**Гайд для новичков**

[Ссылка на оригинал этого дока](https://docs.google.com/document/d/1TwcStaJI6kREz8_xTo-8QkP9QW3c92557TozrZ225UA)

## Что читать?

**Истинно идеального нет ничего. C’est la vie.**

**Идеально для начинающих:**

* [М. Уэйт, С. Прата, Д. Мартин – Язык программирования С.](http://storage.ded32.net.ru/Lib/TX/CPrimerPlus.pdf) (небольшая и клевая, но устаревший стиль примеров).
* [С. Прата, "Язык Си, лекции и упражнения (6 изд.)](http://gofile.me/2fK55/pwzQsGkOJ) – расширенный и более современный вариант предыдущей книги (но сильно больше по объему).

**Для что-то знающих** – параллельно еще книга Кернигана и Ритчи “Язык С” (но там устаревший стиль кода!):

* [Б. Керниган и Д. Ритчи. Язык С.](http://web.archive.org/web/20190712230028/https://ikt.muctr.ru/images/info/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf)

Более развернутый список литературы по системному программированию: [тык](https://vk.com/doc2892586_605986773) и немного по С/С++: [тык](http://web.archive.org/web/20230726231236/https://cpp.com.ru/).

## Какую среду программирования использовать?

В общем, любую. Но ту, с которой вы чувствуете себя уверенно.

**Для начинающих и сомневающихся** лучше всего подойдет эта сборка среды программирования Code::Blocks **(под Windows):**

[Скачать сре­ду про­грам­ми­ро­ва­ния **Code::Blocks** (вер­сия с пра­виль­ны­ми на­строй­ка­ми, она же сборка кодблокса от Дедá)](http://storage.ded32.net.ru/Lib/TX/codeblocks-mingw-setup-rar.exe):  
<http://storage.ded32.net.ru/Lib/TX/codeblocks-mingw-setup-rar.exe>

Что делать после установки? **Ctrl+Shift+N** —> создается новый файл —> пишем простейшую прогу —> далее **F9** или **F5** для компиляции (перевода в машинный код) и запуска проги.

**Для Linux и MacOS** — среда программирования **VSCode.** Установка **рекомендуемых настроек:**

1. Скачиваем [файл](https://drive.google.com/file/d/1sEZRcojqfkigDtoSZyQIKLKNDS_V_phR) с настройками.
2. Заходим в VSCode.
3. Нажимаем комбинацию **Ctrl+Shift+P (**Windows и Linux) или **Shift+Command+P** (MacOS). (Эта комбинация открывает command palette) и пишем:

**Open Settings (JSON)**

или

**Open User Settings (JSON)**

**\*ВНИМАНИЕ\*: НЕ ~~Open Default Settings (JSON)~~**

Откроется файл, имя которого будет иметь такой вид:

**Linux/MacOS**: /home/user/.config/Code/User/settings.json

**Windows:** C:\Users\<имя пользователя>\AppData\Roaming\Code\User\settings.json

**\*ВНИМАНИЕ\*: ПРОВЕРЬТЕ, ЧТО ИМЯ ФАЙЛА ИМЕННО settings.json.**

1. Заменяем содержимое открытого файла (settings.json) на содержимое скачанного в пункте 1 файла и сохраняем.
2. VSCode может предложить перезапуститься автоматически - перезапускаем его. Если не предлагает, то убедитесь, что вы **сохранили измененный файл** **(Ctrl+S),** и **перезапустите** VSCode вручную.
3. Теперь настройки соответствуют стандарту DED и любой ментор с удовольствием узнает, какие аргументы нужно писать у функции printf() :)

Заметьте, что чем навороченнее среда, тем продвинутее надо быть, чтобы не запутаться и все настроить нормально. Вы хотите быть крутыми, поэтому, конечно, сразу захотите все самое сложное. Но этого делать не стоит: вы потеряете слишком много времени, которого уже и так осталось мало до летней школы. Лучше углубиться в чтение книжек (см. пункт 1) и написание простейших программ в простой среде программирования. Поэтому, не рвитесь использовать Visual Studio (это не VS Code), CLion или любые другие среды программирования, предназначенные для мега-больших проектов, когда есть более простые и понятные. Вы потеряете очень много времени на ее настройку и понимание, вместо того, чтобы активно кодить в это время. После того, как научитесь программировать на хорошем уровне, сможете выбрать среду программирования по душе.

## Компиляция из командной строки (в основном для тех, кто в Linux или MacOS)

Компиляция – это перевод файла в машинный код, без этого ее нельзя запустить. В это время программа также проверяется на типичные ошибки.

Если у вас VSCode, откройте в ней консоль (терминал).

Либо открываем консоль в операционной системе (в Windows рекомендуется установить Far Manager, в Linux – Midnight Commander (mc) или far Manager for Linux (far2l)). В MacOS консоль называется Терминал.

Если вы в Windows и у вас **не** сборка Деда (см. п. 2 выше в тексте), вам придется самим разбираться с установкой и запуском компилятора.

Переходим в рабочую папку (где находятся тексты ваших программ) и выполняем:

**g++ FILENAME.cpp** *\*флаги\* (cм. пункт 10: флаги)*

FILENAME.cpp - это имя файла с текстом проги. Выходной файл будет называться a.exe (a.out для Linux).

## Русский язык и Windows

К сожалению, русский язык в консоли Windows это проблема, так как есть разные кодировки символов. Погуглите, что это (да, оно входит в школьную программу по инфе… ((( – минутка душнильства). ~~Так что если дружите с английским, юзайте его или потратьте еще больше времени переходя на Linux.~~ Итак, что же делать, чтобы вы могли лицезреть нужный вам вывод текста, а не вот такое вот:

| Кодировки: \*существуют\* Ваша прога: |
| --- |

**Если у вас CodeBlocks от Деда (см. выше)**, то надо добавить в начало файла ПЕРВОЙ[[1]](#footnote-0) строкой:

**#include <TXLib.h>** // И все, и все, больше ниче не надо, траляляляля!!!**[[2]](#footnote-1)**

**Если у вас другая среда программирования или вы не устанавливали CodeBlocks от Деда, то** решение точно такое же, но надо установить TXLib отсюда: <http://sourceforge.net/project/platformdownload.php?group_id=213688>.

| *P.S. Существует конечно же более грамотные методы решения этой задачи, но о них вы узнаете позже. Больно умный, Мишаня! ) Чем тебе тхлиб не угодил???)) Целая библиотека одним гигантским файлом… ага ага, а если файлов будет два, то они один из них потеряют, проверено Это понятно. Но ради пары строк, фикса кодировки добавляем графическую либу… а настройки консольного окна программно ты тоже будешь рассказывать?.. чтобы шрифт там был уникодовский с русскими буквами Дам ссыль на каникулы)))) Вот когда разберутся, как в тхлибе это устроено (там функция txSetLocale) то пусть сами и пишут, собственно, так это и задумано И ЕЩЕ ТАМ МОЖНО РИСОВАТЬ КОТИКОВ!!! На котиков я ничего не могу противопоставить. Мяу! [[3]](#footnote-2) А ЭТО СВЯЩЕННО!!!!!!! АААААА!!!!! НЕ ТРОЖЬ КОТИКОВ!!!!! Я УПРЛС давай оставим здесь этот диалог, не будем стирать, а?))) Неплохо. Но надо как то выделить. Курсивом ща бахну* |
| --- |

Рис. 1. Пример цивилизованного диалога между [тимлидом](http://vk.com/ded32_ru) и разработчиком. All rights reserved.

А вот скрин проги с работающим русским языком:



*Как видите, все работает.*

## Немного про С и С++

Мы пишем на C, но используем компилятор g++, так как он лучше по ряду причин.

Если у вас есть опыт работы на С++, то, вкратце: не используйте ничего из STL (все что требует std::, и не пишите using namespace std[[4]](#footnote-3)), методов классов, ссылок (указатели можно). Вместо cin и cout используйте функции scanf и printf.

Также: Предупреждения во время компиляции (warnings) – на самом деле это ошибки. То есть любое предупреждение компилятора должно трактоваться как ошибка. Так что если вы не можете пофиксить warning, то спросите, как это сделать правильно, а не забивайте на него.

## Linux?

Linux - это название семейства операционных систем.

Если уже работали, то окей. Но если никогда не сталкивались, не стоит разбираться сейчас, это займет очень много времени и вы пролетите. Бывает, настройка Linux занимает недели, тем временем другие вместо этого прогают, а вы от них невосполнимо отстаете.

Если вы на Windows, то пишите на Windows. В течение первого семестра не будет ничего, специфичного для Linux. Потом можно поставить WSL, это подсистема, работающая в Windows, но эмулирующая сервисы Linux. Также можно поставить и настроить виртуальную машину, это уже практически полноценный Linux, только в настройке она будет удобнее. После первого семестра на каникулах уже поставите Linux окончательно.

## Первое задание

* **Написать программу, решающую квадратное уравнение в действительных числах.**

Несмотря на то, что задание выглядит очень простым, к нему будет много требований с точки зрения профессионального программирования. О них вам расскажут менторы, когда вы им покажете первый работающий вариант. Приготовьтесь к тому, что переделывать (рефакторить) и/или что-то добавлять вы будете несколько раз. Наша цель – написать не просто задачу, а самое лучшее квадратное уравнение в вашей жизни (включая бесконечное количество последующих реинкарнаций).

До летней школы было бы неплохо добраться до простейшего варианта этого задания, будет затравочный вариант решения (прототип). На первом занятии мы разберем, как его надо делать с профессиональной точки зрения.

## На летней школе:

## Что делать, когда написал первый вариант?

Заполняйте форму заявку на проверку, ваш ментор найдет вас. Заполняйте ее, когда у вас уже есть код посмотреть. Для удобства лучше выкладывать на GitHub, инструкцию см. ниже. Если вы не знаете как, то спросите у ментора во время показа первого варианта. Ссылки:

[Форма на запрос менторинга](https://docs.google.com/forms/d/e/1FAIpQLSdaDjRR_D6C8LjbraxtFsd8pSHOj__EraXRbOrukKKTNtOCOQ/viewform) и [ответы на форму](https://docs.google.com/spreadsheets/d/1PCO-fFs2_VPsLlOn9zE7_rXO02mI3hpesYJYMa4skPQ/edit?resourcekey=&gid=1810579476#gid=1810579476), чтобы посмотреть на назначенного вам ментора

## Что делать дальше?

Продолжайте работу с ментором. Мы, менторы, здесь, чтобы помочь вам. Не бойтесь задавать вопросы, даже те, которые кажутся глупыми. Глупых вопросов не бывает.

Работа над заданиями – итеративный процесс. Следуйте наставлениям вашего ментора и просите посмотреть код снова.

Следующие задания интереснее и объемнее, условие вам выдаст ваш ментор, как только вы сделаете предыдущее задание.

## Что такое полностью сданная работа?

* Работа считается сданной, если ментор вам написал или сказал явно *работа полностью принята,* или синонимы этого. До этого момента можно считать, что в работе есть недочеты, которые необходимо устранить.
* После этого работу проверяет другой ментор, вы устраняете его замечания.
* После этого, очно или дистантно, вы, вместе с обоими менторами, сдаете эту работу [Деду.](http://vk.com/ded32_ru)

# Котики и чатик

* Отправленный котик на фото, особенно свежепокормленный и свежепоглаженный, особенно на видео, на порядок увеличивает вероятность успеха любого действия! Если у вас есть какие-либо вопросы по проге – не стесняйтесь, задавайте. Мы, менторы (админы этой беседы), будем помогать вам разбираться. Этот чатик создан специально для этих целей. Пишите вопросы именно сюда, а не в личные сообщения, чтобы:

1) получить более комплексное мнение от разных менторов,

2) этот вопрос отпал у всех остальных, кто стесняется его спросить.

  
Рис. 2. Не без кота.

# Флаги компиляции

Рекомендованные флаги для компиляции: [тык](https://drive.google.com/file/d/1QKsGD89Xz_ydFJ0MOrq2wEsvY9FXoFa8/view?usp=sharing).

**Windows:**

-Wshadow -Winit-self -Wredundant-decls -Wcast-align -Wundef -Wfloat-equal -Winline -Wunreachable-code -Wmissing-declarations -Wmissing-include-dirs -Wswitch-enum -Wswitch-default -Weffc++ -Wmain -Wextra -Wall -g -pipe -fexceptions -Wcast-qual -Wconversion -Wctor-dtor-privacy -Wempty-body -Wformat-security -Wformat=2 -Wignored-qualifiers -Wlogical-op -Wno-missing-field-initializers -Wnon-virtual-dtor -Woverloaded-virtual -Wpointer-arith -Wsign-promo -Wstack-usage=8192 -Wstrict-aliasing -Wstrict-null-sentinel -Wtype-limits -Wwrite-strings -Werror=vla -D\_DEBUG -D\_EJUDGE\_CLIENT\_SIDE

**Linux:**

-D \_DEBUG -ggdb3 -std=c++17 -O0 -Wall -Wextra -Weffc++ -Waggressive-loop-optimizations -Wc++14-compat -Wmissing-declarations -Wcast-align -Wcast-qual -Wchar-subscripts -Wconditionally-supported -Wconversion -Wctor-dtor-privacy -Wempty-body -Wfloat-equal -Wformat-nonliteral -Wformat-security -Wformat-signedness -Wformat=2 -Winline -Wlogical-op -Wnon-virtual-dtor -Wopenmp-simd -Woverloaded-virtual -Wpacked -Wpointer-arith -Winit-self -Wredundant-decls -Wshadow -Wsign-conversion -Wsign-promo -Wstrict-null-sentinel -Wstrict-overflow=2 -Wsuggest-attribute=noreturn -Wsuggest-final-methods -Wsuggest-final-types -Wsuggest-override -Wswitch-default -Wswitch-enum -Wsync-nand -Wundef -Wunreachable-code -Wunused -Wuseless-cast -Wvariadic-macros -Wno-literal-suffix -Wno-missing-field-initializers -Wno-narrowing -Wno-old-style-cast -Wno-varargs -Wstack-protector -fcheck-new -fsized-deallocation -fstack-protector -fstrict-overflow -flto-odr-type-merging -fno-omit-frame-pointer -Wlarger-than=8192 -Wstack-usage=8192 -pie -fPIE -Werror=vla -fsanitize=address,alignment,bool,bounds,enum,float-cast-overflow,float-divide-by-zero,integer-divide-by-zero,leak,nonnull-attribute,null,object-size,return,returns-nonnull-attribute,shift,signed-integer-overflow,undefined,unreachable,vla-bound,vptr

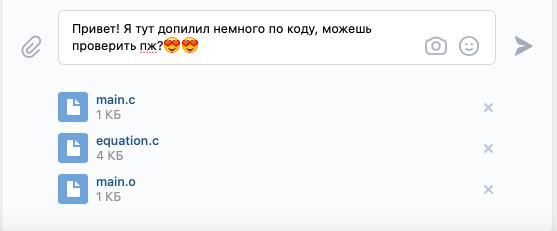
В сборке CodeBlocks от Деда они встроены по умолчанию. Более полную инфу о них см. в конце этого дока: [http://storage.ded32.net.ru/Lib/TX/TXUpdate/Doc/HTML.ru](http://storage.ded32.net.ru/Lib/TX/TXUpdate/Doc/HTML.ru/).

# Github

## Введение

Для начальных сведений достаточно знать, что гитхаб ([**GitHub**](https://github.com/), не путать с просто git, подробнее будет ниже, не бойтесь) - это облачное хранилище, на которое разработчики загружают свои проекты, работают вместе над ними удаленно, просматривают код друг дружки, перекидываются в комментариях картинками с котиками, улюлюкают над багами и делают прочую прогерскую работу. В рамках всего курса вы будете постоянно пользоваться им для того, чтобы показывать код своим менторам.

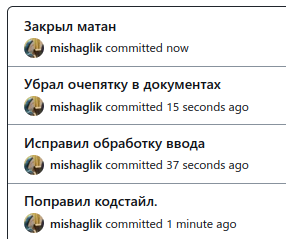
Многие из вас с огнем в глазах и сердце будут делать что-то такое со своими менторами:



Так делать не надо, так как у менторов лапки и шалящие нервишки :) Итак, намного лучше, когда вы делаете что-то вроде этого:



А вот что в истории коммитов видно:

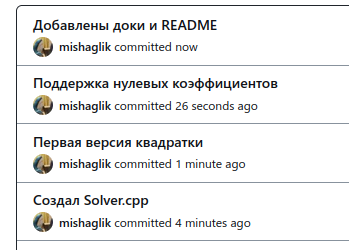


## Что за кошмар? Какие ещё коммиты, гиты, гитхабы?

При редактировании файлов (работе в фотошопе, написании кода и пр.) часто возникает необходимость в сохранении версий файла. К примеру, чтобы иметь возможность откатить изменения назад, когда Ctrl+Z уже недоступно (например, закрыта вкладка с файлом). Решение такой задачи “в лоб”, как правило, представляет собой создание кучи файлов или папок с номерами и названиями вида “Project.c”, “Project(1).c”, “Project\_updated.c”, “Project\_final.c”, “Project\_final\_final.c” и т.д. Очевидно, что это как громоздко, так и неудобно, вызывает путаницу. По этой причине были придуманы **системы контроля версий**, наиболее популярной среди которых является git.



В рамках системы контроля версий в пределах одного **репозитория** (по большому счёту, проекта; к примеру, одной задачи у Деда) поддерживается история и последовательность версий. **Коммиты** в таком случае выполняют роль пунктов (шагов, “снимков”, состояний) в этой истории. Каждому коммиту присущи: номер (на самом деле это хеш), сообщение (краткое описание изменений, к примеру, “добавлена документация функций”, “поддерживаются нулевые коэффициенты”, “исправлены ошибки сравнения чисел double” и пр.) и список изменений. Примером последовательности коммитов может быть такая история:



Github – это сайт, он хранит репозитории, что позволяет разработчику не переживать по поводу потери данных на компе от проблем с операционной системой и всякого иного треша.

Ладно, мы не на лекции по матану, так что время перейти к практике и, наконец, разобраться как работать с git’ом и GitHub’ом.

## Создание профиля

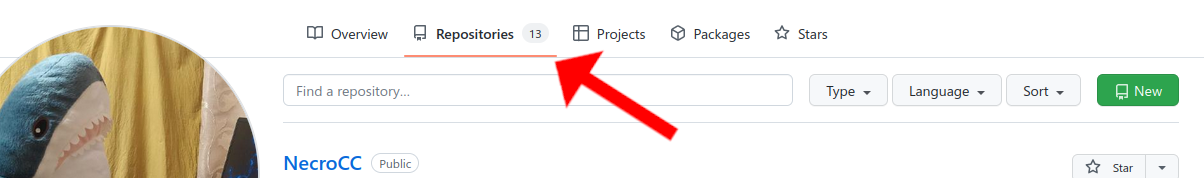
Для создания своего репозитория на гитхабе:

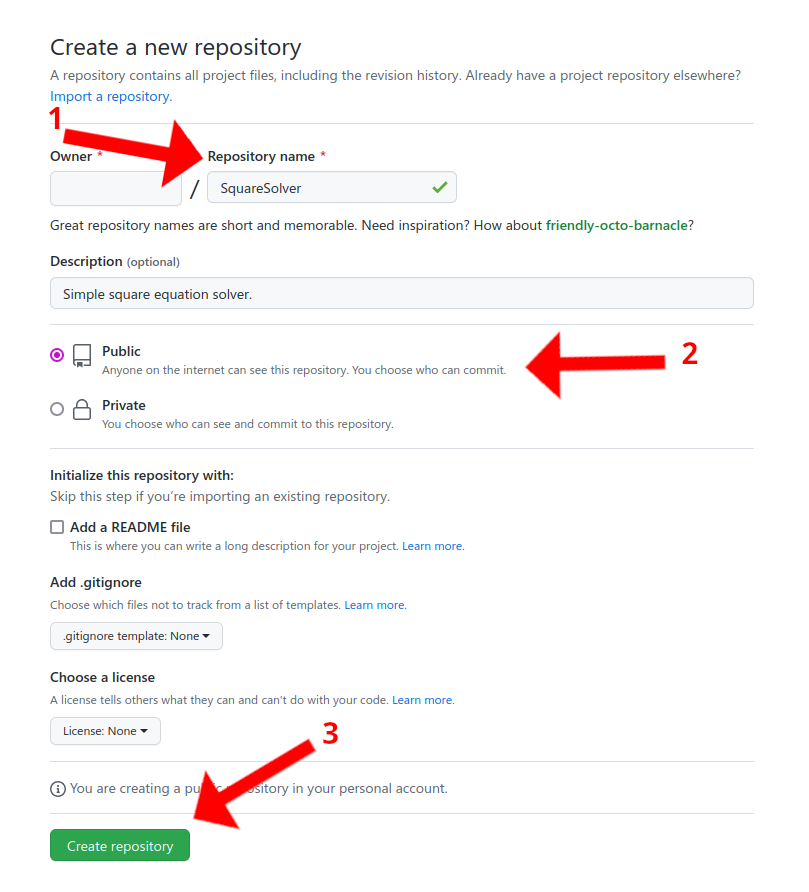
1. Переходим на [GitHub](http://www.github.com).
2. Регистрируемся.
3. Радуемся первым успешным шагам, так как теперь вы успешный юзер гитхаба!

## Создание первого репозитория

Репозиторий (репо) – это грубо говоря папка с вашим крутым проектом, просто эта директория находится в облачном хранилище. Тут все просто, структура не отличается от обычной папке на вашем компьютере.

Для создания репозитория:

1. Заходим в своем профиле во вкладку repositories 
2. Жмякаем на эту большую зеленую кнопку **New**
3. Вводим имя проекта. Например SquareSolver (решатель квадратных уравнений)
4. Не обращаем внимания на кнопочки по типу создания README.md и .gitignore
5. Делаем его публичным, чтобы можно было хвастаться вашим перфомансом друзьям и менторам
6. Нажимаем на кнопку создания:



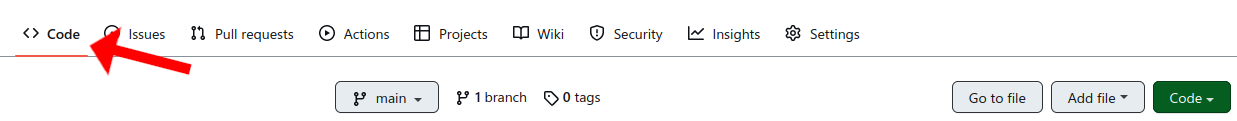
1. Теперь, когда вас просят показать код с вашего репозитория, достаточно просто зайти в него и…
2. …скопировать url ссылку сверху из окошка адреса в браузере.
3. Гитхаб вас поздравит с первым репозиторием! Это круто.

## Заливаем ~~хороший код~~ файлики для ревью

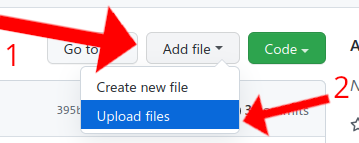
Пустой репозиторий не интересный, надо бы залить туда файлики с кодом с вашего компа. Мы сейчас пойдем самым мега простым путем, не супер правильным. Потом расскажем, как это делать лучше.

Для этого:

1. Перейдите в ваш репозиторий в раздел code

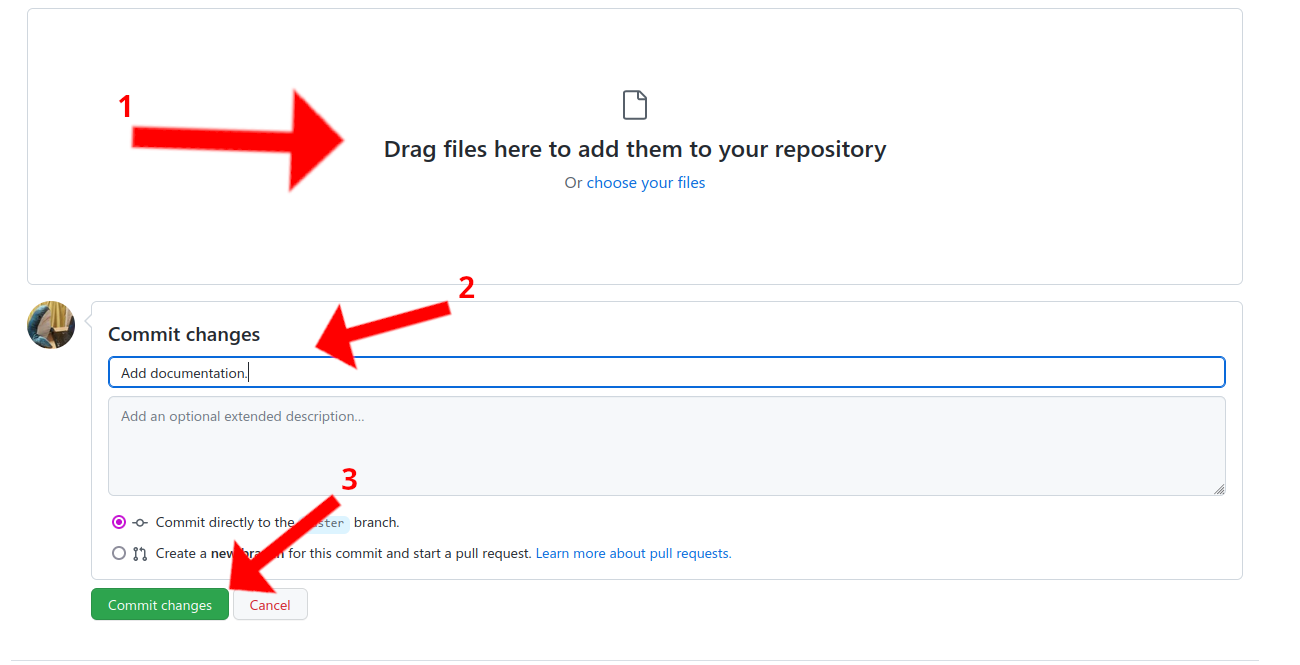


1. Нажмите на Add file , а после чего Upload file



Ну дальше выбираем файлы с компьютера и загружаем их.

1. Придумываем хорошее описание для коммита. Например “Add documentation”. Или “Fixed input issues”. Или “Я пофиксил баги ввода!” Или “Я блин забыл вообще написать ввод чисел, как оно до этого работало???”



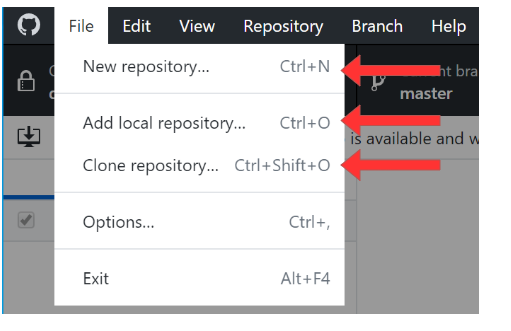
## GitHub Desktop

Если вам неудобно постоянно заливать файлы через браузер (а это обязательно должно быть вам неудобно!), советуем поставить вам GitHub Desktop. Это приложение с графическим интерфейсом, которое упрощает работу с гитхабом. Скачать его можно по ссылкам:

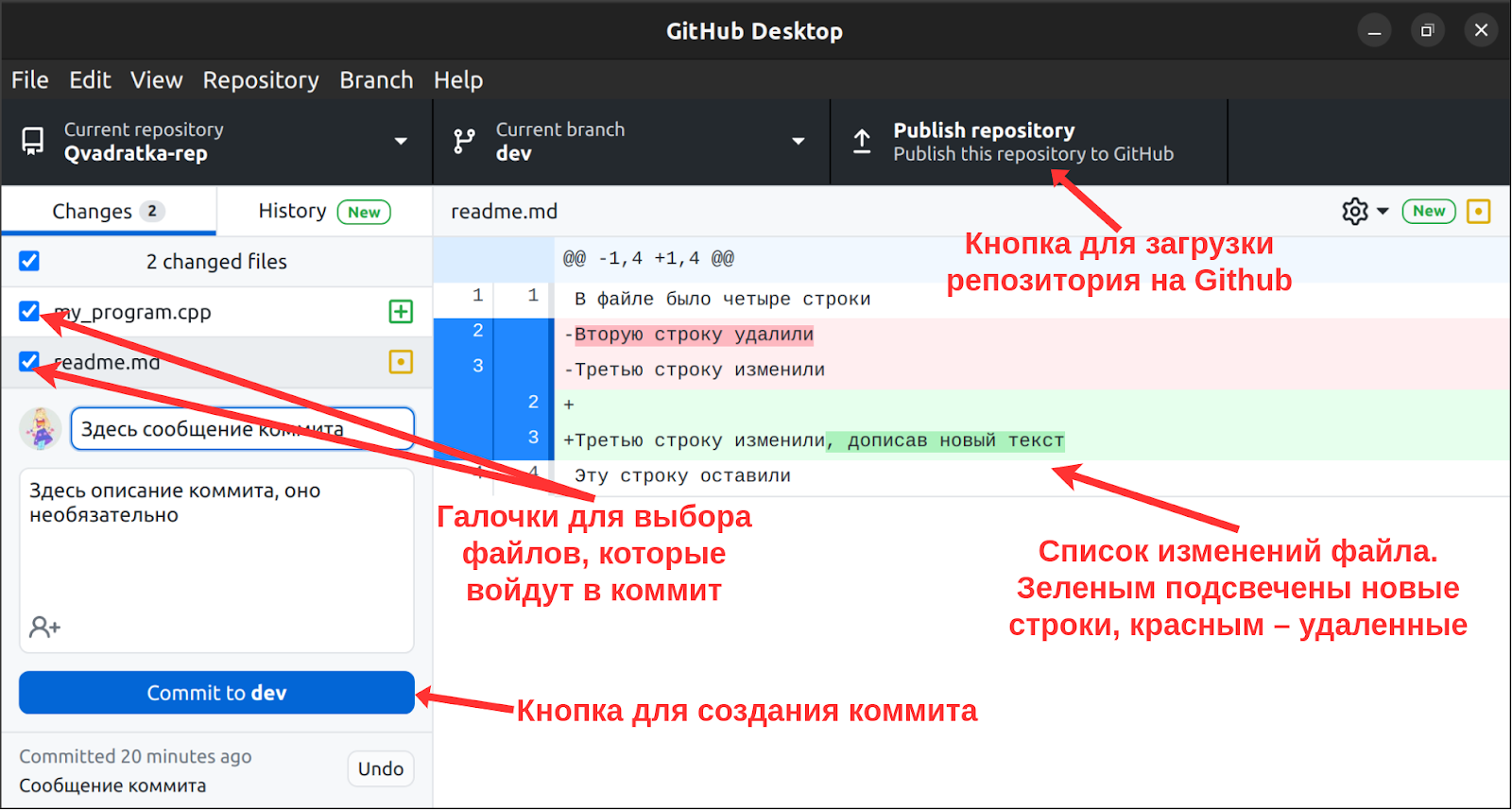
* Для Windows/Mac: [https://desktop.github.com](https://desktop.github.com/)
* Для Linux: <https://github.com/shiftkey/desktop/releases>

После установки вам нужно будет авторизоваться в приложении через ваш аккаунт на Github.

Чтобы добавить репозиторий, откройте раздел “File” из верхнего меню. Там вы можете создать новый репозиторий, склонировать (скачать к себе на компьютер) репозиторий из github или открыть уже существующий локальный репозиторий.

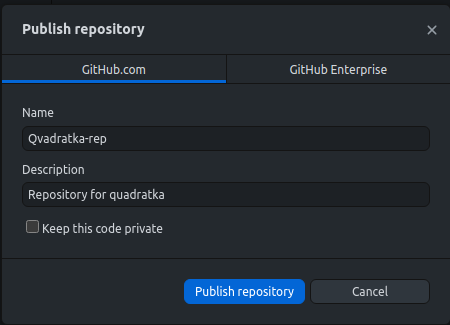


После создания репозитория, вы можете приступить к написанию кода. Как только вы решили сохранить очередную версию своей задачи (сделать коммит), откройте GitHub Desktop, откройте ранее созданный репозиторий. Измененные файлы появятся в панели слева. Вы можете поставить галочки на тех файлах, которые войдут в коммит (да, не обязательно фиксировать изменения во всех файлах разом, можно добавить в коммит только некоторые файлы, но поначалу это вам вряд ли пригодится). В правой части окна будет отображаться, какие изменения вы внесли в файл с момента последнего коммита. В нижней части левого меню будет окошко, чтобы ввести краткое описание изменений (summary) и, по желанию, подробное описание изменений (description).



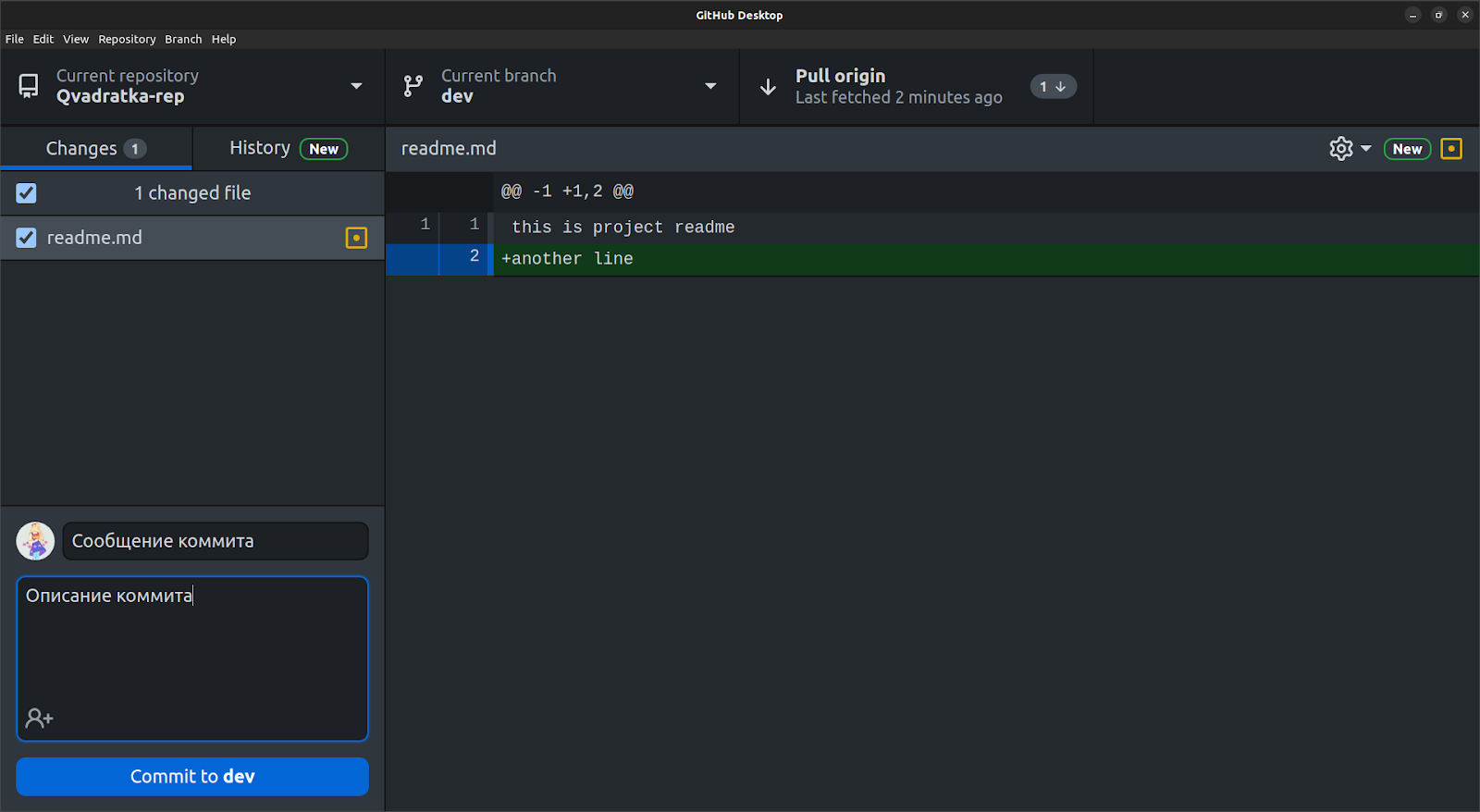
После того, как вы поставили галочки на файлах, которые войдут в коммит и ввели сообщение коммита, можно нажимать кнопку “Commit to …”. Вместо троеточия будет название ветки, у меня это dev, у вас, скорее всего, будет master или main. Поздравляю, вы сделали коммит! Осталось только залить этот коммит на Github (эта операция называется push).

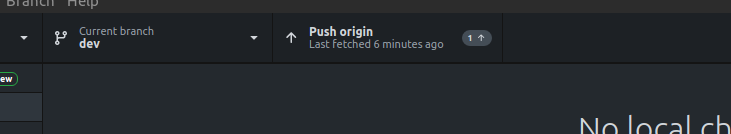
Для того, чтобы загрузить локальный репозиторий на гитхаб, надо нажать кнопку “Publish repository” в верхнем меню. В открывшемся окошке введите имя репозитория (обязательно), и описание репозитория (по желанию). Обязательно снимите галочку “Keep this code private”, иначе ваш репозиторий будет доступен только вам, и менторы не смогут посмотреть ваш код. Настоящие тру пацаны никогда не делают приватные репо.



Как только вы ввели имя репозитория и сняли галочку “Keep this code private”, можно нажимать “Publish repository”. Поздравляем, ваш репозиторий теперь на гитхабе!

Чтобы залить последующие изменения на гитхаб, нужно:

1. Внести изменения в файлы
2. Ввести сообщение коммита и сделать коммит
3. Нажать на кнопку “Push origin”, после создания коммита она появится там, где раньше была кнопка “Publish repository”, – в верхнем меню.

После этого измененные файлы появятся в репозитории на гитхаб и менторы смогут увидеть их. Теперь, если вы даже выбросите свой комп (зачем он вам?), ваши файлы навечно останутся на Гитхабе. Voila.

## ~~Git diff.~~ Нет, Великий Git Diff!

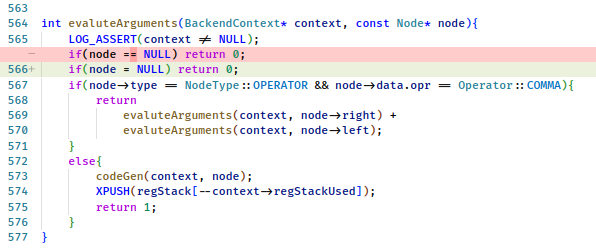
Теперь давайте поговорим о правой части экрана GitHub Desktop, в которой отображаются изменения, которые вы внесли. Это очень полезная область, с ее помощью можно отловить простые ошибки и опечатки. Давайте разберем на примере.

Итак, вы работаете над большим проектом. Вы занятой человек и у вас одновременно на компе запущена и среда программирования, браузер, Telegram, ~~и онеме~~ и много чего. И вот переключаясь между окнами, туда сюда набирая разный текст, вы нечаянно стёрли один символ в коде... Вроде бы. И не заметили. Хм. Ну может тогда компилятор скажет, что я ошибся? Нет, вам не повезло. Компилятор молчит.

А на следующий день программа работает некорректно. Хм странно. Только вчера этот же код работал… Как найти, что изменилось. Можно конечно пристально посмотреть…



М-да. Слишком много… и это довольно сложный код, далеко не первая задача. И как же найти что со вчера поменялось? Но вы как разумный человек конечно же сделали коммит на моменте когда у вас код работал. (Если нет, то **F.** Нужно больше коммитов! Делайте коммит на каждой стабильной позиции!!!) На помощь придет git diff. Он показывает отличия вашей текущей версии кода от последнего коммита. (Или между двумя коммитами).



Красное - то, что было.

Зеленое - то, что стало.

Опа! Да вот же она, опечатка на 566-й строке.

Из примера выше можно сделать вывод – читайте диффы (от англ diff. – различие, так называется область, в которой отображаются изменения), они помогут вам отловить ошибки и опечатки, которые может пропустить даже компилятор.

Здесь нужна картинка “собачка в огне Fine”. Вот она:

|  |  |
| --- | --- |

Справа - то, что народная мудрость говорит о git во время пожара. Соблюдайте правила пожарно-охранной сигнализации! Вы еще услышите ее в общаге в 6 утра. Конечно же, ее включают, чтобы вы не забыли сделать коммит!

Еще более грамотно – использовать Git из консоли или с помощью более продвинутых программ, например, SmartGit или других. Но об этом в кратком гайде мы не будем, а расскажем вам позже.

\*\*\*

This Гайд ®©™ подготовили ~~master~~ [Дед](http://vk.com/ded32_ru) и его ~~slaves~~ менторы**[[5]](#footnote-4)** (больше менторы, чем [Дед](http://vk.com/ded32_ru)[[6]](#footnote-5) (они же slaves (но не в том смысле (а в каком том? (а вот в таком (в правильном (норм чуваки а чо (а ниче (ну и че (пойдем выйдем (не пойду, я **K&R[[7]](#footnote-6)** читаю.)))))))))))) VIVA LISP (А это к Винокурову (это норм препод такой (Лисп любит (а почему ЛИСП? а скобочки)))))))))

\*\*\*

ОПУБЛИКОВАНО ПОД ВСЕМИ КОПИРАЙТАМИ И КОПИЛЕФТАМИ[[8]](#footnote-7)

ИЗДАТЕЛЬСТВО ФИЗТЕХ КРИНЖ BOTAY ПАБЛИШЕРС poltorashka edition

~~МОСКВА~~ (ДОЛГОПА БЛИН!!!) 2024

1. Главное что до подключения стандартных библиотек [↑](#footnote-ref-0)
2. [Дед](http://vk.com/ded32_ru) упрлс. Это бывает [↑](#footnote-ref-1)
3. Глас свыше, среагировавший на упоминание о котиках. [↑](#footnote-ref-2)
4. За это мы прям убиваем. Натурально, да. [↑](#footnote-ref-3)
5. Попросили остаться анонимными, дабы избежать преследования со стороны перовокуров. [↑](#footnote-ref-4)
6. Так-то он отвечает за большую часть кринжа здесь8. [↑](#footnote-ref-5)
7. Конечно, имеется в виду K&R8. [↑](#footnote-ref-6)
8. Нет дискриминации для правшей и для левшей. А всех несогласных мы погоним взашей8.  
   8 Разумеется 5, 8. [↑](#footnote-ref-7)