# Навигация

pwd — Показать текущий каталог  
ls - Показать файлы в данной папке, кроме скрытых  
ls -f — Показать файлы в данной папке, включая и скрытые  
cd c:/ — Перейти в конкретный каталог  
cd - — Вернуться назад  
cd .. — Выйти на 1 уровень вверх  
cd ../.. — Выйти на 2 уровня вверх

# Основная конфигурация

**git init** – создаст гит-репозиторий (появиться папочка .git в корне проекта)

**git config --global user.name "John Doe"** – уккзать свое имя

**git config --global user.email** [**johndoe@example.com**](mailto:johndoe@example.com) **–** указать свою почту

**cat <.git/config>** - показать содержимое файла <config>

Настройки бывают:

* **--system** – общесистемные, для всех пользователей, хранится в **C:\Program Files\Git\etc\gitconfig**
* **--global** – на уровне конкретного пользователя (относится ко всем проектам пользователя) пишутся в **C:\Users\Ilya\.gitconfig**
* **--local** – на уровне проекта – они пишутся в файл .**git/config**

Git ищет эти параметры снизу вверх

**git config --list (--global)** – список параметров конфигурации (только глобальные)

**git config --unset** (something) – что-то отменить в настройках

**git config –h** – посмотреть все команды конфига

**git help** **<command>** - помощь

**/q** – выход из листалки less

# Создание репозитория, первый коммит

**git init** – создаст гит-репозиторий (появиться папочка .git в корне проекта)



В index пишутся отслеживаемые на данный момент файлы, в Repository – вся история коммитов

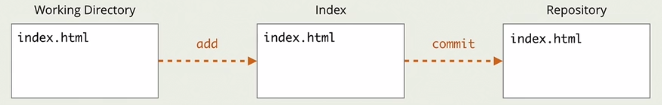
**git status** – взгляд со стороны git на файлы нашего проекта.

* Untracked files – те, которые не добавлены в index

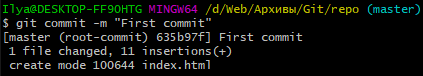
**git add <file>** – добавит файл в index

**git commit -m "comment"**  – добавить файл в прямоугольник Repository

*Когда git status ничего не выводит, значит картинка такая:*



Состояние рабочей области индекса и репозитория равны

 - в третьей строке – id коммита

**git reset HEAD <file>** – удаляет из index лишний <file>

# Игнорирование: .gitignore

[https](https://learn.javascript.ru/screencast/git#config-gitignore)://learn.javascript.ru/screencast/git#config-gitignore

# Git show, автор и коммиттер

**git show <первые 4 символа id коммита>** - детали коммита, если без id – то текущего коммита

# Подключение к удаленному серверу и git репозиторию

**git remote add origin** [**https://github.com/Qexly/<repos**](https://github.com/Qexly/%3crepos)**> -** Подключение к github

**git push -u origin master –** залить из локального репо на гитхаб

**git clone** [**https://github.com/Qexly/test2.git**](https://github.com/Qexly/test2.git) - получить репо из гитхаба на ПК

**git pull** – обновит наш локальный репо с облака github

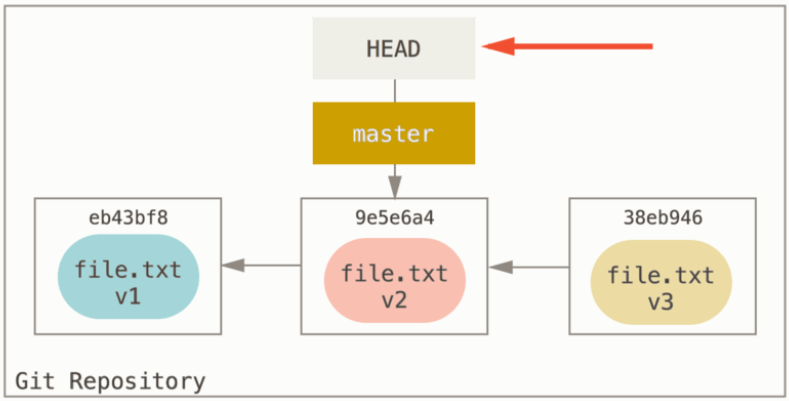
# Отмена изменений и возврат к другому коммиту

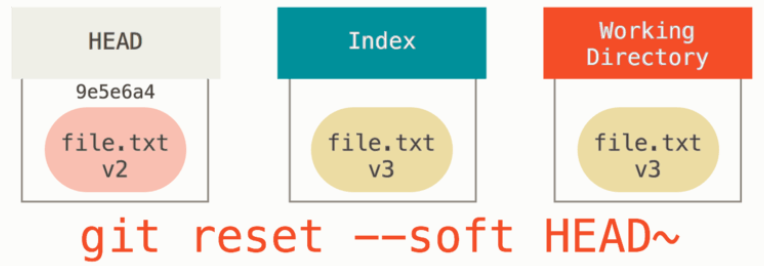
[https://ru.stackoverflow.com/questions/431520/%D0%9A%D0%B0%D0%BA-%D0%B2%D0%B5%D1%80%D0%BD%D1%83%D1%82%D1%8C%D1%81%D1%8F-%D0%BE%D1%82%D0%BA%D0%B0%D1%82%D0%B8%D1%82%D1%8C%D1%81%D1%8F-%D0%BA-%D0%B1%D0%BE%D0%BB%D0%B5%D0%B5-%D1%80%D0%B0%D0%BD%D0%BD%D0%B5%D0%BC%D1%83-%D0%BA%D0%BE%D0%BC%D0%BC%D0%B8%D1%82%D1%83](https://ru.stackoverflow.com/questions/431520/Как-вернуться-откатиться-к-более-раннему-коммиту)

**Git reset**

**git reset --soft HEAD~<n>**

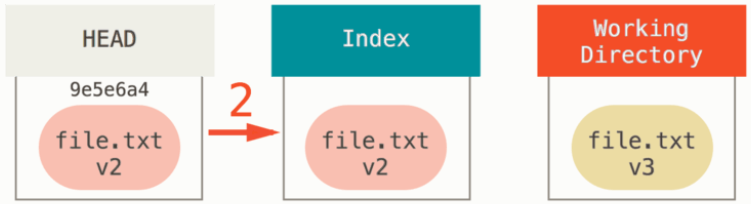
* Перемещает ветку, на которую указывет HEAD на n коммитов назад (комиты отменяются)
* Состояние индекса и рабочей области остается как при ласт комите:





**git reset [--mixed] HEAD~<n>**

То же самое, что и --soft, но теперь еще обновит и index содержимым того снимка, на который теперь указывает HEAD.



Это поведение по умолчанию, как если бы *git reset HEAD~*

**git reset --hard HEAD~<n>**

Вы отменили ваши последние n коммитов, результаты выполнения команд git add и git commit, а также все изменения, которые вы сделали в рабочем каталоге.

**git reset <file>** - (reset с указанием пути файла) удалить последние изменения из индекса, а потом можно и **checkout**

То есть эта форма записи (так как вы не указали ни SHA-1 коммита, ни ветку, ни опций --soft или --hard) является сокращением для git reset --mixed HEAD <file> которая:

1. Перемещает ветку, на которую указывает HEAD (будет пропущено)
2. Делает Индекс таким же как и HEAD (остановится здесь)

git reset eb43bf <file> - положит в индекс версию файла из комита eb43bf

**С помощью reset можно объединять комиты**

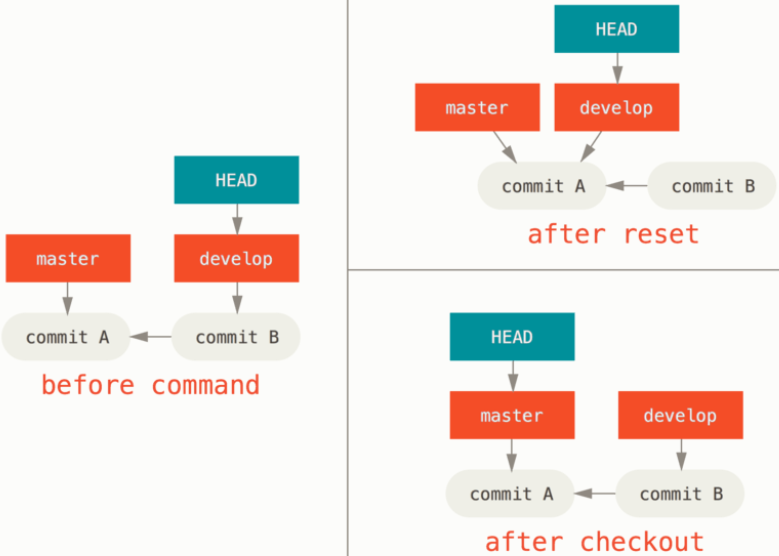
**Git checkout**

Как и reset, команда checkout управляет тремя деревьями Git, и также её поведение зависит от того указали ли вы путь до файла или нет.

**git checkout <branch> -** индекс и рабочая область становится как ласт комит в branch.

*git checkout [branch]* очень похожа на *git reset --hard [branch].* Отличия от reset:

* пытается выполнить в Рабочем Каталоге простые слияния так, чтобы все файлы, которые вы не изменяли, были обновлены в нем. С другой стороны, команда reset --hard просто заменяет всё целиком, не выполняя проверок.
* reset перемещает ветку, на которую указывает HEAD, а checkout перемещает сам HEAD так, чтобы он указывал на другую ветку:



**git checkout -- <file> -** (checkout с указанием пути файла)Отменить изменения, которые пока не внесены в индекс.

обновляет файл в индексе версией из коммита, но дополнительно она обновляет и файл в рабочем каталоге.

**git checkout <номер комита>** - перейти к комиту

**git log** – просмотреть коммиты

**gitk --all&** - посмотреть дерево комитов графически

# Ветки

**git branch** – посмотреть все существующие ветки (\* - текущая)

**git branch –v –** показывает ветки плюс последний коммит

**git branch <name> -** создать ветку

**git checkout <branchName>** - перейти в ветку. Когда переходим в другую ветку, но открытые файлы принимают состояние последнего (HEAD) комита в этой ветке

**git branch –m <name>** - переименовать текущую ветку

**git push --set-upstream origin <dev> -** добавить ветку в облако github чтобы делать push

Пример мержа:

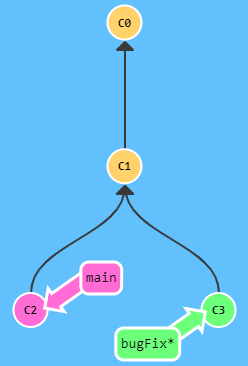
[https://git-scm.com/book/ru/v2/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D0%B2%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D1%8F](https://git-scm.com/book/ru/v2/Ветвление-в-Git-Основы-ветвления-и-слияния)

Переключение с одной ветки на другой без комита: [https://ru.stackoverflow.com/questions/327240/%D0%9F%D0%BE%D1%87%D0%B5%D0%BC%D1%83-%D0%BF%D1%80%D0%B8-%D0%BF%D0%B5%D1%80%D0%B5%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B8-%D0%B2%D0%B5%D1%82%D0%BE%D0%BA-%D0%BE%D1%81%D1%82%D0%B0%D1%8E%D1%82%D1%81%D1%8F-%D0%B2%D0%BD%D0%B5%D1%81%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5-%D0%B8%D0%B7%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F](https://ru.stackoverflow.com/questions/327240/Почему-при-переключении-веток-остаются-внесенные-изменения)

# Основы слияния

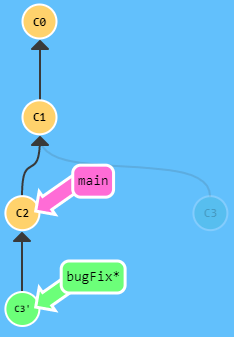
Чтобы слить ветку, нужно переключиться на ветку, в которую вы хотите включить изменения, и выполнить команду git merge <branch\_name>, где branch\_name имя ветки, из которой вливаем изменения

# Git Rebase

Есть две ветки. Обрати внимание, что выбрана ветка bugFix (отмечена звёздочкой).

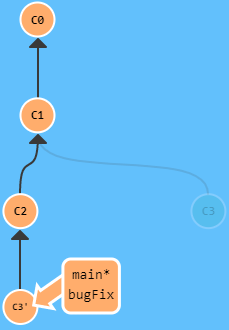
Хочется сдвинуть наши изменения из bugFix прямо на вершину ветки main. Благодаря этому всё будет выглядеть, как будто эти изменения делались последовательно, хотя на самом деле - параллельно.

Применим **git rebase main**:

Обрати внимание, что коммит С3 до сих пор существует где-то, а С3' - это его "копия" в ветке main

Единственная проблема - ветка main не обновлена до последних изменений. Это легко исправить:

1. Переключиться на main (git branch main)
2. git rebase bugFix

Так как main был предком bugFix, git просто сдвинул ссылку на main вперёд.

# HEAD

HEAD - это символическое имя **текущего выбранного коммита** — это, по сути, тот коммит, над которым мы в данным момент работаем.

HEAD всегда указывает на последний коммит из вашего локального дерева. Большинство команд Git, изменяющих рабочее дерево, начнут с изменения HEAD.

Обычно HEAD указывает на имя ветки (например, bugFix). Когда вы делаете коммит, статус ветки bugFix меняется и это изменение видно через HEAD.

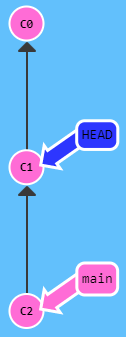
Отделение (detaching) HEAD означает лишь присвоение его не ветке, а конкретному коммиту.

Пример явного закрепления HEAD:

**git checkout C1**, где C1 – это хэш комита

# Относительный HEAD

* **git checkout main^** - переключиться на родителя ветки main:



* **git checkout HEAD~4** – переключиться на 4 коммита назад

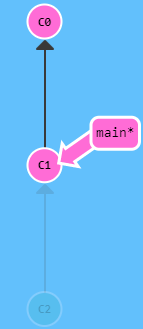
# Branch forcing

Можно напрямую прикрепить ветку к коммиту при помощи опции -f. Например, команда:  
git branch -f <branch\_name> HEAD~3  
Переместит (принудительно) ветку <branch\_name> на три родителя назад от HEAD:

# Отмена изменений в Git

git reset перенесёт ветку назад, как будто некоторых коммитов вовсе и не было.

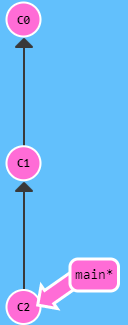
git reset HEAD~1:

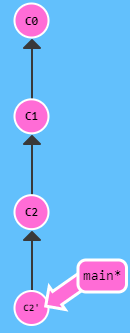
Git просто перенёс ссылку на main обратно на коммит C1.

Reset не сработает на удалённых ветках, которые используют другие пользователи. Для этого есть git revert:

git revert HEAD:

было:

стало:

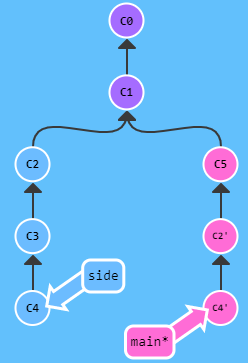
новый коммит C2' содержит изменения, полностью противоположные тем, что сделаны в коммите C2.

# Git Cherry-pick

git cherry-pick <Commit1> <Commit2> <...> - простой и прямолинейный способ сказать, что ты хочешь копировать несколько коммитов на место, где сейчас находишься (HEAD). cherry-pick поместит любой коммит сразу после HEAD (только если этот коммит не является предком HEAD)

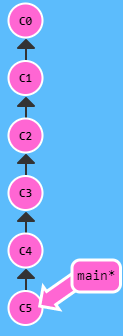
Пример:

git cherry-pick C2 C4:



Git rebase –i

Было:



*git rebase –I HEAD ~4:*

# 

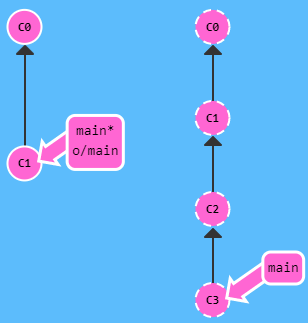
От 4-го предка создал ветку и расставил в ней коммиты в нужном порядке

# **Удалённые репозитории в Git (clone & fetch)**

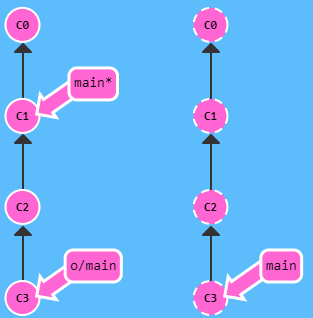
**git clone** - в реальной жизни - это команда, которая создаст локальную копию удалённого репозитория.

**git fetch**:

было:



Стало:



C2 и C3 были успешно скачаны в наш локальный репозиторий, и наша удалённая ветка o/main отобразила эти изменения соответствующим образом.

fetch делает:

* связывается с указанным удалённым репозиторием и забирает все те данные проекта, которых у вас ещё нет, при этом...
* у вас должны появиться ссылки на все ветки из этого удалённого репозитория (например, o/main)

Команда всего лишь скачивает все необходимые данные, но вам потребуется вручную слить эти данные с вашими, когда вы будете готовы.

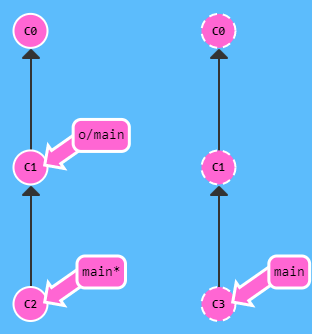
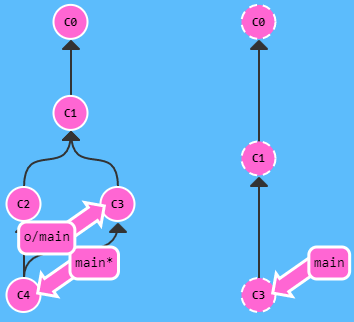
# Git Pull

Чтобы после fetch слить изменения с локальными ветками, можно:

* git cherry-pick o/main
* git rebase o/main
* git merge o/main

либо вместо fetch юзать pull.

Тут **git fetch; git merge o/main** и **git pull** сделают одно и то же:

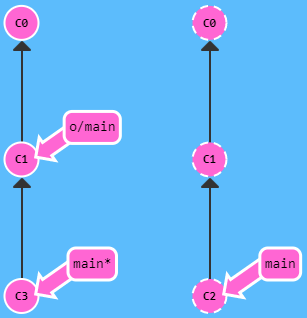
 

Git fakeTeamwork branch n

# Git Push

Команда git push отвечает за загрузку ваших изменений в указанный удалённый репозиторий, а также включение ваших коммитов в состав удалённого репозитория (противоположно pull).

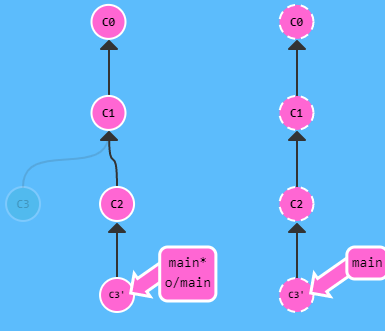
**Расхождения:**



Пуша не произошло. Всё потому, что команда git push не выполнилась успешно. Дело в том, что ваш последний коммит C3 основан на удалённом коммите C1. В свою очередь, удалённый репозиторий уже изменился под воздействием C2. Вот почему git отклонил ваш push.

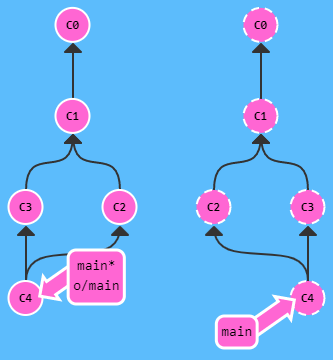
Можно воспользоваться командами:

*git fetch; git rebase o/main; git push*



Либо через merge:

*git fetch; git merge o/main; git push*



**Но можно короче!** Fetch заменить на pull либо pull –rebase (чтобы не через merge):

* ***git fetch; git rebase o/main; git push*****===*****git pull --rebase; git push***
* ***git fetch; git merge o/main; git push === git pull; git push***