Гайд о том, как пользоваться git

*“Люди делятся на три типа: кто не делает бэкапы, кто делает бэкапы и кто делает бэкапы бэкапов.”  
Ли Си Цын, искусство написания кода в ночь перед сдачей.*

*128 г. д.н.э*

**Версия от 21.11.2022**

**Автор**: Штанов Евгений Юрьевич

Есть вопросы? Для связи со мной:

<https://t.me/Klavishnik0_o>

<https://vk.com/klavishnik0_o>

shtanov.klavishnik@yandex.ru

GIT

# Для кого и для чего

Итак, GIT. Что это такое и почему это нужно использовать?  
GIT - это система контроля версий. Основные её достоинства - простота, “дешевизна” и повсеместное использование - иметь навыки в данной системе необходимо любому разработчику, чтобы получить работу.

*Что такое «система контроля версий» и почему это важно? Система контроля версий -  это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться к предыдущему состоянию.*

Проще говоря - если вы внесли нежелательные изменения в структуру проекта (написали лишнюю строку или удалили файл), вы всегда можете откатиться к прошлому состоянию проекта.

Также данная система контроля версий позволяет **хранить ваши проекты на внешних бесплатных хранилищах (репозиториях), доступных везде.**

**САМОЕ ГЛАВНОЕ ДЛЯ ВАС!**

ИСПОЛЬЗУЯ СИСТЕМУ ВЕРСИЙ МОЖНО ВСЕГДА ВЕРНУТЬСЯ К СОСТОЯНИЮ, КОГДА ВСЕ РАБОТАЛО!

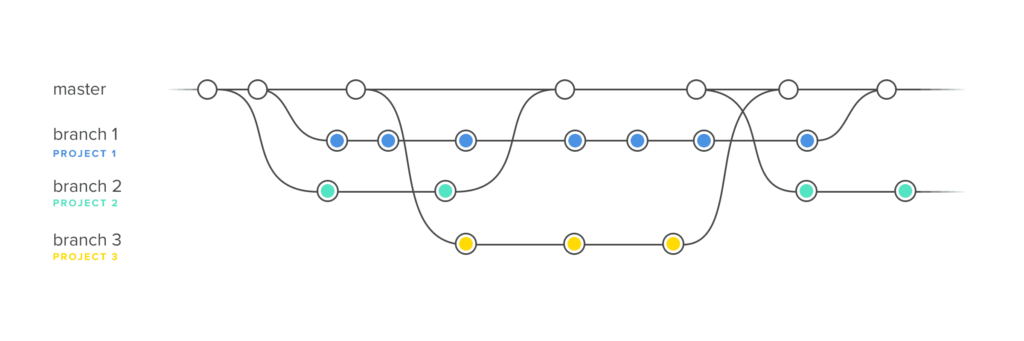
## Как это работает

GIT позволяет вести разработку проектов. Место, где хранится код вашего проекта называется **репозиторием**. Он может быть локальным - храниться только на вашем компьютере. Может быть внешним - есть множество сайтов, предоставляющих возможность вести репозитории (самые известные <https://github.com/> и <https://bitbucket.org/>). Внешние репозитории могут быть частными (private) и публичными (public). Частные репозитории доступны только для владельца и допущенных пользователей, публичные - всем.

В репозитории хранится ваша информация. Она может быть любой, но будем считать, что вы там храните код. Также там ведется история изменений вашего кода. Создается она не автоматически, а за счет так называемых **коммитов (commit).** Коммит - это зафиксирование состояние вашего репозитория - эдакая контрольная точка, как сохранение в компьютерной игре - вы всегда можете создать несколько сохранений, откатиться (загрузиться) на предыдущее.

GIT позволяет вести одновременную разработку проекта за счет системы **веток (branch).** Что это такое? Это когда вы от одного коммита (состояния кода) ведете несколько изменений кода параллельно. Т.е. вы можете одновременно реализовывать библиотеку в одной ветке и main в другой и они никак не влияют друг на друга. В конце вы можете объединить это вместе в один коммит .

Звучит сложно, но лучше всего это описывает такая картинка:



# Переход к практике

## Регистрация и создание внешнего репозитория

Если вы намерены “поднять” свой сервер git или использовать его локально, то данный шаг можно пропустить

В качестве примера будем использовать <https://github.com/>  
Регистрируемся там.

**ВАЖНО!** *С недавних пор данный ресурс отключил аутентификацию по логину и паролю. Теперь есть аутентификация по токену (их можно создать несколько штук [например для разных систем] он используется как пароль, но его можно отозвать в любой момент) либо по ssh ключу. В качестве примера будем использовать вторую*

Профиль на Гитхабе и все проекты в нём — ваше публичное портфолио разработчика, поэтому нужно завести профиль, если у вас его еще нет.

1. Зайдите на сайт https://github. com и нажмите кнопку Sign up.
2. Введите имя пользователя (понадобится в дальнейшей работе), адрес электронной почты (такой же, как при настройке Git) и пароль.
3. На почту придет код активации — введите на сайте.
4. Появится окно с выбором тарифного плана. Если вы пользуетесь Гитхабом для учёбы, то укажите, что профиль нужен только для вас и вы студент.
5. Опросы и выбор интересов можно пропустить. На этом всё — вы зарегистрировались и у вас есть собственный профиль.

## Устанавливаем ssh ключи

Заходим на сервер и терминале пишем следующее:

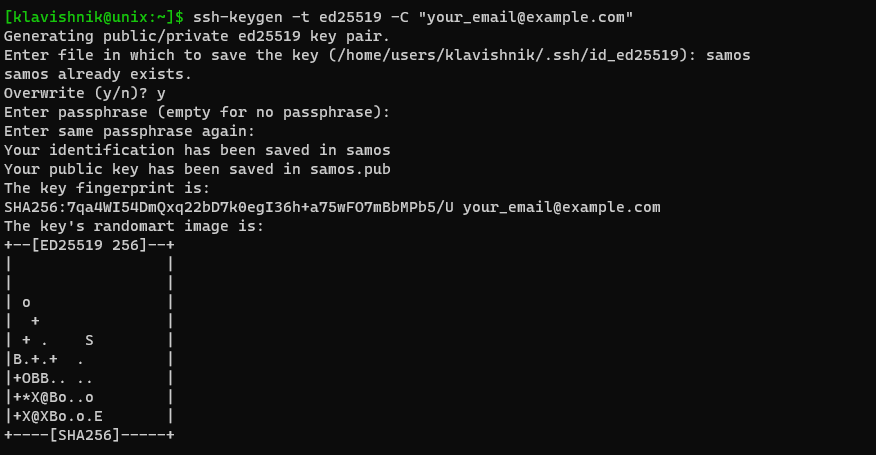
ssh-keygen -t ed25519 -C "[your\_email@example.com](mailto:your_email@example.com)"

С помощью данной команды мы создаем пару ssh ключей (про ассиметричное шифрование, RSA можно почитать отдельно или попытать вашего преподавателя).

После выполнения данной команды у нас откроется интерактивное окошко.

1. Первый запрос будет на имя файлов с ключами, которые создаст данная утилита. Предлагаю назвать samos
2. Ввод парольной фразы и её подтверждение пропустим.

На экране будет примерно такой вывод



После этого делаем

ls -la ~/.ssh/

У нас появились два файла



**samos** - это закрытый ключ. Его передавать нельзя, храним только на сервере.

**samos.pub** - это открытый ключ. Его можно передавать, поскольку раскрытие такого рода ключей на безопасность не повлияет.

**ВАЖНО!** Обычно файлики создаются по адресу ~/.ssh/

У нас на сервере они создаются в корне, т.е. по адресу ~/

Теперь мы должны записать наш **закрытый ключ (samos)** в локальный менеджер ssh ключей.

Проверим, запущена ли утилита:

eval "$(ssh-agent -s)"

Если в ответ терминал покажет надпись **«Agent pid»** и число — значит, всё ок, агент запущен.

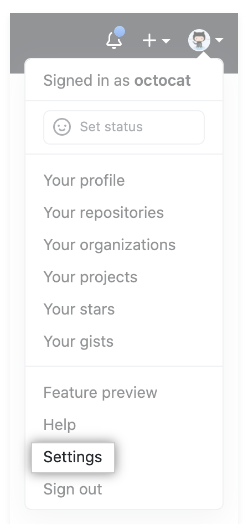
Теперь добавим наш ключ командой.

ssh-add ~/samos

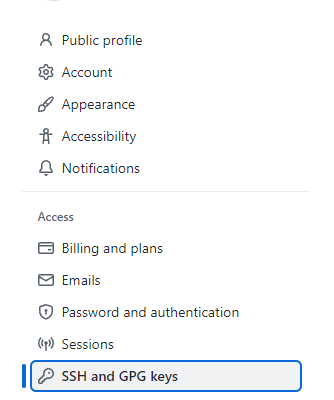
Теперь добавим **открытый ключ (samos.pub)** в профиль на гитхабе

Сделаем  
 cat samos.pub

Все что выведется на экран - копируем правой кнопкой мыши.

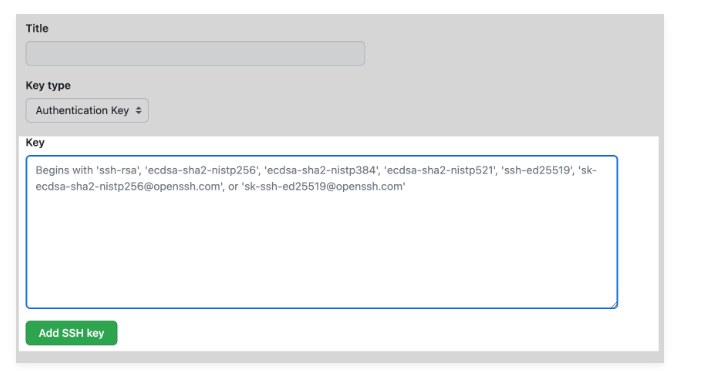
Переходим на сайт гитхаба и справа вверху нажимаем лкм на свой аватар и выбираем пункт Settings  


Далее переходим в раздел с SSH ключами



Нажимаем new ssh key

В поле внутри вставляем наш ключ



Поле title заполняем по собственному усмотрению, например samos.

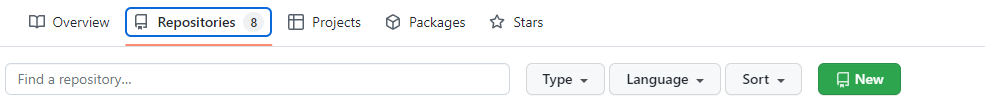
Нажимаем add ssh key и сохраняем.

У нас появится возможность заливать на репозитории гитхаба нашу информацию.

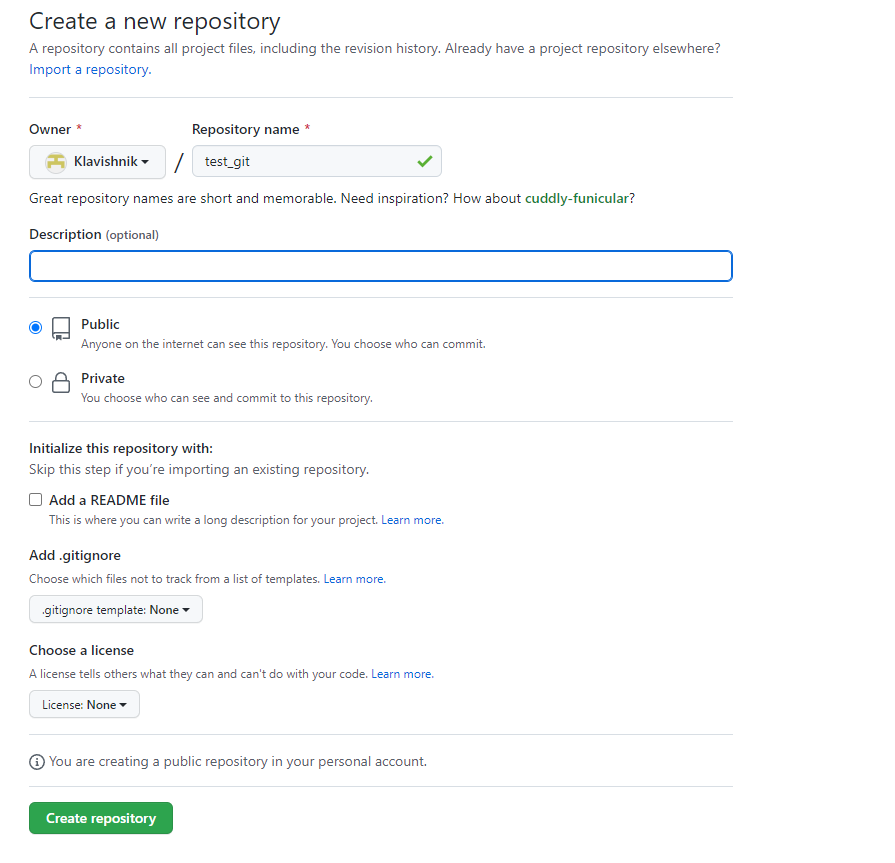
## Создаем внешний репозиторий

Переходим на страницу с личным профилем (нажимаем на аватар и первая строчка в списке).

Жмем на вкладку Repositories



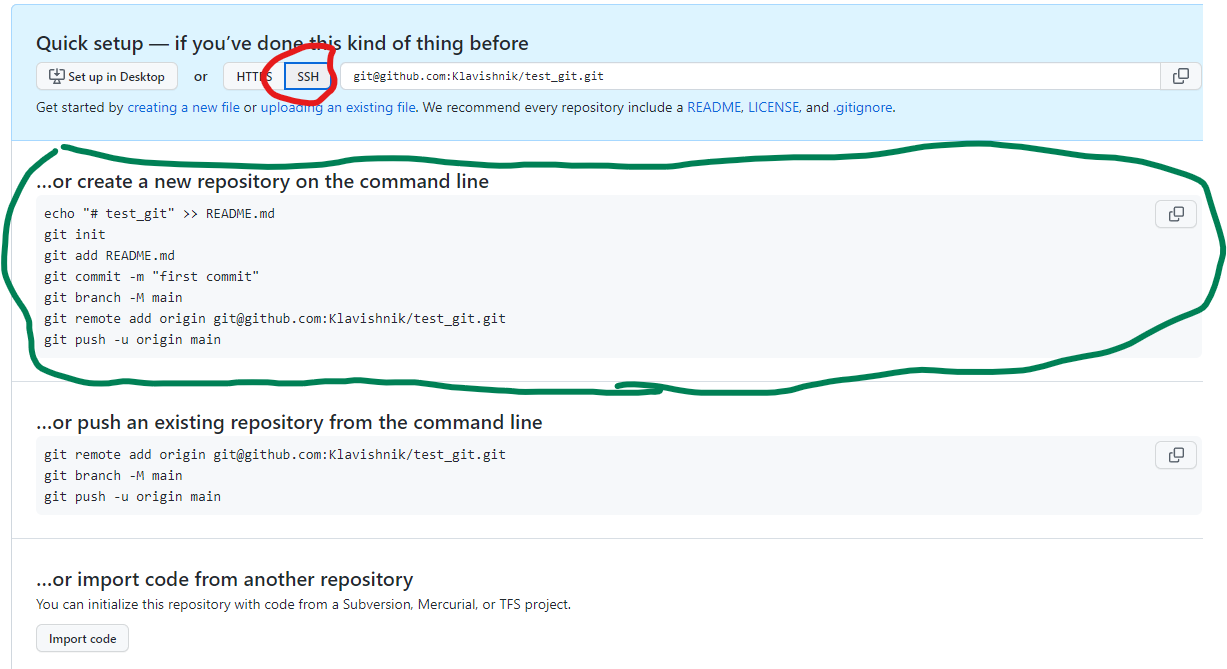
И на кнопку NEW справа



Заполняем информацию. Из обязательного только название. Нажимаем Create repository

Внешние репозитории могут быть частными (private) и публичными (public). Частные репозитории доступны только для владельца и допущенных пользователей, публичные - всем.

Попадаем на такую вкладку. Выбираем SSH (выделено красным)



Далее возвращаемся на сервер.

Сначала сделаем команду



Если выдало версию, то git установлен, можно использовать.

Создаем папку с нашим проектом, например test\_git

mkdir test\_git

И вставляем команды, которые даны выше (выделено зеленым).

Далее будет объяснения, что происходит.

## Инициализация репозитория и первый коммит

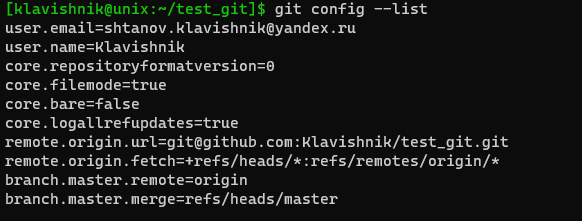
Сначала необходимо добавить ваши данные в систему (почта нужна для аутентификации и а имя будет отображаться в коммитах)

git config --global user.email "ВАША ПОЧТА с гит"

git config --global user.name "Ваш логин на гит"

Проверим

git config --global --list



Перейдем в нашу папку с будущим проектом

cd test\_git

Создадим файлик README.md с одной строкой test\_git

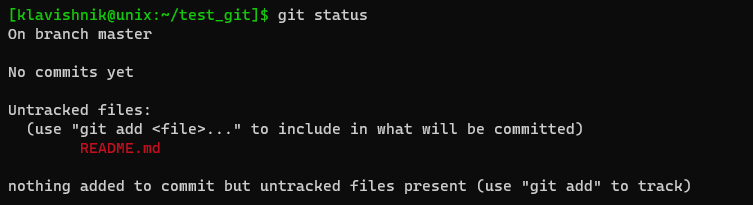
echo "# test\_git" >> README.md

**!ВАЖНО** *- .md - это расширение файлов языка разметки Markdown. README.md - это стандатное название файла заметки, которое будет отображаться внутри вашего репозитория, если зайти на веб версию гитхаба (покажу далее).*

Сделаем инициализацию репозитория командой

git init

Посмотрим состояние репозитория командой git status



GIT подсказывает, что был добавлен новый файл, но чтобы его отслеживала система контроля версий, нужно его добавить в отслеживаемые. Сделаем это.

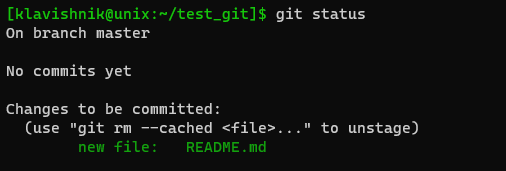
Добавим файлы, отслеживаемые системой контроля версий.   
Можно сделать так, указав конкретный файлик

git add README.md

Или так, добавив всю папку

git add .

Посмотрим, что теперь пишет git status



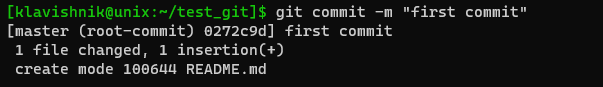
У нас есть изменения, которые нужно закоммитить.

Для этого пишем следующее

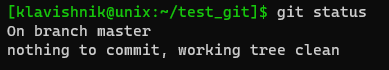
git commit -m "first commit"

Поздравляю! Мы сделали первый локальный коммит.Т.е сделали контрольную точку и зафиксировали состояние проекта. При коммите необходимо писать краткое описание коммита. Это и делается через флаг -m. Данный комментарий потом можно будет увидеть в истории коммитов. Нужно это для того, чтобы быстро найти коммит, в котором вы сделали определенные изменения.

Можно не использовать флаг -m, а написать git commit. Тогда у вас откроется стандартный для системы текстовый редактор (nano или vim) и в нем нужно будет дать описание коммита



Ну видно, что один файл изменился, какой это файлик был



Новых веток пока создавать не будем.

Попробуем залить наши изменения на внешний репозиторий

Для этого сначала обновим список внешних репозиториев на этой локальной машине

git remote add origin [git@github.com](mailto:git@github.com):USERNAME/GIT\_NAME.git

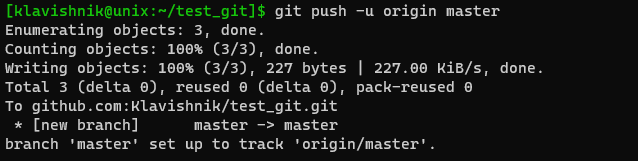
где

USERNAME - имя пользователя на github

GIT\_NAME - название репозитория

И зальем изменения

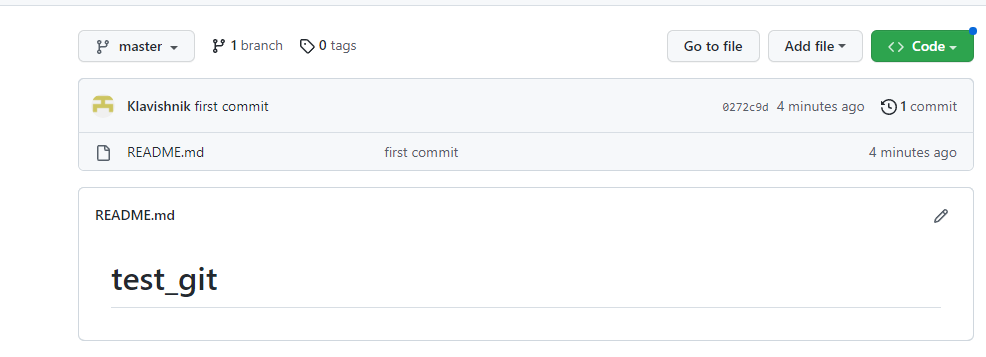
git push -u origin master



Примечание: если вы выполняли все указания гитхаба при создании репозитория, ваша основная ветка будет называться main вместо master, соответственно здесь и далее необходимо будет заменять master на main.

Видно, что на репозиторий все улетело.

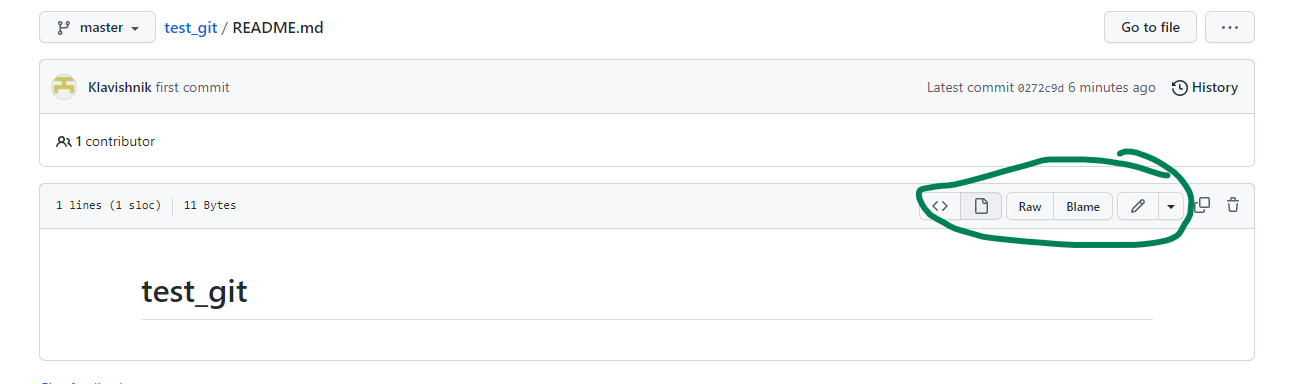
Пойдем на репозиторий, посмотрим что произошло



Видно, что у нас в списке файлов появился один файлик README.md

По традиции в данном файлике должно содержаться описание проекта / инструкция по использованию. Гитхаб ищет данный (именно с этим именем) файлик в корне репозитория и сразу отображает его в веб версии репозитория.

Если щелкнуть на сам файл README.md (рядом с ним еще ярлык файла), то мы сможем посмотреть исходный код файла. Сделаем это:

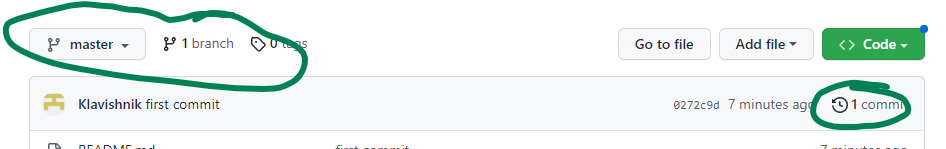


Если нажать на RAW, то можно увидеть исходное содержимое файла в “сыром” виде.

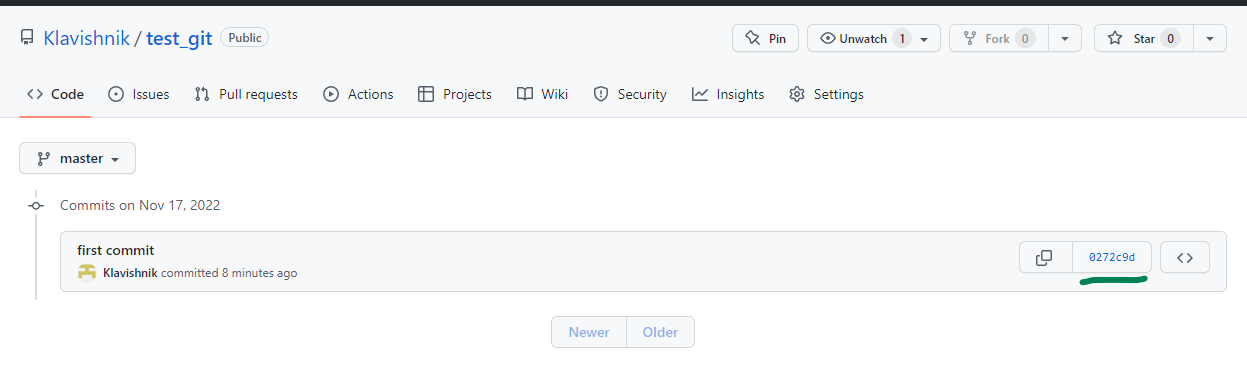
Если нажать на карандаш - то можно править файлик прямо на репозитории.

О синхронизации изменений в следующих главах.

Вернемся обратно



Здесь можно смотреть все ветки и историю коммитов.

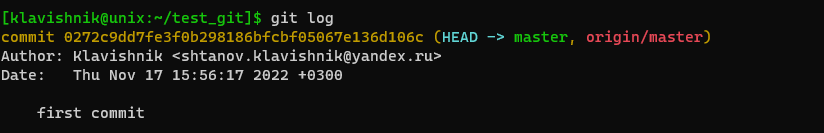


Это история изменений репозитория на данной ветке. Можно посмотреть подробнее, если щелкнуть на коммит. Также показывается сообщение, с которым мы коммитили код. (first commit)

Справа есть хэш коммита, он нужен, чтобы сразу можно было на него переключиться.

Как все это сделать в терминале? Как посмотреть историю коммитов?

git log



Как посмотреть ветки?

git branch



Звездочка укажет на ту, на которой мы сейчас находимся

Как посмотреть все наши действия с репозиторием?

git reflog



Здесь будут указываться изменения репозитория, осуществляемые вами во всех ветках. Например, добавление коммита, переход на другую ветку, обновление репозитория и так далее.

# Коммитим, пушим и откатываемся

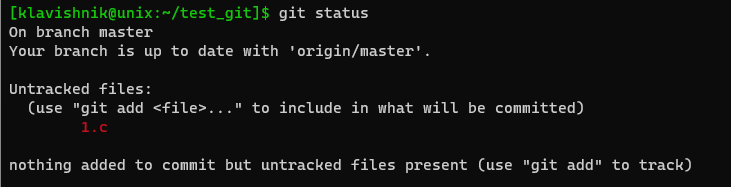
## Коммиты

Итак, попробуем покоммитить наш проект.

Создадим в нашем проекте файл с именем 1.с и и заполним в нем пару строчек, например, подключим библиотеки.



git status

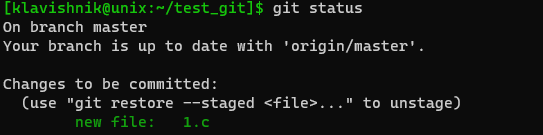


Добавим файл 1.с в индекс

git add 1.c

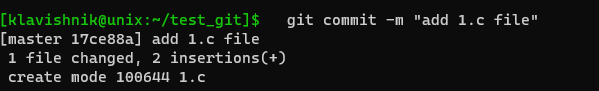
Посмотрим, что вышло

git status

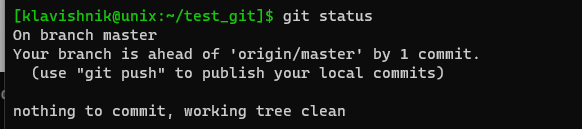


Коммитим

git commit -m “add 1.c file”



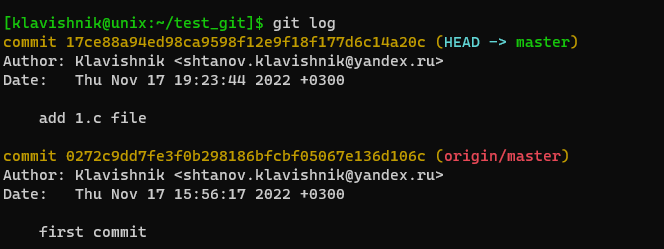
git status



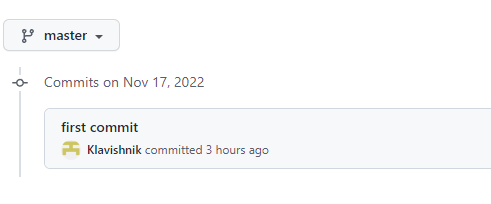
**Здесь написано, что коммитить нечего - все изменения репозитория зафиксированы, но наш локальный репозиторий опережает внешний репозиторий на один коммит.**

Проверим

Тут два коммита



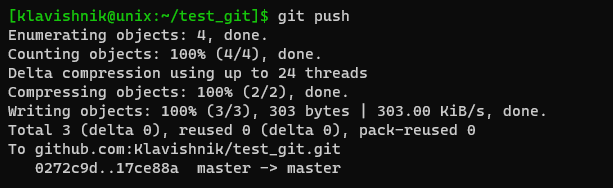
А тут один

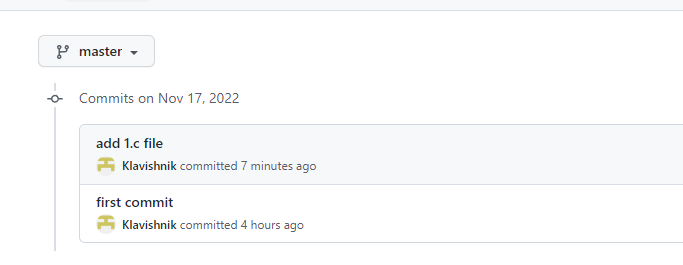


На сервере ничего нового

Запушим на сервер

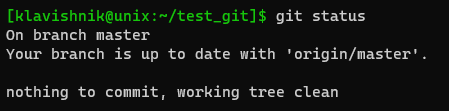
git push





Залилось на удаленный репозиторий и коммит наш отобразился в истории.

git status



Все обновлено, в том числе и на сервере.

## Откаты (состояния репозитория)

Сохраняться мы научились, а как загружаться?

А вот это правильный вопрос (с) потому что ответов на него даже больше, чем два.

Итак, рассмотрим две основные ситуации:

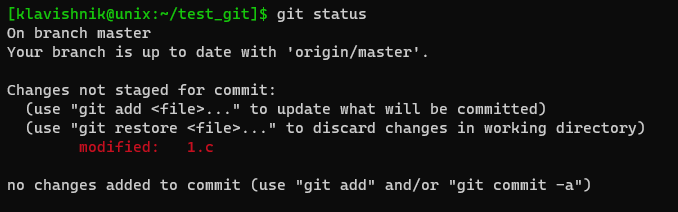
### Нам нужно отменить все изменения в репозитории (вернуться к состоянию последнего коммита)

Допустим вы что-то делали, оно перестало работать и вы хотите вернуться к последнему состоянию (сделать загрузку быстрого [последнего] сохранения).

Допустим я чуть подправлю файл 1.с

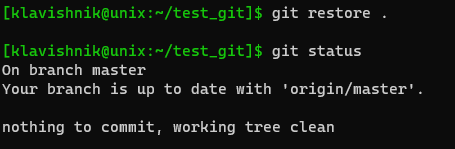


И хочу отменить эти изменения (т.е. удалить последнюю строку)

Состояние репозитория у меня не закоммичено  


Для этого делаем  
git restore . для отката всей папки

git restore file\_name для отката конкретного файла



Естественно в истории коммитов ничего отображаться не будет.

### Нам нужно вернуться к определенному коммиту

Если вы долго что-то делали, успели несколько раз зафиксировать состояние репозитория и поняли, что ~~ваша жизнь не имеет смысла~~ разработка зашла немного не туда и вам нужно “загрузиться” на пару “уровней раньше”, то нужно понять следующее:  
  
Даже если вы ошиблись - **удалять коммиты / ветки не нужно!**

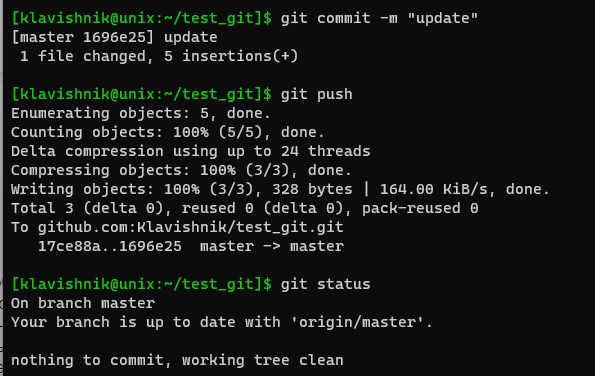
История изменений в гит - **священна!**

Если вы ошиблись, то алгоритм следующий:

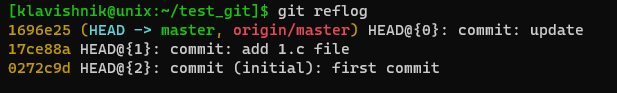
1. откатываемся на конкретный коммит, где все нормально;
2. от него ведем разработку в новой ветви.

Система git устроена таким образом, что создание новой ветки обходится достаточно “дешево” для системы, поэтому механизм ветвления можно использовать часто (в рамках разумного).   
Повторю, что ветви идут параллельно друг другу и изменения в них не отражаются в других ветвях. Чтобы объединить ветки, нужно сделать специальную операцию - **запрос на вытягивание (pull request)**, случайно сделать его не выйдет.

Итак, что мы делаем.  
Во-первых, я немного поменяю файлик 1.с и зафиксирую изменения и отправлю их::



А теперь я хочу откатиться на 2 коммит. Как это сделать?   
 git reflog

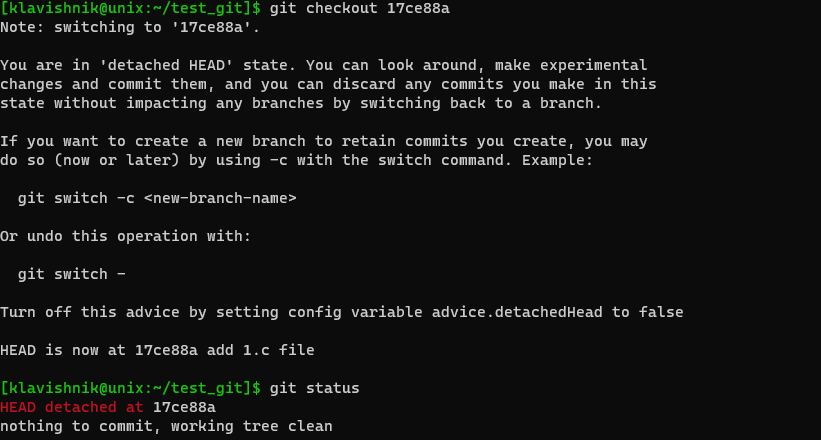


Перейдем на коммит HEAD@{1}

Для этого используем команду

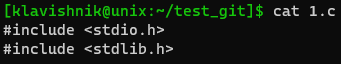
git checkout commit

Вместо commit подставляем хэш (цифры из желтого первого столбца)



Переключились. Проверим

cat 1.c



Да, все верно.

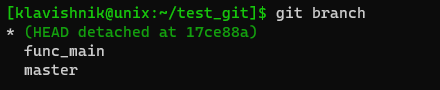
А что теперь? А теперь создаем новую ветку и продолжаем вести разработку от этого коммита:

Создадим новую ветку с названием func\_main

git branch func\_main

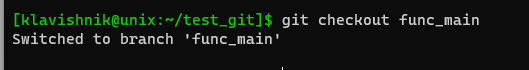
Проверим

git branch



Переключаемся на новую ветку

git checkout func\_main



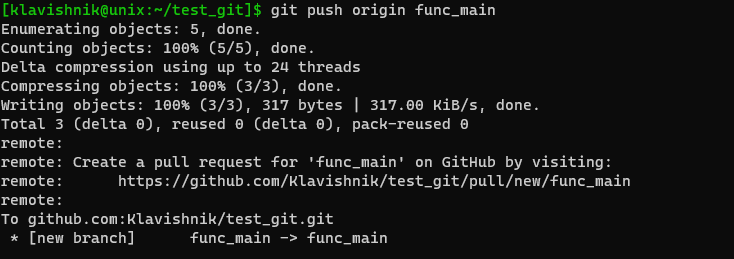
Делаем изменения в 1.с

Коммитим и загружаем на внешний репозиторий

git add 1.c

git commit -m "added func"

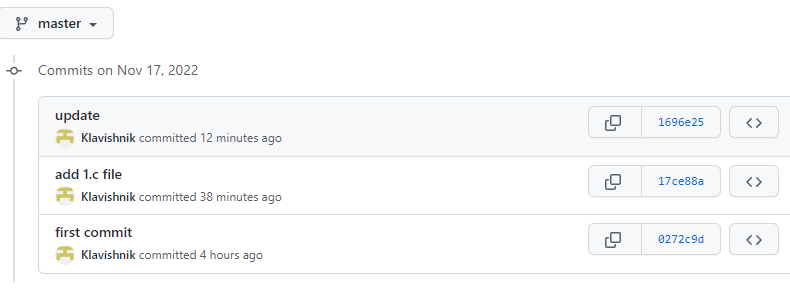
git push origin func\_main



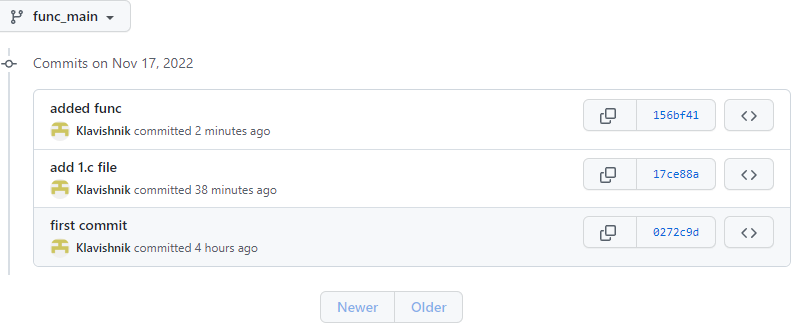
Видно, что на репозитории появилась новая ветка func\_main

Посмотрим, что у нас на веб версии репозитория

Ветка master



Ветка func\_main (новая)



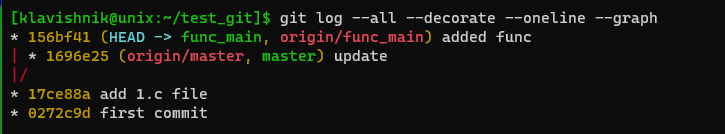
Обратите внимание на цифры справа - это хэш коммита. Видно, что первые два (снизу) совпадают, а третий отличается - собственно, мы откатились до второго коммита и сделали третий уже новым.

# Вывод истории в виде графа

Для наглядного отображения истории вашего репозитория, его можно вывести в виде графа.

Выполним:

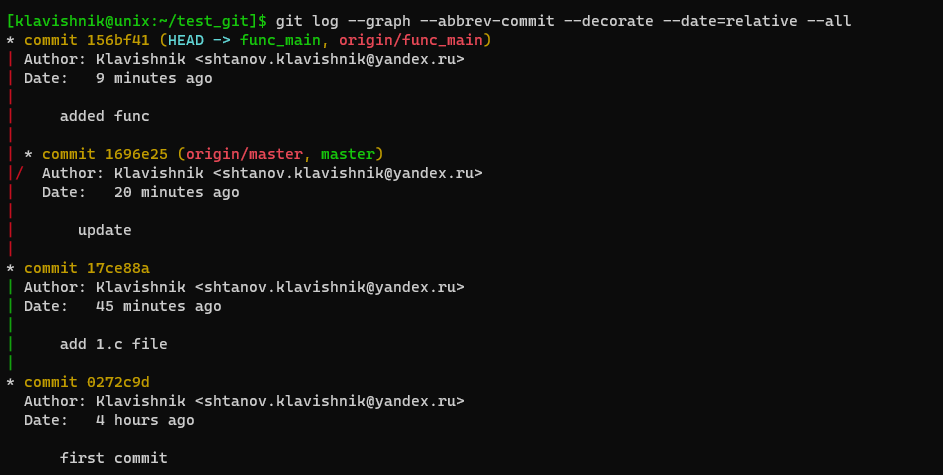
git log --all --decorate --oneline --graph



 Слева можно видеть, как у нас расположились ветки (**|**) и коммиты (**\***)

А вот так покрасивее

git log --graph --abbrev-commit --decorate --date=relative --all



# Работаем с существующим внешним репозиторием из нескольких клиентов

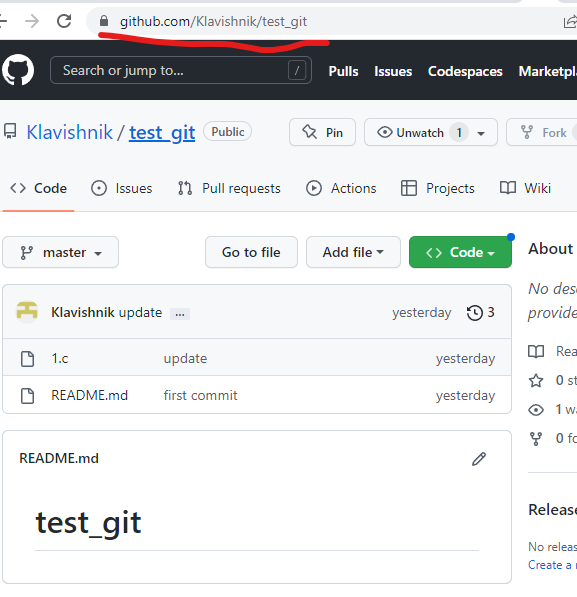
## Клонируем существующий репозиторий (git clone)

Итак, у нас есть существующий репозиторий, который мы, например, вели из дома с нашей домашней машины.   
Мы пришли в универ и нужно скачать наши изыскания, чтобы показать преподавателю.  
Что делать?

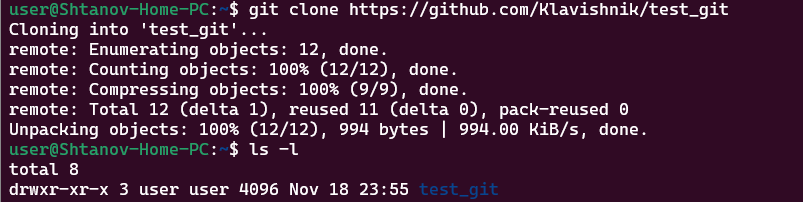
Для этого есть команда

git clone <название репозитория>

Название можно взять тут



Выполним



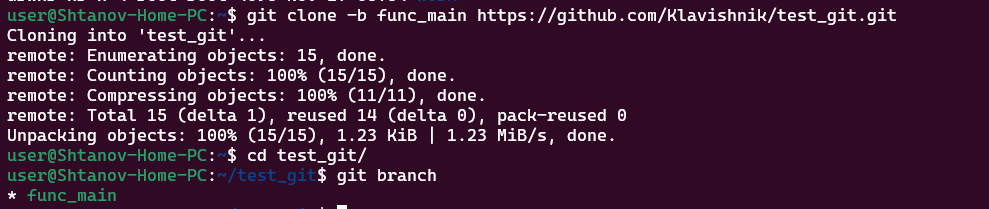
Видно, что все скачалось.

Если хотим скачать другую ветку, то можно использовать такие флаги:

git clone -b [название ветки] <ссылка до репозитория>

Пример

git clone -b func\_main https://github.com/Klavishnik/test\_git.git



Собственно, у нас есть локальная копия репозитория. Её можно менять, можно коммитить, но только **локально.** Ввиду того, что скачали репозиторий по **ссылке https** при попытке **запушить изменения** появится **ошибка авторизации!**

****

## Клонируем и пушим в существующий репозиторий

Для работы с внешним репозиторием с другого компьютера нужно пройти этап авторизации.

Чтобы пройти авторизацию с другого устройства, необходимо по аналогии с первым компьютером обеспечить наличие пары ключей и загрузить публичный ключ в настройки своего профиля на гитхабе. Здесь возможно два принципиально различных подхода:

1. Создать абсолютно новую пару ключей для нового устройства (см. “Устанавливаем ssh ключи”);
2. Копировать уже имеющиеся ключи с первого устройства на новое.

У обоих подходов есть свои плюсы и минусы, однако 1-й является более общепринятым так как обеспечивает гораздо большую степень безопасности. По этой причине, для подключения нового устройства рекомендуется повторить шаги, описанные в “Устанавливаем ssh ключи”.

Однако, если по какой-то причине вы решили воспользоваться 2-м способом, вы можете сделать это, например, при помощи команды scp:

scp username@name\_server1:path/to/file path/local/pc

Выполнять эту команду нужно на машине, КУДА вы хотите скачать файлик.

server1 - имя сервера откуда вы хотите файлы скопировать

Через двоеточие указывается путь до файла и его название

Пример

scp user@samos:~/samos ~/.ssh

Потребуется ввод пароля

После получения ключа добавляем его по аналогии с предыдущим разделом

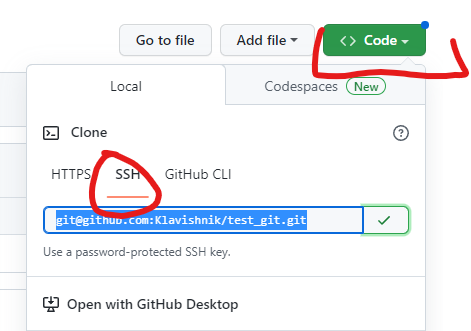
Проверим, запущена ли утилита:

eval "$(ssh-agent -s)"

Если в ответ терминал покажет надпись **«Agent pid»** и число — значит, всё ок, агент запущен.

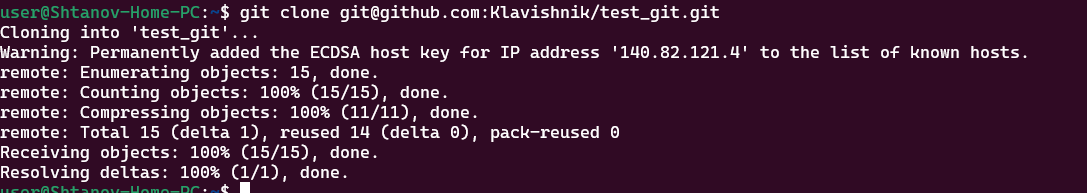
Теперь добавим наш ключ командой.

ssh-add ~/samos

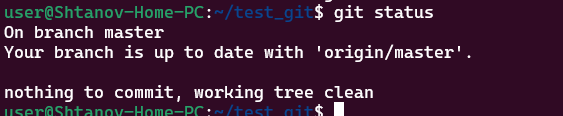
После добавления ключа нового устройства скачаем репозиторий через ssh  


Скачаем master ветку

git clone [git@github.com](mailto:git@github.com):Klavishnik/test\_git.git



Делаем git status



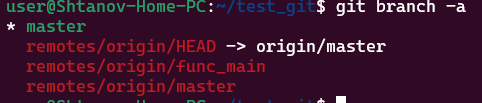
Мы на ветке master и она синхронизирована с веткой на удаленном репозитории

Попробуем переключиться на ветку func\_main и что-нибудь в неё залить

Сначала найдем эту ветку

Посмотрим все ветки

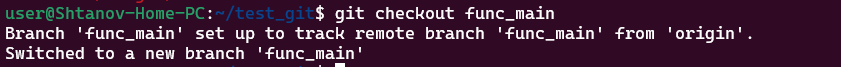
git branch -a



Напоминаю origin - название нашего внешнего репозитория (remotes намекает), которое мы дали при синхронизации

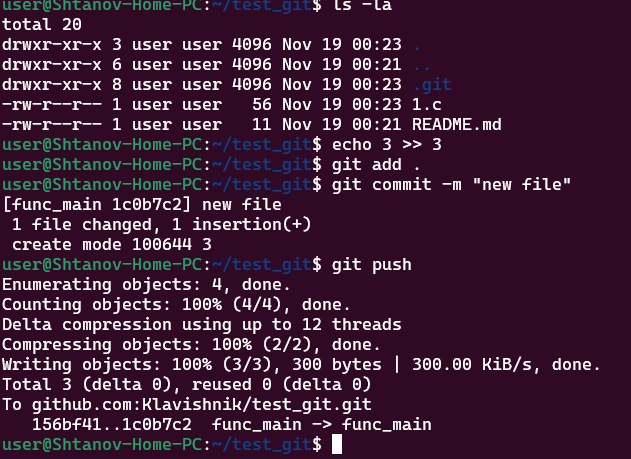
Переключаемся на ветку

git checkout func\_main

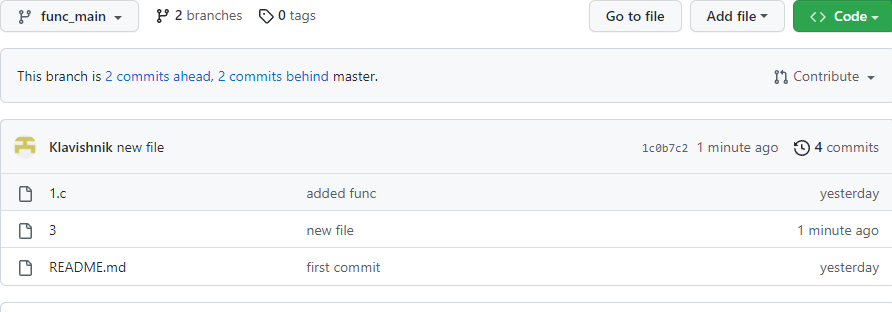


Ветка func\_main обновлена до состояния внешнего репозитория (не отличается от него). Мы успешно переключились на эту ветку.

Поменяем что-нибудь



Я добавил новый файл 3 и успешно запушил его в внешний репозиторий



Вот он появился

#### 

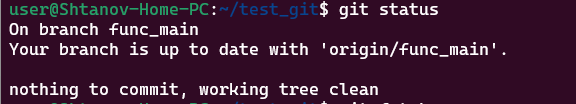
# Синхронизация изменений репозитория на разных клиентах.

## Синхронизация при работе в одиночку

Допустим ваш репозиторий живет на двух машинах. На обеих он уже склонирован.   
Например, на домашней машине вы залили изменения на гитхаб, пошли в универ и там хотите обновить папку проекта. Вопрос как?

git status

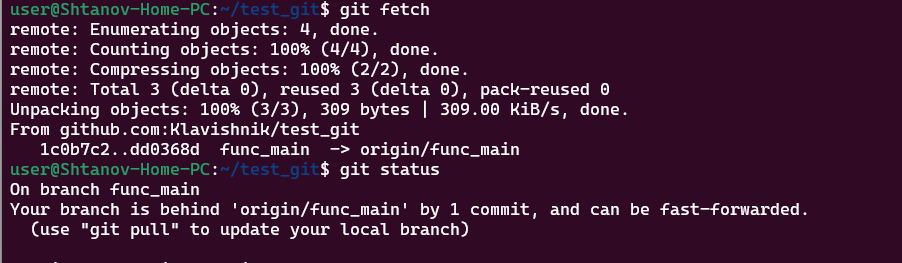
Покажет, что все обновлено. Но это не совсем верно.



Синхронизируемся с сервером командой

git fetch

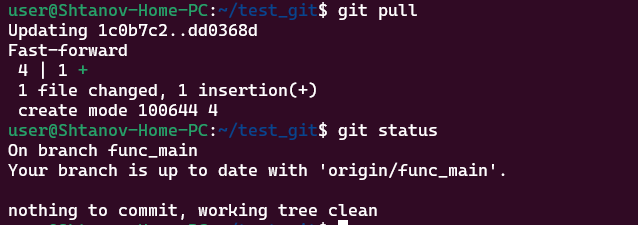
И посмотрим вывод git status



Система подсказывает, что наш репозиторий “опаздвает” на один коммит и сама подсказывает, что нужно выкачать изменения командой git pull

Делаем

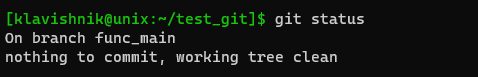
git pull



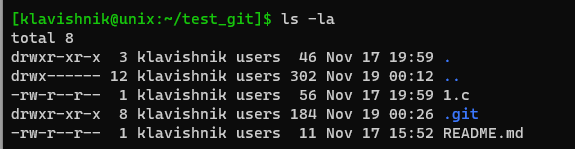
## Синхронизация при работе в команде

Рассмотрим ситуацию, когда ваш коллега запушил в эту же ветку что-то пока вы работали и вас об этом не предупредил.

Перейдем обратно на сервер



Мы на той же ветке



Нового файлика нет

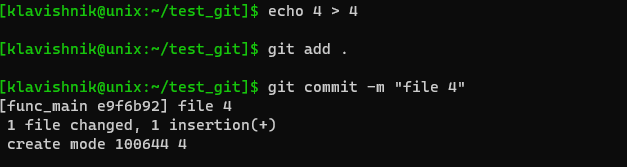
Попробуем создать новый файл 4

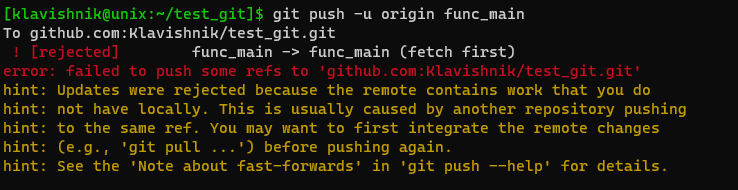
echo 4 > 4

git add .

git commit -m “file 4”

git push -u origin func\_main





Конфликт

git подсказывает, что причина в том, что на удаленном репозитории есть изменения, которых нет на локальной машине. Это могло произойти, например, когда ваш коллега одновременно запушил в эту же ветку что-то пока вы работали.

**!Поэтому, прежде, чем залить что-то на репозиторий, нужно выкачать последние изменения!**

Нужно посмотреть, не приведет это к конфликтам

Делаем

git fetch

Так получим последнюю информацию о изменениях

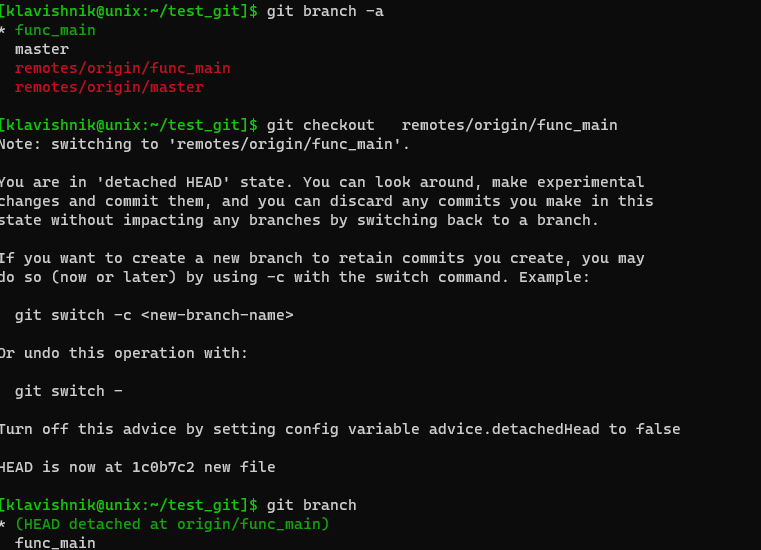
И осмотримся, что там на репозитории натворили без нас

git branch -a

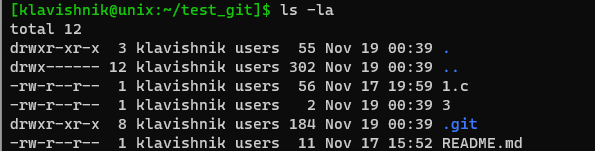
Переключимся на внешнюю ветку. **На ней мы находимся в режиме чтения и лучше ничего не менять**

git checkout remotes/origin/func\_main

git branch

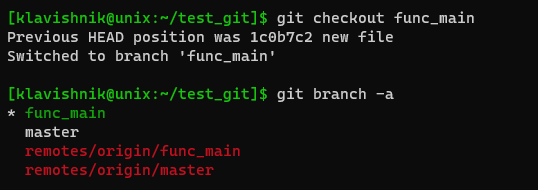


Смотрим, что изменилось



Ну вроде ничего критичного. Эти изменения наш код не затрагивают, поэтому возвращаемся обратно на нашу ветку

git checkout func\_main

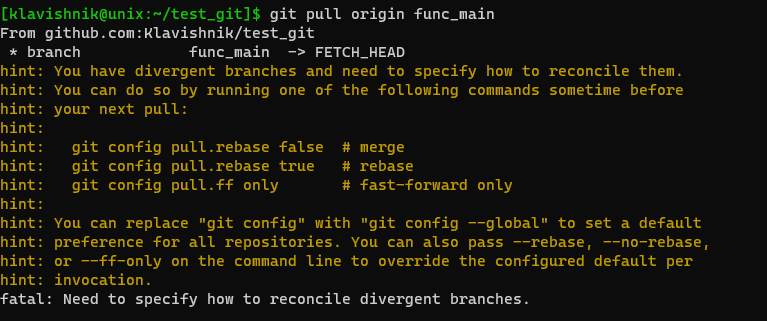


И делаем

git pull

Так мы скачаем все изменения и объединит с нашими локальными

git pull origin func\_main



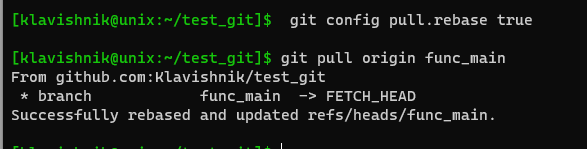
Ошибка

Это произошло, потому что мы поменяли нашу папку и скачали изменения из репозитория. Они расходятся и нужно их синхронизировать.

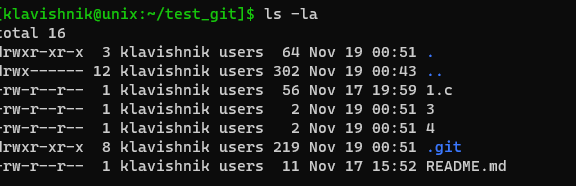
Мы уже посмотрели, что изменения внешнего репа не влияет на локалку, поэтому сделаем merge - сольем все изменения вместе.

git config pull.rebase true - таким способом мы сделали настройку для ВСЕХ таких будущих конфликтов. Т.е. при git pull у нас все будет объединяться

git pull origin func\_main

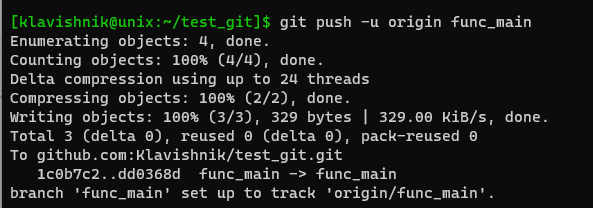


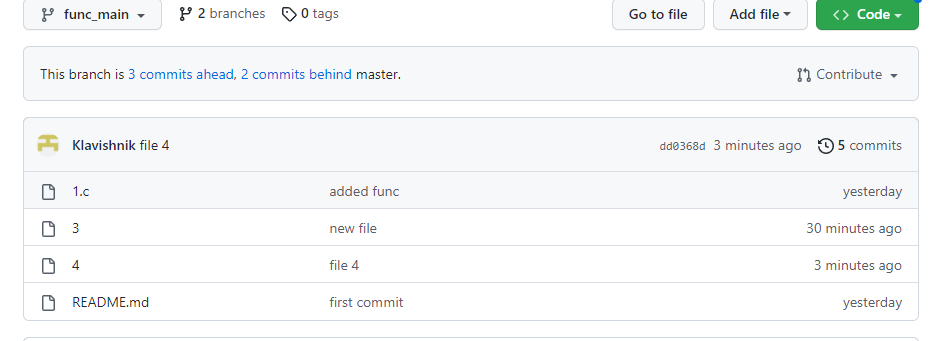
Вот что из этого вышло



Зальем наши изменения на репозиторий

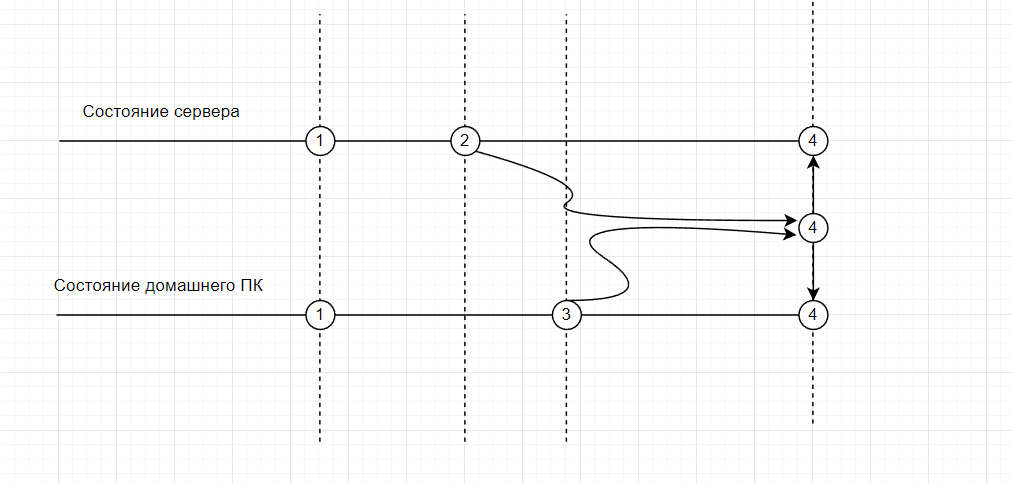
git push -u origin func\_main





Наши ветки синхронизированы

А теперь еще раз, что же у нас произошло, но в картинках



Итак, мы склонировали репозиторий с сервера в состоянии (1).   
Потом начали менять наши файлы локально и пришли в состояние (3).

В это время на сервер были загружены данные (2)

Мы объединили состояние (2) и (3) в состояние (4) и отправили на сервер.

# 

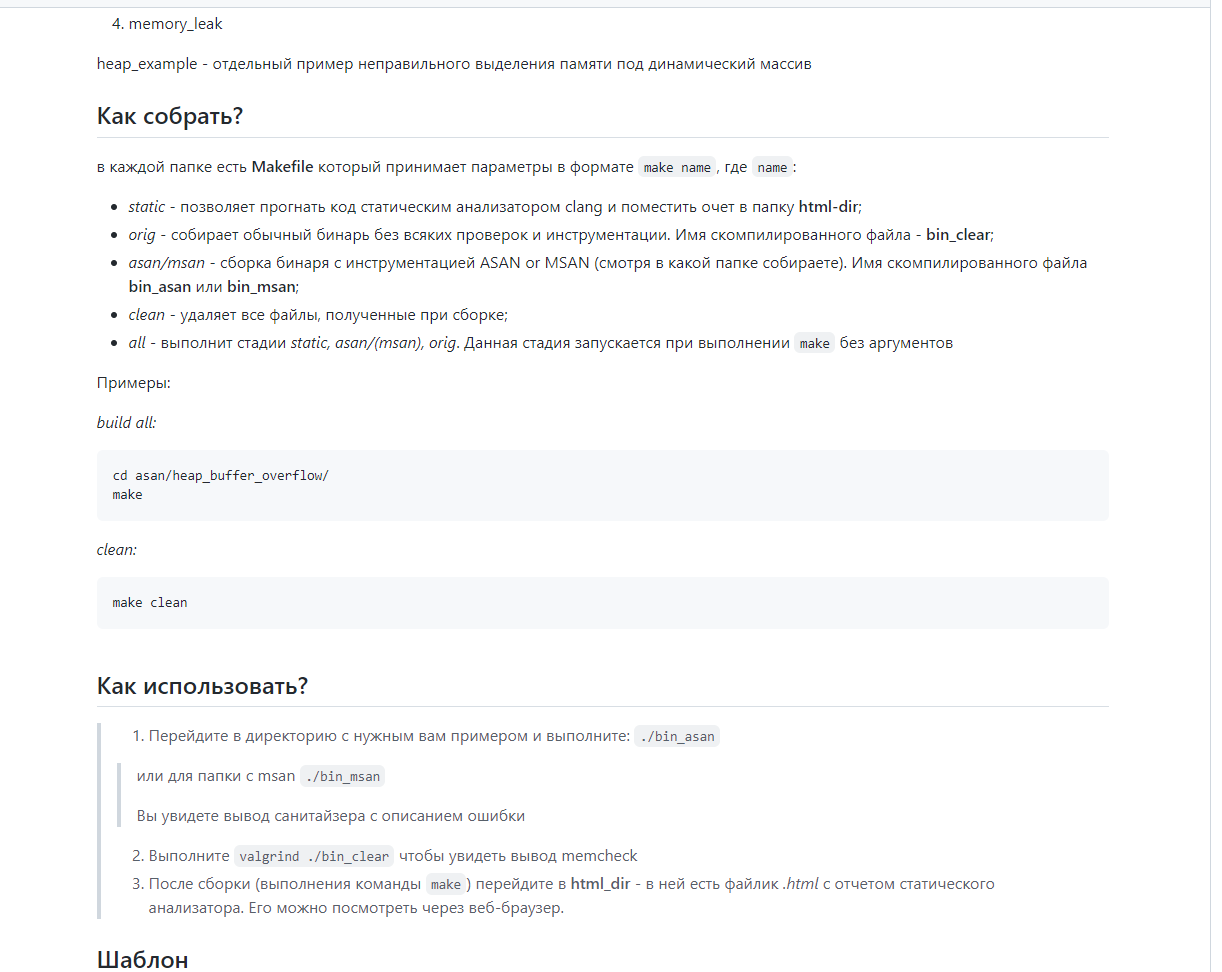
# 

# 

# Markdown

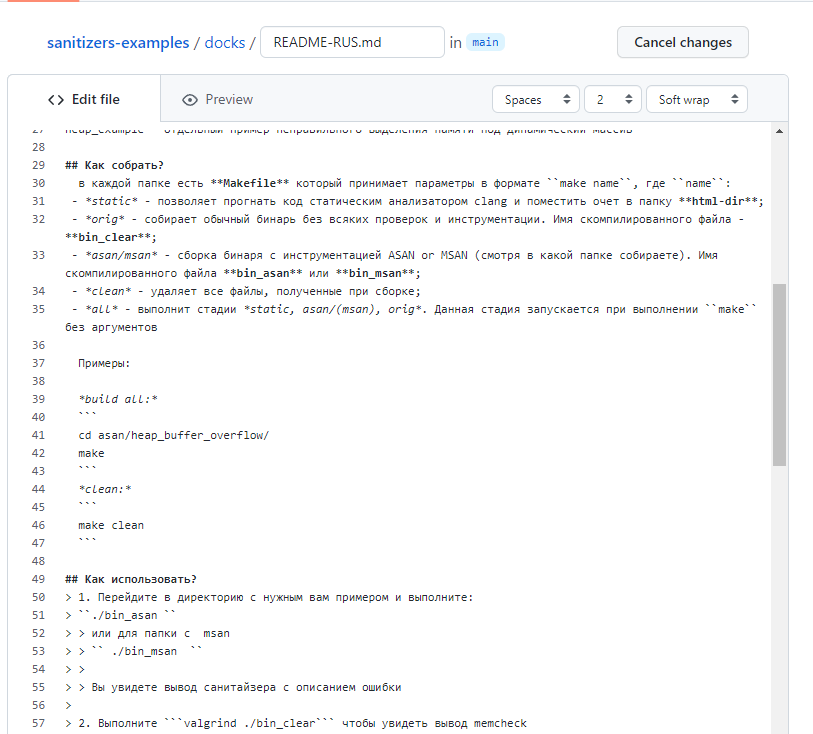
**Markdown -**

чтобы ваш README.md (и описание репозитория на веб морде) выглядело примерно так



можно использовать язык разметки **markdown**

В сыром виде выглядит примерно вот так



Как сделать также?

Смотрите здесь

<https://www.markdownguide.org/basic-syntax/>

Работает по аналогии с HTML, даже проще

# 

# 

# Ссылки

<https://git-scm.com/book/ru/v2>

Официальная книга - руководство по git от **создателей** и **полностью на русском!**

[**https://www.atlassian.com/ru/git/tutorials/**](https://www.atlassian.com/ru/git/tutorials/) **Примеры**

**А здесь интерактивный симулятор данной системы**

https://learngitbranching.js.org

https://git-school.github.io/visualizing-git/

**Markdown Синтаксис**

https://www.markdownguide.org/basic-syntax/

# Советы

Есть GUI версии гит в том числе под Windows

Искать тут <https://git-scm.com/downloads/guis/>  
От себя советую SourceTree (для Bitbucket) или GitHub Desktop (для github)

# Выводы

Ну пока собственно и все

Для самообразования нужно посмотреть систему pull request’ов и merge ветвей, но это оставляю на самостоятельное изучение