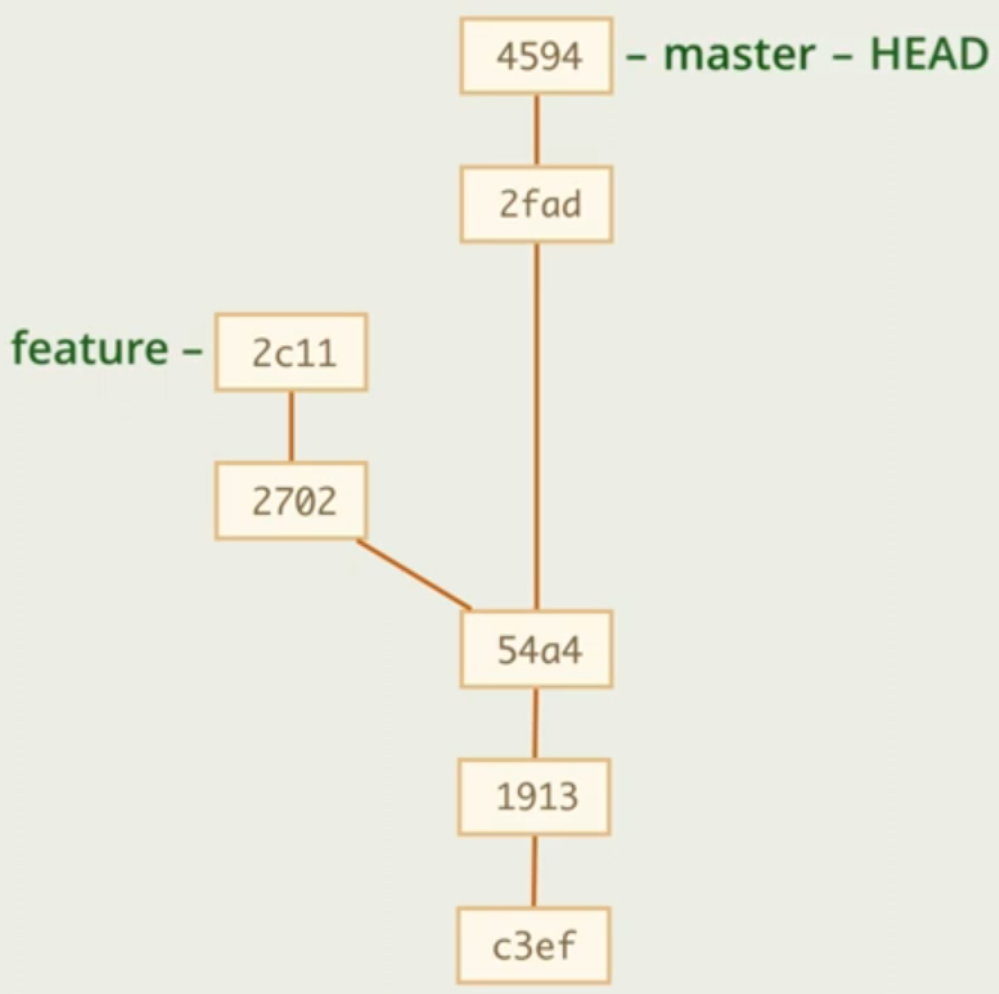
**git diff HEAD**- как изменилась Work Directory относительно HEAD (какие изменения были внесены в проект относительно последнего коммита)

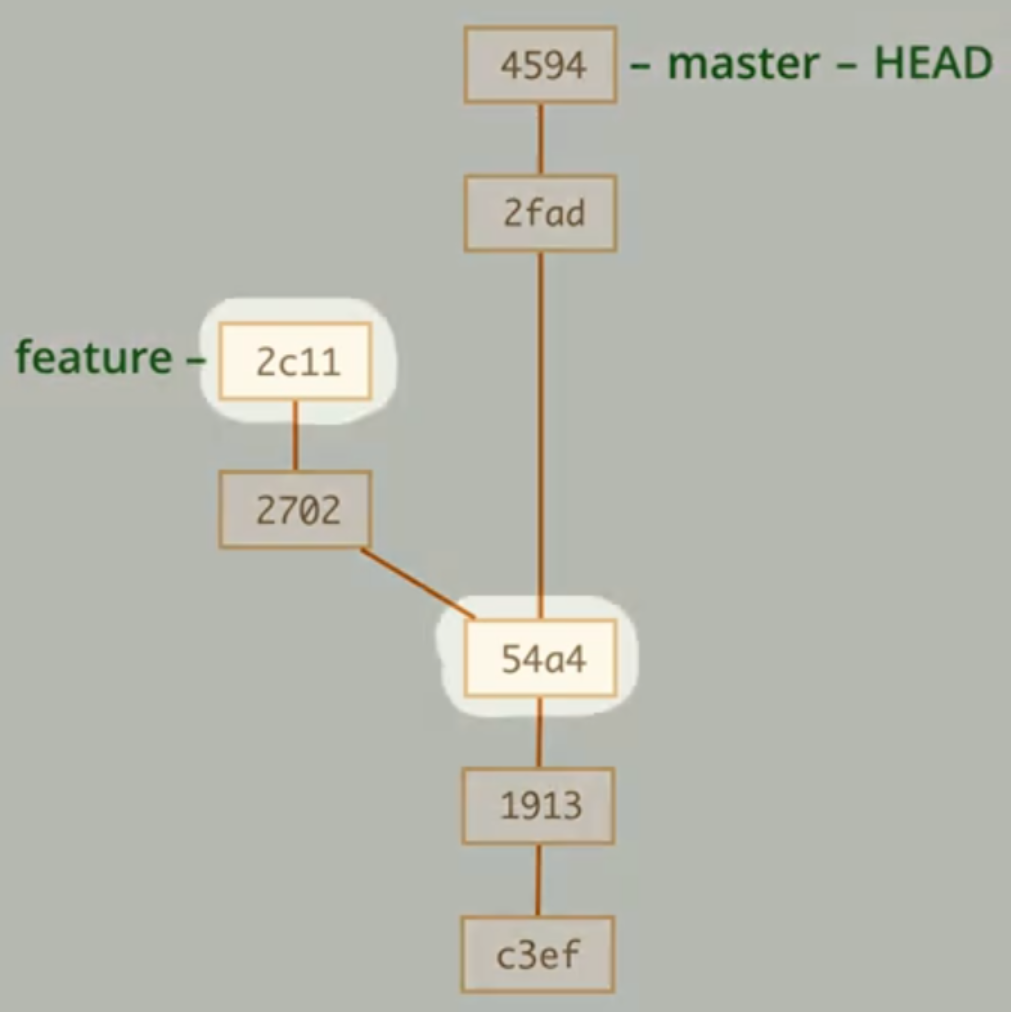
**git diff** - как изменилась Work Directory относительно Index (какие изменения были внесены в проект относительно последнего индексирования)

**git diff --cached** или **git diff --staged** - какие изменения есть в Index относительно HEAD (какие изменения проиндексированы, но еще не закоммичены)

**git diff branchName** - как изменился HEAD относительно вершины ветки branchName

**git diff master feature** или **git diff master..feature** - какие отличия в feature относительно master

**git diff master...feature** - какие отличия в вершине feature относительно коммита отхождения от ветки master

**git diff feature...master** - какие отличия в вершине master(HEAD) относительно коммита

отхождения ветки feature

**git diff --name-only master feature** - перечень различающихся файлов из вершин веток master и feature

**git diff ... -- <paths>** любому сравнению можно указать определенный файл или директорию (в том числе текущую .)

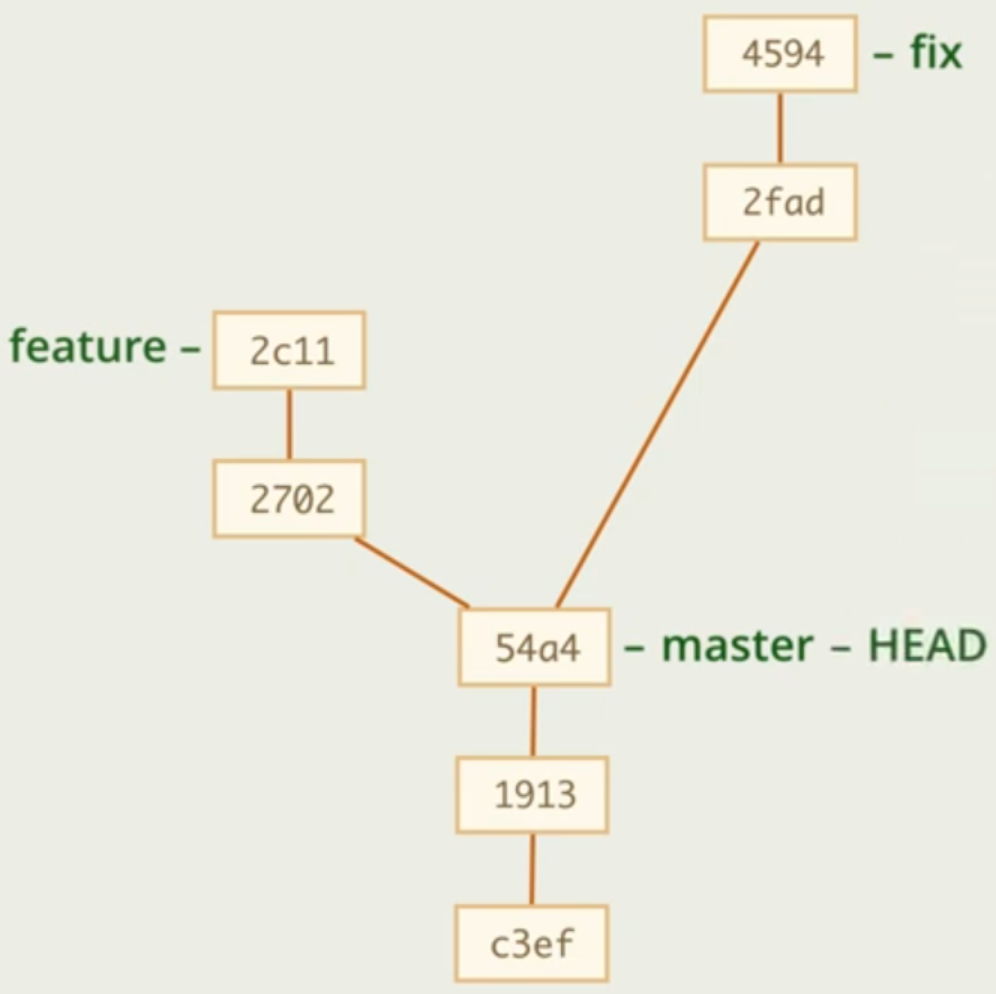
https://www.youtube.com/watch?v=1oExHLJXBIg&list=PLDyvV36pndZFHXjXuwA\_NywNrVQO0aQqb

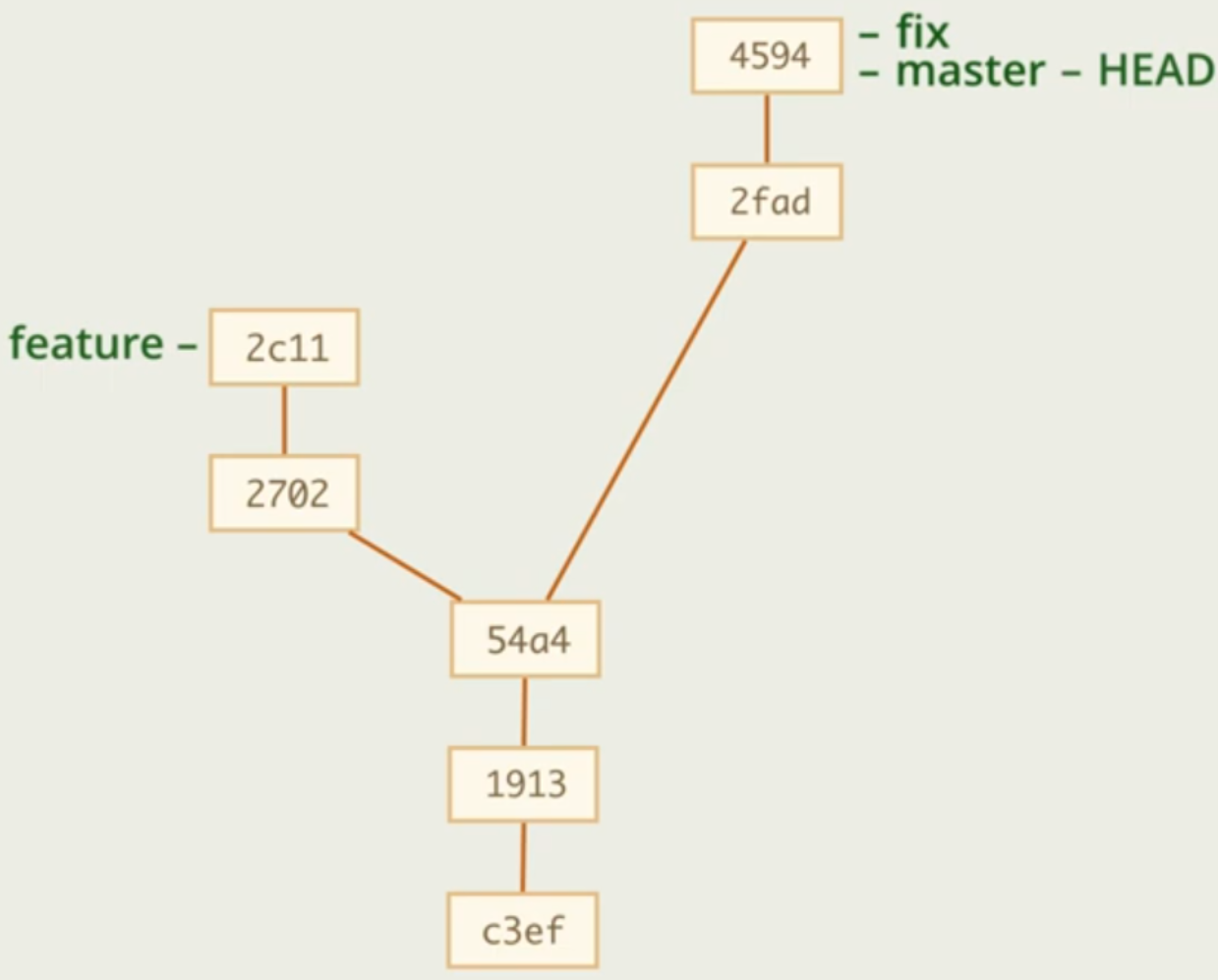
---merge---

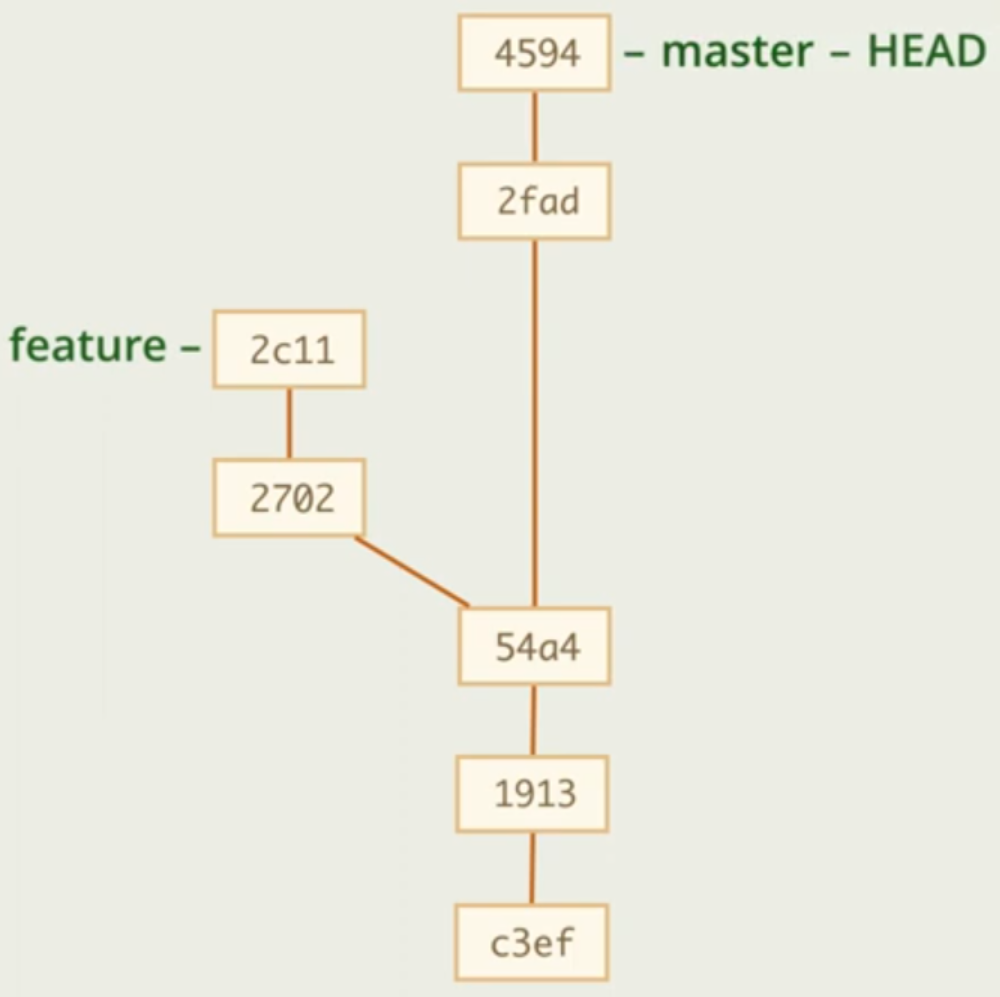
---слияние перемоткой (fast-forward)---

подходит если ответвление и слияние с основной веткой происходит в одном коммите (54а4),

аналог перемещения master в fix (обратное примеру выше)

**git merge fix** - слияние методом fast-forward ветки fix с master

**git branch -f master ORIG\_HEAD** - отмена слияния. Git хранит ID головного коммита до слияния в файле .git/ORIG\_HEAD

**git** **branch -d fix** - удалить ветку в которой нет отличия от ветки main

**git branch -D branchName** - принудительно удалить ветку. Git удалит лишь ссылку(имя) ветки, некоторое время ветка будет доступна к восстановлению (см ниже)

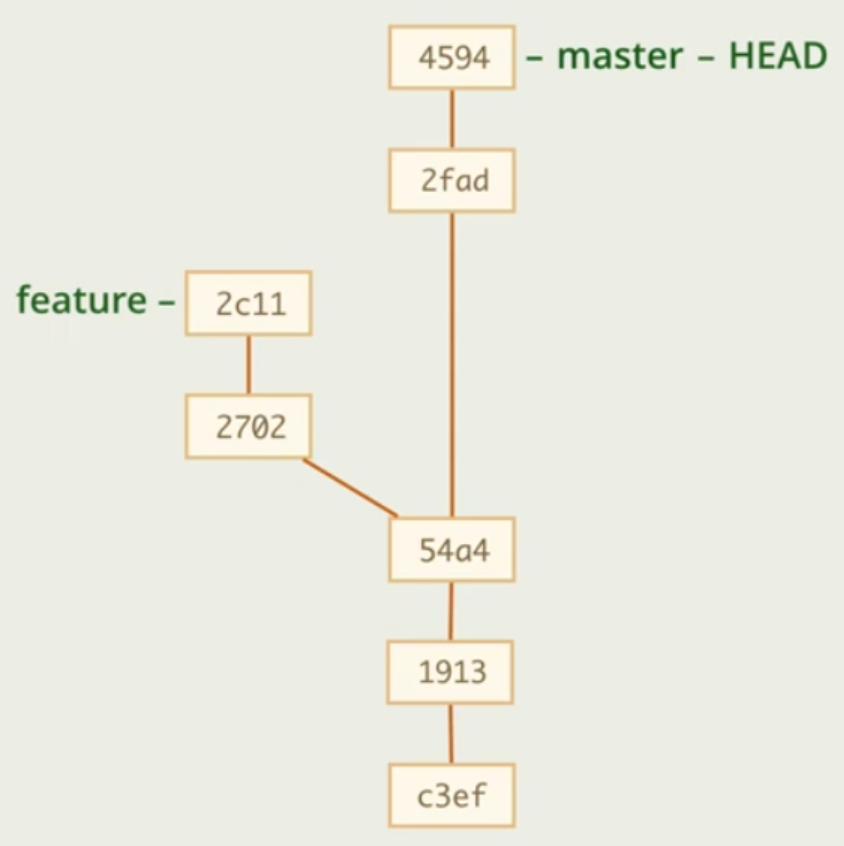
**git merge --no-ff fix** - принудительно сделать истинное слияние, а не fast-forvard

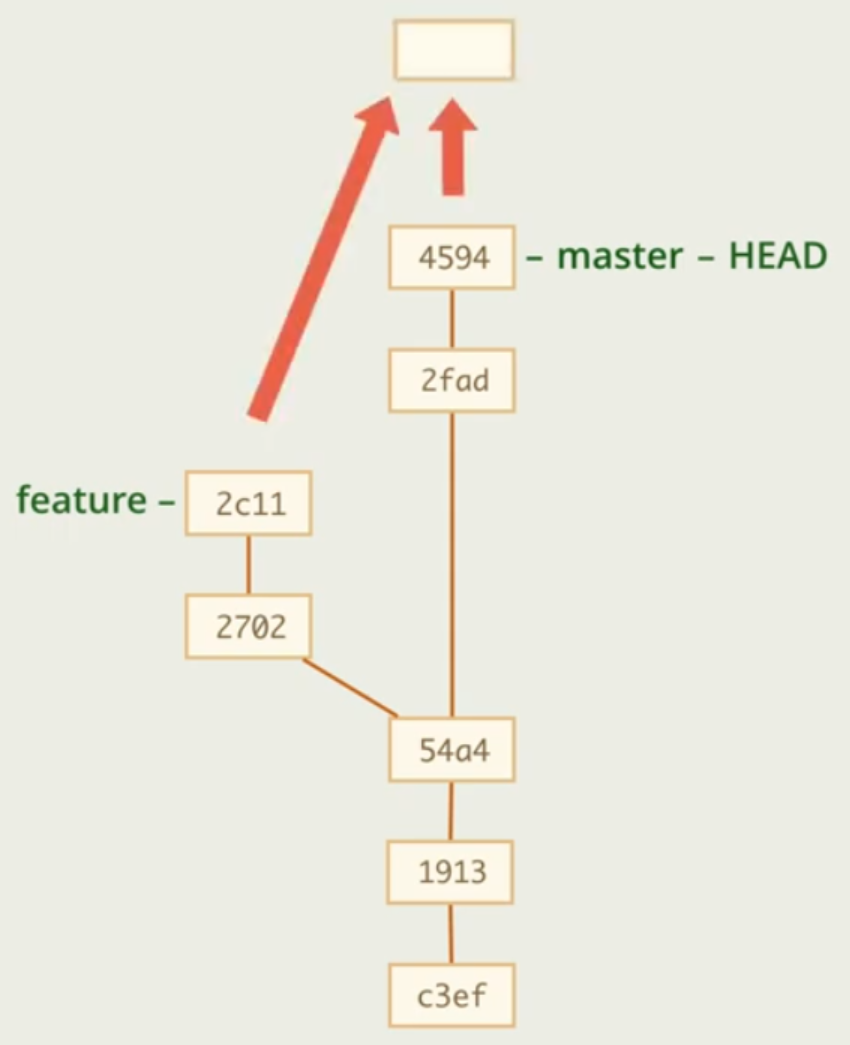
**git merge --ff fix** - принудительное слияние fast-forvard, если в конфиге оно отключено по умолчанию

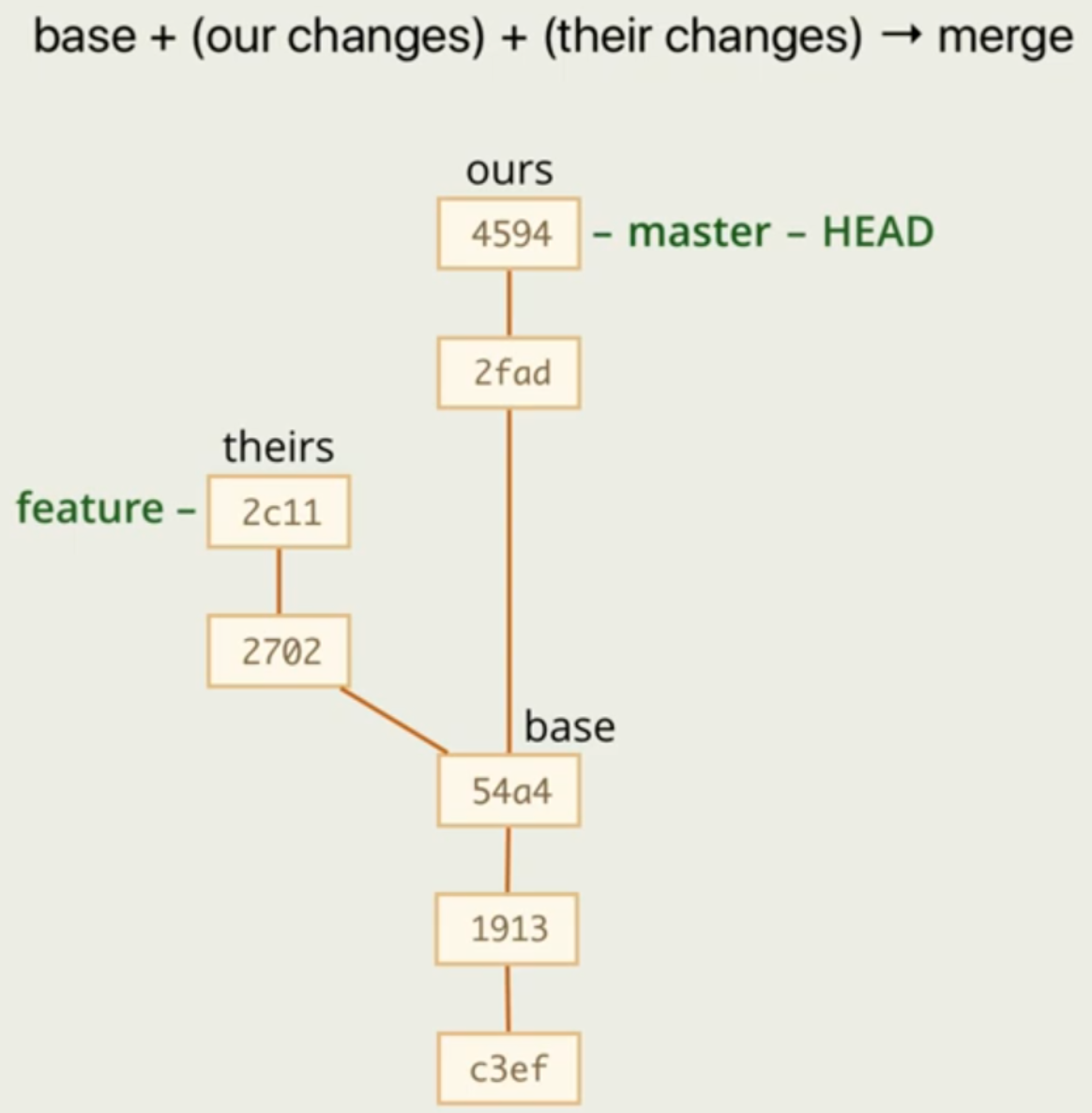
---истинное слияние---

Произведем слияние ветки feature с master. Желательно иметь чистый git statu, или незакоммиченые изменения должны быть одинаковы.

**git merge feature** - слияние ветки feature с master

После слияния образуется новый коммит содержащий изменения из обеих веток

При слиянии git за основу возьмёт "базовый" коммит **base** (общий предок обоих веток) добавит в него изменения из "нашего" коммита **ours** и добавит изменения из "их" коммита **theirss**.

**git merge-base master feature** - вывести ID базового коммита

Если при слиянии веток в одной файл менялся а в другой нет, то в финальный коммит попадет измененный файл (как при fast-forward): base = ours -> theirs; base = theirs -> ours

Git использует много алгоритмов при истинном слиянии и даже если файл в ours и theirs подвергся изменениям, то git попробует их объединить. И только если изменения в ours и their произошли в одном месте фала возникнет Merge conflict.

В состоянии прерванного слияния (Merge conflict) Git делает отдельный коммит **MERGE\_HEAD** и перенесет нас туда.

**cat .git/MERGE\_HEAD** - посмотреть ID MERGE\_HEAD

**cat conflictFileName** или **subl conflictFileName** - в конфликтном файле Git отметит конфликтные места:

<<<<<<< ours

//code from the master branch

=======

//code from the feature branch

>>>>>>> theirs

**git checkout --ours conflictFileName** - привести конфликтный файл к состоянию master

**git checkout --theirs conflictFileName** - привести конфликтный файл к состоянию feature

**git checkout --merge conflictFileName** - привести конфликтный файл к состоянию MERGE\_HEAD

**git checkout --conflict=diff3 --merge conflictFileName** - отобразить в конфликтном файле состояния base, ours и theirs

**git merge --abort** или **git reset --merge** - отменить merge, откатит до старта слияния

**git reset --hard** - отменить merge. ВНИМАНИЕ! все не закоммиченные изменения удалятся!

При конфликте слияния в индексе будут все три варианта файла:

**git show :1: conflictFileName** - base

**git show :2: conflictFileName** - ours

**git show :3: conflictFileName** - theirs

И поэтому некоторые команды git будут работать иначе (например git commit выдаст ошибку)

**git add conflictFileName** - после устранения конфликта в файле рабочей директории индексируем его

**git merge --continue** или **git commit** - продолжить слияние после удаления конфликта

---Git + GitHub---

**upstream** (вверх по течению) - исходный проект(репозиторий) в стороннем аккаунте GitHub

**origin** (источник) или **fork** (вилка, ответвление)- копия исходного проекта в моем аккаунте

**local copy** (локальная копия) - копия проекта(репозитория) на моем компьютере

**upstream fork button** - кнопка копирующая сторонний проект(репозиторий) в мой аккаунт

**projLinkHTTPS** - Code->HTTPS->Copy - копировать HTTPS ссылку на проект

**projLinkSSH** - Code->SSH->Copy - копировать SSH ссылку на проект

**git clone projLink** - сделать локальную копию проекта на компьютере в текущей директории

**git remote add origin ...** и **git remote add upstream ...** - в созданном локальном проекте добавить настройки с адресами удаленных серверов (своего и стороннего) или изменить эти настройки в существующем локальном проекте

**git remote** - посмотреть перечень удаленных серверов

**git remote -v** - посмотреть настройки удаленных серверов

**git remote show origin** или **git remote show upstream** - показать что git сделает при push или pull в origin или upstream

**git pull** - скачать изменения из удаленного репозитория в локальный (команда pull без аргументов берет адрес из origin; может создаваться коммит слияния)

**git pull upstream main** - скачать изменения из upstream ветки main (может создаваться коммит слияния)

**git push** - отправить закомиченные изменения из локального репозитория в удаленный (команда push без аргументов берет адрес из origin)

**git push -u origin main** - при первом заливе в пустой удаленный репозиторий настройки нужно указывать принудительно

**git push upstream main** - отправить закомиченные изменения из локального репозитория в upstream (если у пользователя есть право)

---Creating a personal access token--- [Создание личного маркера доступа](https://docs.github.com/ru/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token)

Фото профиля->Settings->Developer settings->Personal access tokens

## **Fine-grained personal access tokens** - создать пользователю токен с тонкими настройками, можно выдать права на один репозиторий. Для доступа к репозиторию на запись и чтение обязательно в Permissions установить настройку **Contents** на read and write. Максимальное время действия - год.

## **Personal access tokens (classic)** - создать пользователю токен на все репозитории аккаунта. Для получения прав на запись и чтение обязательна галочка repo (и всех ее подгалочек)

**git remote add origin https://github.com/satnetuser001/repoName.git** - добавить/изменить в локальный репозиторий адрес origin и метод передачи данных HTTPS

**git remote add upstream https://github.com/satnetuser003/upstream** - добавить/изменить в локальный репозиторий адрес upstream и метод передачи данных HTTPS

**git remote -v** - посмотреть локальные настройки remote:

origin **https://**github.com**/**satnetuser001/origin.git (fetch)

origin **https://**github.com**/**satnetuser001/origin.git (push)

upstream **https://**github.com**/**satnetuser003/upstream (fetch)

upstream **https://**github.com**/**satnetuser003/upstream (push)

---Creating SSH connection---

[Создание нового ключа SSH](https://docs.github.com/ru/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent) [Добавление нового ключа SSH в GitHub](https://docs.github.com/ru/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account)

**/home/anton/.ssh** - место расположения ключей ssh

**ssh-keygen -t ed25519 -C "satnetuser001@gmail.com"** - генерирование приватного (id\_ed25519) и публичного (id\_ed25519.pub) ssh ключей для пользователя

**eval "$(ssh-agent -s)"** - запустить SSH агент в фоновом режиме

**ssh-add ~/.ssh/id\_ed25519** - добавить приватный SSH ключ в ssh-agent

Фото профиля->Settings->SSH and GPG keys->New SSH key - ввести публичный ssh-ключ и описание (для какого устройства, ОС, пользователя)

**ssh -T git@github.com** - проверка ssh соединения

**git remote add origin git@github.com:satnetuser001/repoName.git** - добавить/изменить в локальный репозиторий адрес origin и метод передачи данных SSH

**git remote add upstream git@github.com:satnetuser003/upstream** - добавить/изменить в локальный репозиторий адрес upstream и метод передачи данных SSH

**git remote -v** - посмотреть локальные настройки remote:

origin **git@**github.com**:**satnetuser001/origin.git (fetch)

origin **git@**github.com**:**satnetuser001/origin.git (push)

upstream **git@**github.com**:**satnetuser003/upstream (fetch)

upstream **git@**github.com**:**satnetuser003/upstream (push)