# 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

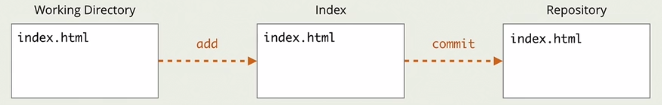
**git status** – взгляд со стороны git на файлы нашего проекта.

* Untracked files – те, которые не добавлены в index

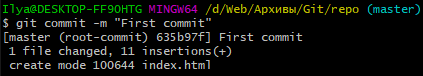
**git add <file>** – добавит файл в index

**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" \l "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/<repos)**> -** 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 log** – просмотреть коммиты

**git checkout -- <file> -** Отменить изменения, которые пока не внесены в индекс

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

**git reset <file>** - удалить последние изменения из индекса, а потом можно и **checkout**

**git reset --hard HEAD^1** – удалить ласт коммит и удалить его изменения из файла и индекса

**git reset --soft HEAD^1 –** удалить ласт коммит, но изменения из индекса не убирать и из файла тоже

**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:

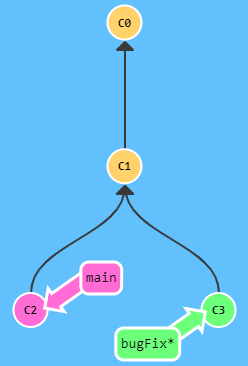
[https://git-scm.com/book/ru/v2/%D0%92%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-Git-%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D1%8B-%D0%B2%D0%B5%D1%82%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F-%D0%B8-%D1%81%D0%BB%D0%B8%D1%8F%D0%BD%D0%B8%D1%8F](https://git-scm.com/book/ru/v2/Ветвление-в-Git-Основы-ветвления-и-слияния)

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

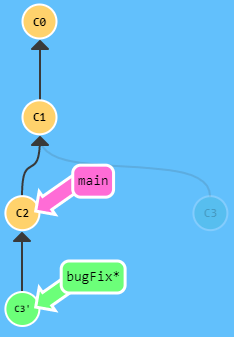
Чтобы слить ветку, нужно переключиться на ветку, в которую вы хотите включить изменения, и выполнить команду git merge <branch\_name>, где branch\_name имя ветки, из которой вливаем изменения

# Git Rebase

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

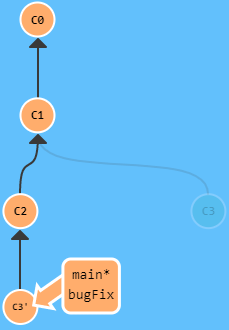
Хочется сдвинуть наши изменения из bugFix прямо на вершину ветки main. Благодаря этому всё будет выглядеть, как будто эти изменения делались последовательно, хотя на самом деле - параллельно.

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.

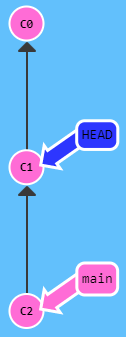
Отделение (detaching) HEAD означает лишь присвоение его не ветке, а конкретному коммиту.

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

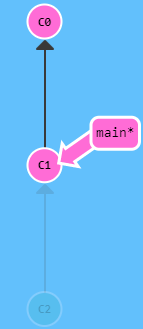
# Branch forcing

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.

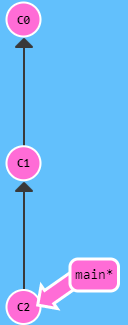
git reset HEAD~1:

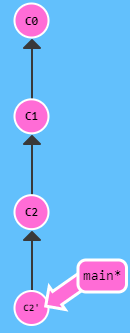
Git просто перенёс ссылку на main обратно на коммит C1.

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)

Пример:

git cherry-pick C2 C4:

