Министерство образования Российской Федерации

Тверской государственный технический университет

Кафедра Программного обеспечения

Программа для работы с личными делами студентов

Программный модуль «Личные дела студен­тов»

Выполнил: Бакеев А.В

Проверил: Мальков А.А

Тверь, 2024

Оглавление

[Введение 3](#_Toc178678338)

[Постановка задачи 3](#_Toc178678339)

[Построенная диаграмма 4](#_Toc178678340)

[Код для Mermaid 4](#_Toc178678341)

# Введение

В данной лабораторной работе рассматривается разработка информационной системы "Личные дела студентов", которая нацелена на автоматизацию учета и управления данными студентов учебного заведения. Система предназначена для хранения информации о личных делах студентов, их успеваемости, городе рождения, статусе работы. Так же система предоставим возможность сотрудникам получать все необходимые данные для составления справок и документов. Создание такой информационной системы позволит улучшить организацию работы с данными студентов, повысить эффективность учебного процесса и облегчить доступ к информации как для преподавателей, так и для студентов.

# Постановка задачи

**Вариант 2.** Разработать программный модуль «Личные дела студентов». Программный модуль предназначен для получения сведений о студентах сотрудниками деканата, профкома и отдела кадров. Сведения должны храниться в течение всего срока обучения студентов и использоваться при составлении справок и отчетов.

# Модель waterfall

Техническое задание:

1. **Введение**

Настоящее техническое задание распространяется на разработку программы личных дел студентов, предназначенной для использования деканатом, отделом кадров и профкомом при работе, в моменты, когда необходимо получить некоторые личные данные того или иного студента (для заполнения отчетов, справок и т.д.)

1. **Основание для разработки**
   1. Программа разрабатывается на основе учебного плана кафедры «Программного обеспечения»
   2. Программный модуль «Личные дела студентов»
   3. Исполнитель: компания Бакеев & co.
   4. Соисполнители: нет.

**3. Назначение**

Программа предназначена для работы сотрудников отдела кадров, деканата и профком, для оперативного получения доступа к личным данным студентов.

**4. Требования к программе или программному изделию**

Функциональные требования:

1. Возможность получить все имеющиеся в базе данные о конкретном студенте (Фамилия, имя, отчество, город рождения, средний балл за предыдущий семестр, рабочий статус (работает, не работает, самозанятый, имеет Юр. Лицо))
2. Получить выборочно данные (которые, по мнению заказчика, чаще всего нужны) чаще всего необходимые для заполнения справок и документов
3. Возможность заполнять базу данных и редактировать существующие данные сотрудникам отдела кадров, остальные пользователи должны иметь возможность только на чтение данных

Нефункциональные требования:

1. Безопасность системы. Человек, не являющийся нашим сотрудником не сможет вводить данные студентов, дабы не фальсифицировать их.
2. Понятный интерфейс для пользователя любого уровня (отсутствие кликабельных картинок, очень маленького шрифта или надписей/кнопок, сливающихся с фоном окна приложения)
3. Возможность масштабирования (дополнительные справки, дополнительные данные в системе)

Требования к составу и параметрам технических средств:

* Процессор: Intel Core 2 Duo
* Оперативная память: 2 GB ОЗУ
* Видеокарта: DirectX 11 compatible video card (integrated or dedicated with min 512MB memory)
* DirectX: версии 11
* Место на диске: 1 GB

**5. Требования к программной документации**

1. Программа должна иметь исчерпывающие комментарии во всех существующих и будущих блоках кода.
2. Пояснительная записка для разработчиков с описанием технических решений.
3. Документация для пользователя

**6. Сроки разработки продукта**

Разработка приложения должна начаться в период с 1.10.2024 по 1.11.2024 и продлится не больше 3 недель

**#убрать Плюсы и минусы waterfall модели:**

+ Четкость и структура: Этапы проекта четко определены (требования, проектирование, разработка, тестирование, внедрение). Это упрощает управление проектом.

+ Легкость планирования: В связи четкой структуры проще планировать и прогнозировать сроки выполнения.

+ Документирование: Каждый этап располагает к написанию документации, что обеспечивает хорошую запись требований и решений.

+ Управляемость: Изменения в проекте контролируются на каждом этапе, что упрощает отслеживание процесса

- Негибкость: Трудно вносить изменения в требования после начала разработки, это может быть проблематично при изменении условий после начала работы над проектом.

- Риски: Возможность выявления серьезных проблем или несоответствий только на поздних этапах разработки и тестирования, как следствие это может увеличить затраты на исправление, как временные, так и финансовые.

- Задержки: Если один этап задерживается, это может задержать весь проект, так как этапы следуют друг за другом «паровозиком».

- Проблемы с требованиями: Если требования изменяются по ходу проекта, это может привести к необходимости переделки уже завершенных этапов, что увеличивает затраты и время. А в худшем варианте данного сценария, элементы проекта, шедшие за переделанными становятся не работоспособными в принципе.

# V-образная модель SDLC

Техническое задание

* 1. **Введение**

Настоящее техническое задание распространяется на разработку программы личных дел студентов, предназначенной для использования деканатом, отделом кадров и профкомом при работе, в моменты, когда необходимо получить некоторые личные данные того или иного студента (для заполнения отчетов, справок и т.д.)

**2. Основание для разработки**

* 1. Программа разрабатывается на основе учебного плана кафедры «Программного обеспечения»
  2. Программный модуль «Личные дела студентов»
  3. Исполнитель: компания Бакеев & co.
  4. Соисполнители: нет.

**3. Назначение**

Программа предназначена для работы сотрудников отдела кадров, деканата и профком, для оперативного получения доступа к личным данным студентов.

**4. Требования к программе или программному изделию**

Функциональные требования

1. Возможность получить все имеющиеся в базе данные о конкретном студенте (Фамилия, имя, отчество, город рождения, средний балл за предыдущий семестр, рабочий статус (работает, не работает, самозанятый, имеет Юр. Лицо))
2. Получить выборочно данные (которые, по мнению заказчика, чаще всего нужны) чаще всего необходимые для заполнения справок и документов
3. Возможность заполнять базу данных и редактировать существующие данные сотрудникам отдела кадров, остальные пользователи должны иметь возможность только на чтение данных

Нефункциональные требования:

1. Безопасность системы. Человек, не являющийся нашим сотрудником не сможет вводить данные студентов, дабы не фальсифицировать их.
2. Понятный интерфейс для пользователя любого уровня (отсутствие кликабельных картинок, очень маленького шрифта или надписей/кнопок, сливающихся с фоном окна приложения)
3. Возможность масштабирования (дополнительные справки, дополнительные данные в системе)

Требование к составу и параметрам технических средств:

* Процессор: Intel Core 2 Duo
* Оперативная память: 2 GB ОЗУ
* Видеокарта: DirectX 11 compatible video card (integrated or dedicated with min 512MB memory)
* DirectX: версии 11
* Место на диске: 1 GB

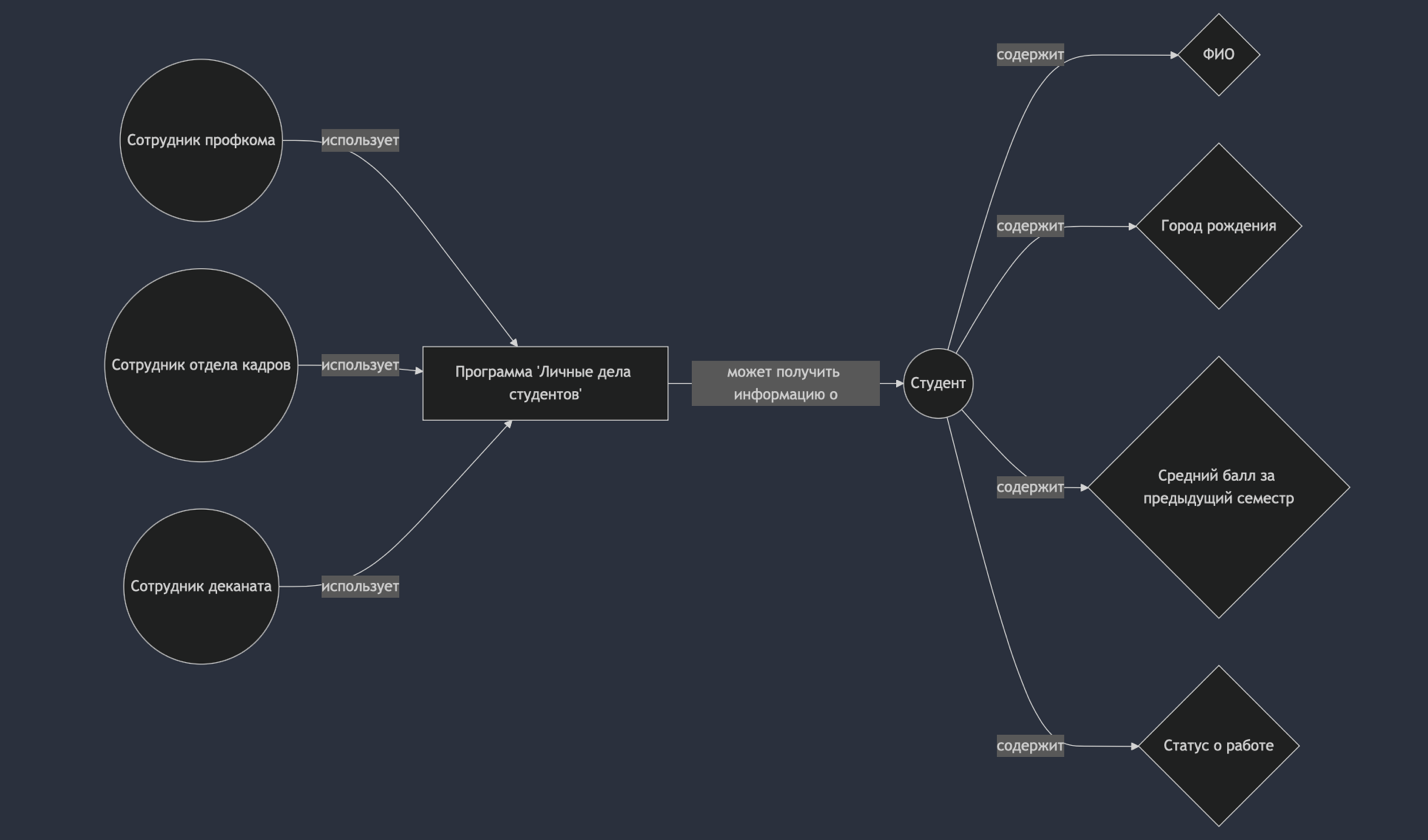
**5. Требования к программной документации:**

1. Программа должна иметь исчерпывающие комментарии во всех существующих и будущих блоках кода.
2. Пояснительная записка для разработчиков с описанием технических решений.
3. Документация для пользователя

**6. Сроки разработки продукта**

Разработка приложения должна начаться в период с 1.10.2024 по 1.11.2024 и продлится не больше 3 недель

# Построенная диаграмма



Сущности в виде «Сотрудник профкома», «сотрудник отдела кадров», «сотрудник деканата» используют программу «Личные дела студентов».

«Личные дела студентов» обращается (может получить по необходимости) информацию о сущности «Студент».

«Студент» в свою очередь содержит о себе следующие информационные записи – ФИО; Город рождения; Средний балл за предыдущий семестр; Статус о работе

# Код для Mermaid

**graph** LR

A[Программа 'Личные дела студентов']

C((Сотрудник профкома)) **-->**|использует| A

D((Сотрудник отдела кадров)) **-->**|использует| A

B((Сотрудник деканата)) **-->**|использует| A

E((Студент))

A **-->**|может получить информацию о| E

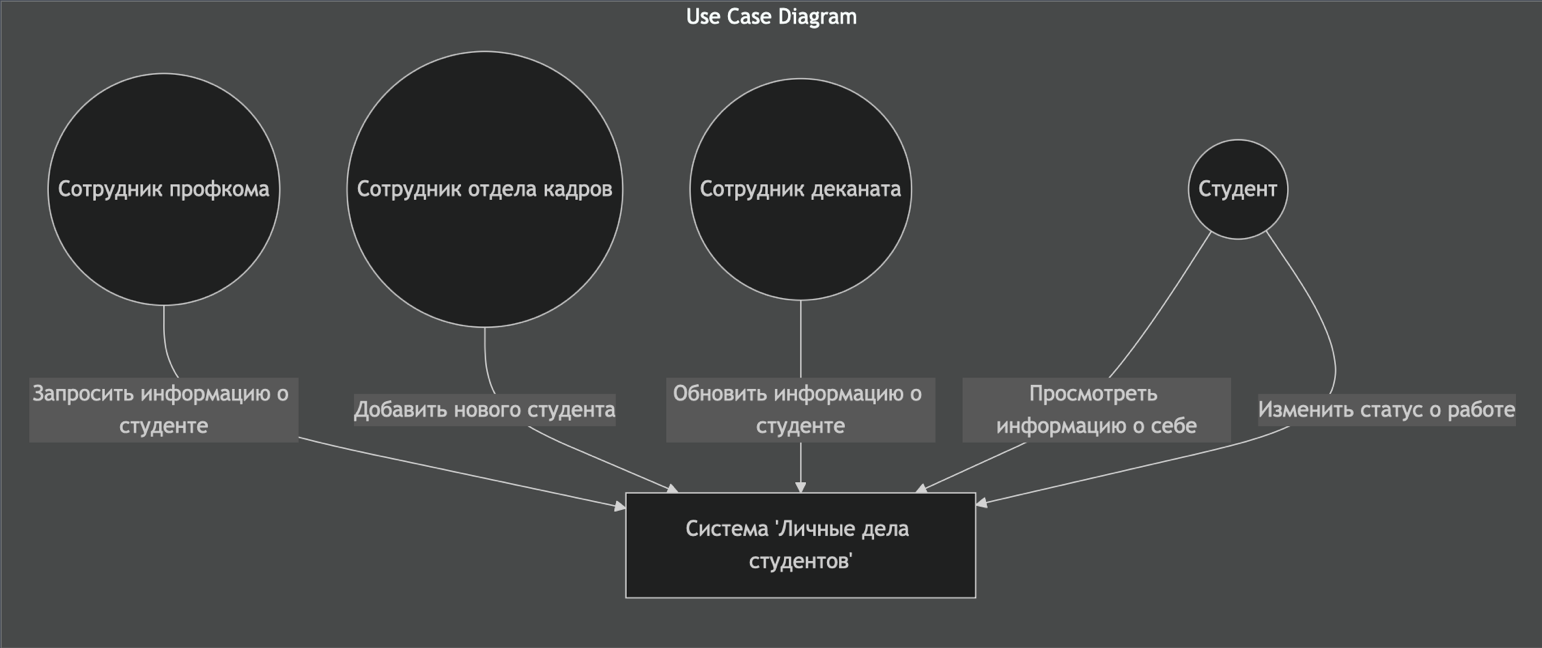
E **-->**|содержит| F{ФИО}

E **-->**|содержит| G{Город рождения}

E **-->**|содержит| H{Средний балл за предыдущий семестр}

E **-->**|содержит| I{Статус о работе}

# Use case diagram



Данный кейс демонстрирует следующие возможные варианты использования приложения:

1. Сотрудник профкома обращается к системе для запроса информации о интересующем их студенте
2. Сотрудник отдела кадров обращается к системе, чтобы добавить нового, только что зачисленного в университет студента
3. Сотрудник деканата обращается к системе для того, чтобы обновить информацию о студенте, на основании того, что, например, изменился статус «Средний балл за предыдущий семестр» у студента и деканат заполняя данные за новый семестр, попутно обновляет данные в системе
4. Сам студент может обратиться к системе для того, чтобы:
   1. Посмотреть какая информация о нем содержится и может быть запрошена/получена деканатом и иными пользователями с доступом (отдел кадров, профком).
   2. Изменить свой статус о работе. Например, студент нашел себе летом подработку и решил сохранить ее и в учебный год. А чтобы у университета были актуальные сведения – обновил данные сразу же сам и они появились в системе

Код для Use case diagram в среде Mermaid lucidchart

**graph** LR

C((Сотрудник профкома))

D((Сотрудник отдела кадров))

B((Сотрудник деканата))

E((Студент))

%% Use Case Diagram

**subgraph** Use\_Cases [Use Case Diagram]

direction TB

A2[Система 'Личные дела студентов']

C **-->**|Запросить информацию о студенте| A2

D **-->**|Добавить нового студента| A2

B **-->**|Обновить информацию о студенте| A2

E **-->**|Просмотреть информацию о себе| A2

E **-->**|Изменить статус о работе| A2

**end**

# Пункт первый

Для реализации моего варианта я буду использовать следующий технологии:

Основная среда разработки – PyCharm (из под macOS)

Язык программирования – python (3.12)

Библиотека для реализации GUI (graphical user interface) – PyQt5 или tkinter

# Пункт второй

PyCharm является самой популярной и удобной средой разработки для создания проектов на языке программирования python. Это так же и место, содержащее множество мощных инструментов для написания и тестирования кода. Новые технологии-помощники сильно упрощают работу – предлагают где какой блок кода нужно дописать, при передаче в функцию каких-либо параметров подсвечивают серым, что есть что, дабы не перепутать. А удобный интерфейс оказывается весьма дружелюбным и понятным, чего нельзя сказать о некоторых других средах (например - VS Code, который так же подходит для задачи куда менее удобен, т.к. не является полноценной средой разработки, а лишь прокаченной версией редактора кода).

Python – простой и приятный синтаксически язык, способный легко и быстро решать множество задач, не требующих сверх быстрой работы кода или очень маленького пространства, занимаемого программой на жестком диске. Наш случай не является ни первым, ни вторым. К тому же на Пайтон существует множество библиотек, которые легко и удобно устанавливаются на любую версию интерпретатора, имеют хорошую совместимость друг с другом и достаточно полную документацию.

PyQt5 – библиотека для того, чтобы можно было работать на пайтоне с фреймворком Qt – достаточно популярным способ создания кроссплатформенных графических интерфейсов для десктопных приложений.

Tkinter – так же библиотека для написания несложных gui. Простая, не требовательная к мощностям машины, на которой будет запускаться программа, понятный и простой синтаксис команд.

# Пункт третий

Декомпозиция с описанием каждого модуля:

1. Продумать модуль хранения данных студента (какой тип данных у какого поля будет, каким способом будут хранить – с базой данных или нет, а если с базой данных – то какой)
2. Продумать функционал части сотрудника профкома – какие данные сможет получать о каждом студенте и в каком формате
3. Продумать функционал части сотрудника деканата– какие данные сможет получать о каждом студенте и в каком формате
4. Продумать функционал части сотрудника отдела кадров – какие данные сможет получать о каждом студенте и в каком формате
5. Так же продумать общую часть интерфейса – изначально программа будет одинаковая для всех – что со стороны университета, что со стороны деканата. Но в стартовом меню будет несколько вопросов, утоняющих, какой вид программы должен увидеть пользователь после ответа на них. Для стороны университета будут уникальные id, без которые нельзя будет авторизоваться в качестве сотрудника. Для студента же – паспортные данные.
   1. Id всегда целое число (не имеет значение знак, лишь отсутствие дробной части).
   2. Если id будет четным, значит это аккаунт студента
   3. Если id нечестный – это аккаунт сотрудника
   4. Стартовая форма просит ввести ваш id и на его основании определяет, какой интерфейс вы увидите дальше, после проверки.
   5. Студент – свои личные данные и возможность их редактировать
   6. Сотрудник – Список студентов и возможно получить данные о каждом конкретном из них

# Пункт четвертый

Набор необходимых задач:

1. Импортировать библиотеки для работы с GUI в среду разработки PyCharm
2. Сделать mvp\* версию приложения с хранением всех данных в файлах. У файла будет строго определенная система – у каждой строки свое предназначение
   1. Фамилия Имя Отчество
   2. Год рождения
   3. Средний балл за предыдущий семестр
   4. Статус о работе
3. Связать эту версию с интерфейсом и проработать систему id. Провести тесты, цель которых выявить ограничения функционала приложения, в зависимости от статуса пользователя (студент или сотрудник)

\*mvp подразумевает наличие меню, которое, по введенному id будет определять ваш уровень доступа (студент или сотрудник университета). В зависимости от этого уровня будет открываться разное меню приложения. В меню студента можно будет посмотреть свои текущие данные, а в меню сотрудни

# Установка git

Открываем терминал и пишем brew install git

macbook@MacBook-Pro-macbook ~ % brew install git

==> **Fetching git**

==> **Downloading https://raw.githubusercontent.com/Homebrew/homebrew-core/4315a95**

######################################################################### 100.0%

==> **Downloading https://mirrors.edge.kernel.org/pub/software/scm/git/git-htmldoc**

######################################################################### 100.0%

==> **Downloading https://mirrors.edge.kernel.org/pub/software/scm/git/git-manpage**

######################################################################### 100.0%

==> **Downloading https://cpan.metacpan.org/authors/id/R/RJ/RJBS/Net-SMTP-SSL-1.04**

######################################################################### 100.0%

==> **Downloading https://mirrors.edge.kernel.org/pub/software/scm/git/git-2.47.0.**

######################################################################### 100.0%

…

==> **Running `brew cleanup git`...**

Disable this behaviour by setting HOMEBREW\_NO\_INSTALL\_CLEANUP.

Hide these hints with HOMEBREW\_NO\_ENV\_HINTS (see `man brew`).

Removing: /Users/macbook/Library/Caches/Homebrew/git--Net::SMTP::SSL--1.04.tar.gz... (2.4KB)

==> **Caveats**

==> **git**

The Tcl/Tk GUIs (e.g. gitk, git-gui) are now in the `git-gui` formula.

Subversion interoperability (git-svn) is now in the `git-svn` formula.

# Создание локального репозитория и коммит

Настройка репозитория:

С помощью cd путь перемещаемся в папку, где хотим сделать локальный репозиторий и пишем git init

macbook@MacBook-Pro-macbook Мальков % cd "Лабораторные работы"

macbook@MacBook-Pro-macbook Лабораторные работы % git init

Initialized empty Git repository in /Users/macbook/Desktop/4 курс/Мальков/Лабораторные работы/.git/

macbook@MacBook-Pro-macbook Лабораторные работы %

Первый коммит:

Пишем git add. Для фиксации текущий изменений, а затем git commit -m для непосредственного коммита

macbook@MacBook-Pro-macbook Лабораторные работы % git add .

macbook@MacBook-Pro-macbook Лабораторные работы % git commit -m "Initial commit"

[main (root-commit) 07d8e6f] Initial commit

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 "\320\222\320\260\321\200\320\270\320\260\320\275\321\202 2 \320\233\320\260\320\261 1.docx"

create mode 100644 "\320\222\320\260\321\200\320\270\320\260\320\275\321\202 2 \320\233\320\260\320\261 2.docx"

macbook@MacBook-Pro-macbook Лабораторные работы %

# Новые ветки и их слияние

Создаем новую ветку командой macbook@MacBook-Pro-macbook Лабораторные работы % git checkout -b branch\_2

Switched to a new branch 'branch\_2'

Добавляем в нее изменения и коммитим стандартной конструкцией git add . а затем git commit -m «Дублируем отчет 2»

Переключаемся на основную ветку macbook@MacBook-Pro-macbook Лабораторные работы % git checkout main

Switched to branch 'main'

Добавляем в нее изменения и коммитим стандартной конструкцией git add . а затем git commit -m «Дублируем отчет 1»

При этом, после переключения на main ветку, дублированный отчет 2 не будет видно, т.к он находится в другой

Проделываем слияние веток (второй в мейн) с помощью команды

macbook@MacBook-Pro-macbook Лабораторные работы % git merge branch\_2

Merge made by the 'ort' strategy.

.DS\_Store | Bin 0 -> 6148 bytes

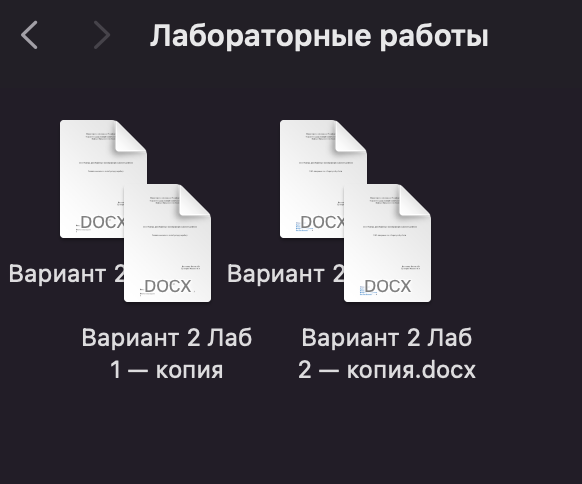
...20\261 2\302\240\342\200\224 \320\272\320\276\320\277\320\270\321\217.docx" | Bin 0 -> 361255 bytes

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 .DS\_Store

create mode 100644 "\320\222\320\260\321\200\320\270\320\260\320\275\321\202 2 \320\233\320\260\320\261 2\302\240\342\200\224 \320\272\320\276\320\277\320\270\321\217.docx"

Получается обе копии отчетов должны находиться в основной ветке (так и есть)



Проверяем изменение веток (смотрим историю)

macbook@MacBook-Pro-macbook Лабораторные работы % git log --oneline --graph --all

\* 4742c87 (**HEAD -> main**) Merge branch 'branch\_2'

|\

| \* 164c0ed (**branch\_2**) Дублирование отчета за 2 лр

\* | 3bc5512 Дублирование отчета за 1 лр

|/

\* 07d8e6f Initial commit

# Создание удаленного репозитория и загрузка проекта

Для того, чтобы создать удаленный репозиторий необходимо проделать схожие этапы, как и при локальном:

Перейти в желаему папку

Прописать git init

Далее происходит ключевое отличие. Для настройки удаленного репозитория нужен PAT (персональный токен доступа), который мы генерируем на своей странице гитхаба в разделе для разработчиков – персональный токен доступа. При его генерации можно настроить права доступа по этому токену.

Затем пишем команду в следующем виде

Git pull remote main

git remote set-url origin https://<YOUR\_TOKEN>@github.com/username/repository.git

Мы синхронизируем с репозитиорием сервера, свой локальный и подключаем его используя токен, юзернейм и название репа

Далее прописываем несколько команд настройки:

git config pull.rebase false

git config pull.rebase true

git config pull.ff only

git add .

git commit -m “Some text”

Все. Теперь после любых изменений на локальном пишем

git add .

git commit -m “Some text”

При изменениях на глобальном

Git push origin main