**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**по лабораторной работе № 2**

*дисциплина: Архитектура компьютера*

Студент: Бызова М.О.

Группа: НПИбд-01-23

**МОСКВА**

2023 г.

**Содержание.**

**Цель работы………………………………………………………….. 3   
  
Теоретическое введение…………………………………………….. 4  
  
Выполнение лабораторной работы……………………………….. 5**

1. **Настройка Github ………………………………................................ 5**
2. **Базовая настройка Git……………………………............................. 6**
3. **Создание SSH ключа………………………………………………… 7**
4. **Создание рабочего пространства и репозитория курса на основе шаблона.………………………. ……………………………………… 8**
5. **Создание репозитория курса на основе шаблона………………… 9**
6. **Настройка каталога курса…………………………………………... 11**

**Описание результатов выполнения заданий для   
самостоятельной работы……………………………………………. 13  
  
Выводы………………………………………………………………… 16**

**Цель работы.**Целью данной лабораторной работы является изучение идеологии и применение средств контроля версий, приобретение практических навыков по работе с системой git.

**Теоретическое введение.**Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется. В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельта-компрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить изменения, сделанные разными участниками, вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет 7 другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом привилегированный доступ только одному пользователю, работающему с файлом. Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить. В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным. Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд. Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями. Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией. Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений). Затем можно вносить изменения в локальном дереве и/или ветке. После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории.

**Выполнение лабораторной работы.**

1. **Настройка Github.**

Чтобы приступить к настройке, зайдя через браузер на сайт [GitHub](https://github.com/), создаем учетную запись или входим в ранее уже созданную. При необходимости заполняем основные данные. Для выполнения лабораторной работы я воспользуюсь ранее созданной учетной записью. (рис. 1-3)

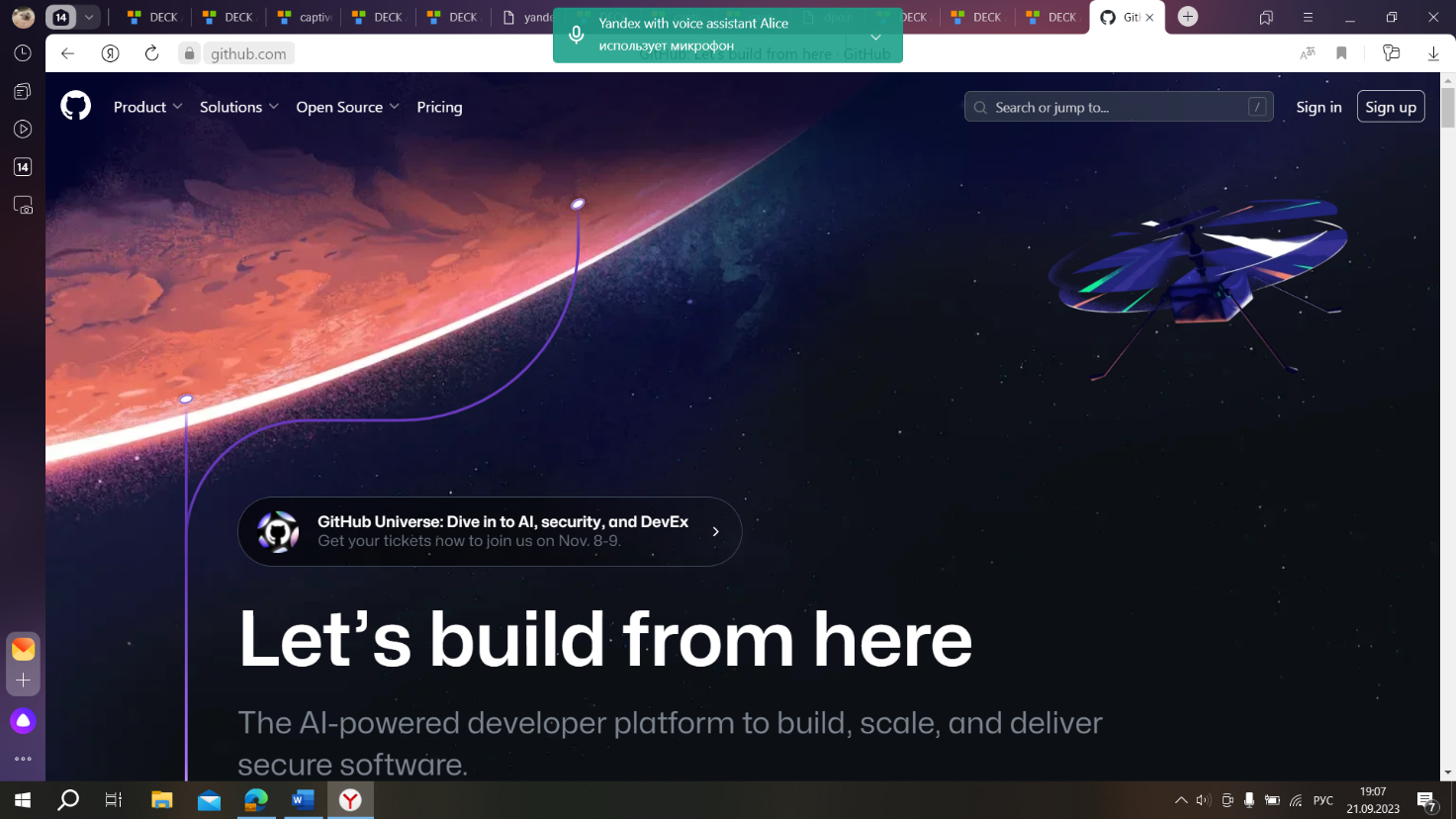
****

Рисунок 1. Главная страница сайта Github.

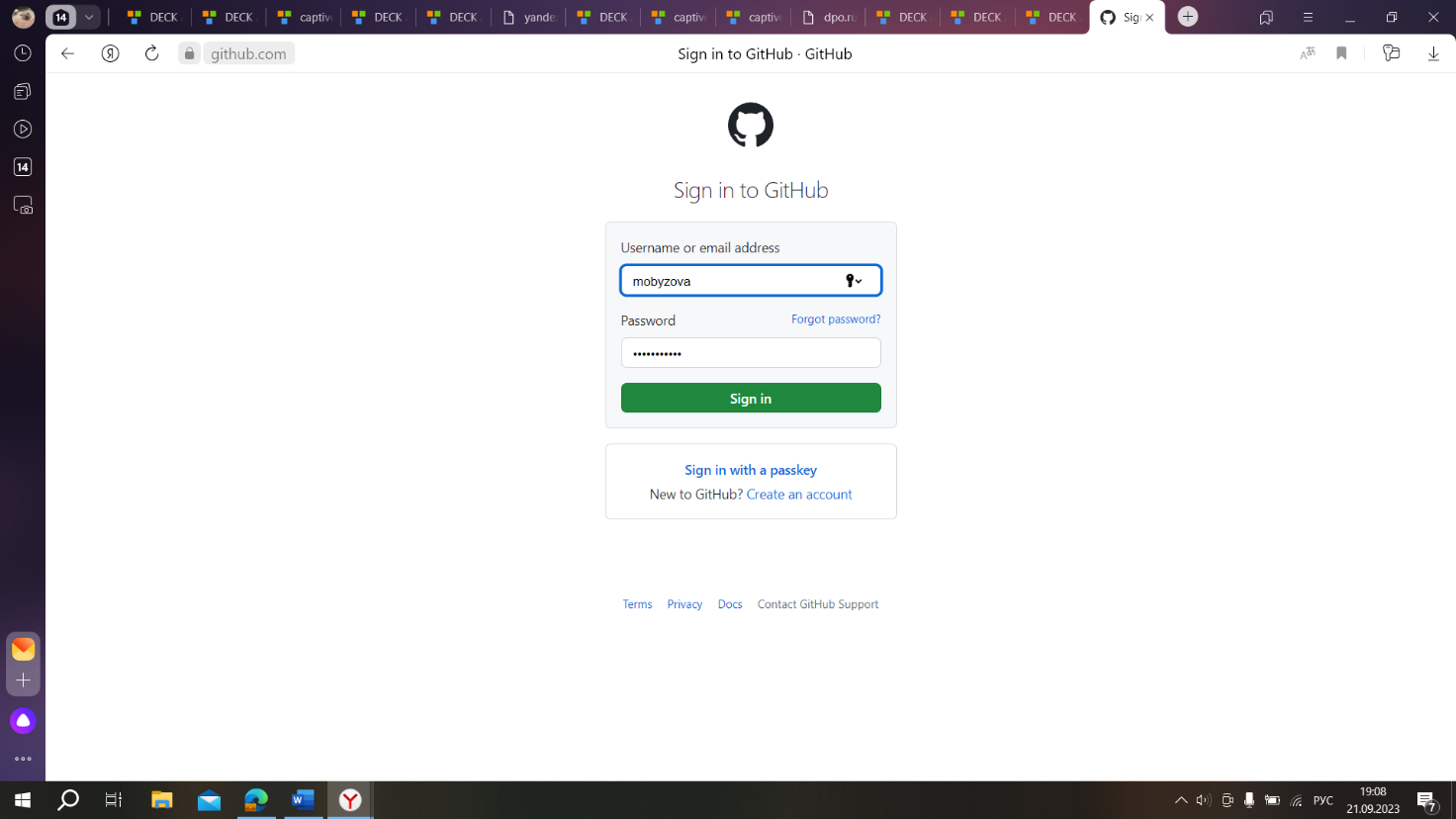


Рисунок 2. Вход в ранее созданный аккаунт.

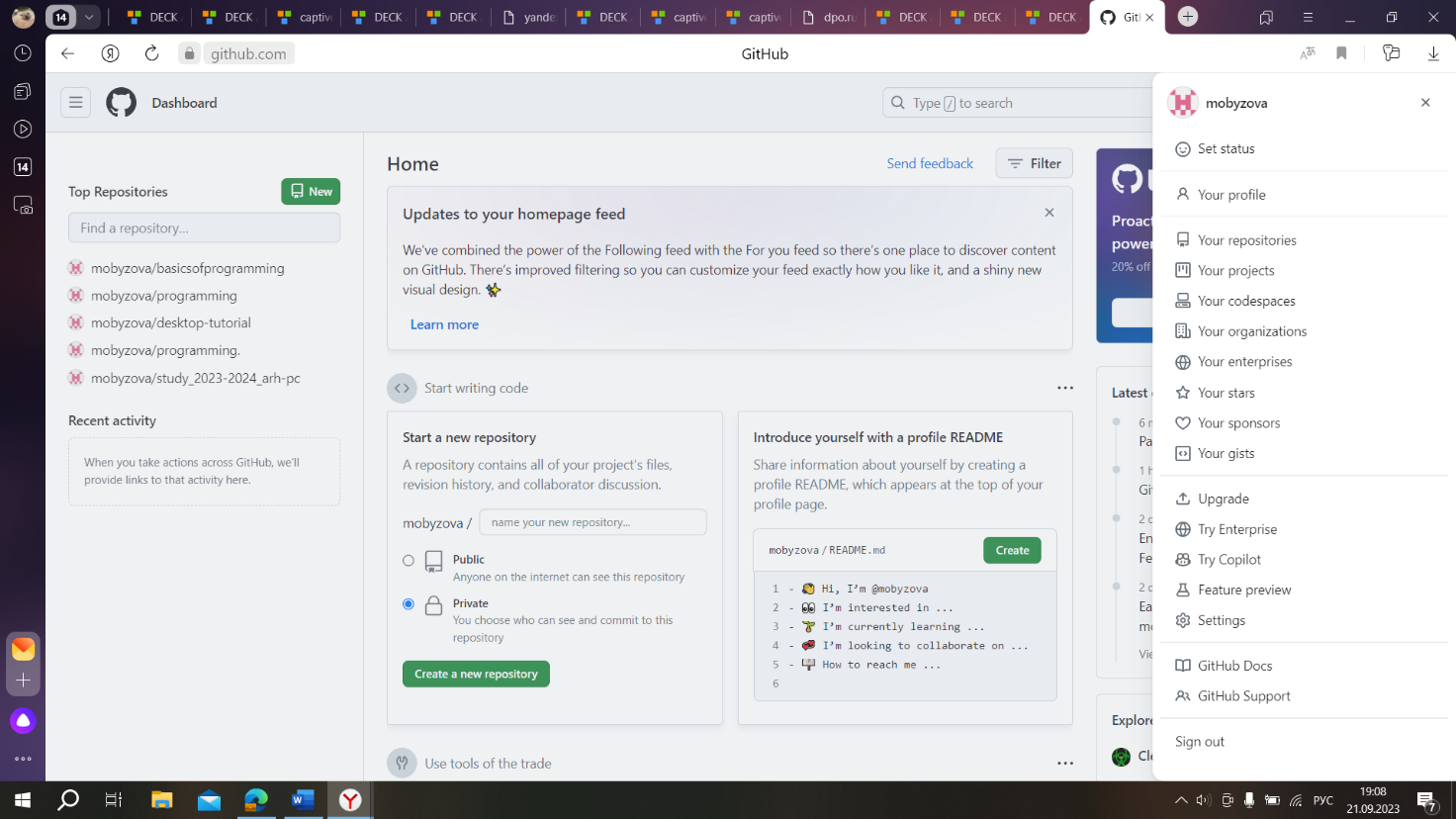


Рисунок 3. Аккаунт Github.

1. **Базовая настройка Git.**

Для дальнейшей настройки Git открываем терминал и вводим следующие команды: команду git config –global user.name “”, указывая свое имя, и команду git config –global user.email “work@mail”, указывая в ней электронную почту владельца репозитория (рис. 4).

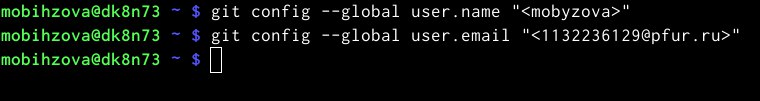
****

Рисунок 4. Предварительная конфигурация git.

Настраиваем utf-8 в выводе сообщений git для корректного отображения символов (рис. 5).

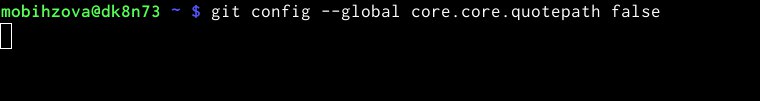


Рисунок 5. Настройка кодировки

Далее зададим имя начальной ветки. Назовём её «master» (рис. 6).



Рисунок 6. Создание имени для начальной ветки.

Зададим параметр autocrlf со значением input для того, чтобы конвертировать CRLF в LF только при коммитах. Уточним, что CR и LF – это символы, которые можно использовать для обозначения разрыва строки в текстовых файлах (рис. 7)

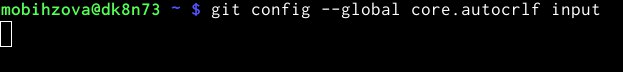


Рисунок 7. Параметр autocrlf.

Далее зададим параметр safecrlf со значением warn. Таким образом, Git будет проверять преобразование на обратимость. В дальнейшей работе при значении warn Git только выведет предупреждение, но будет принимать необратимые конвертации (рис. 8).

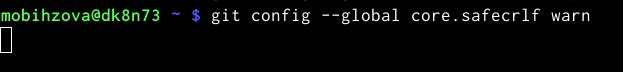


Рисунок 8. Параметр safecrlf.

1. **Создание SSH ключа.**

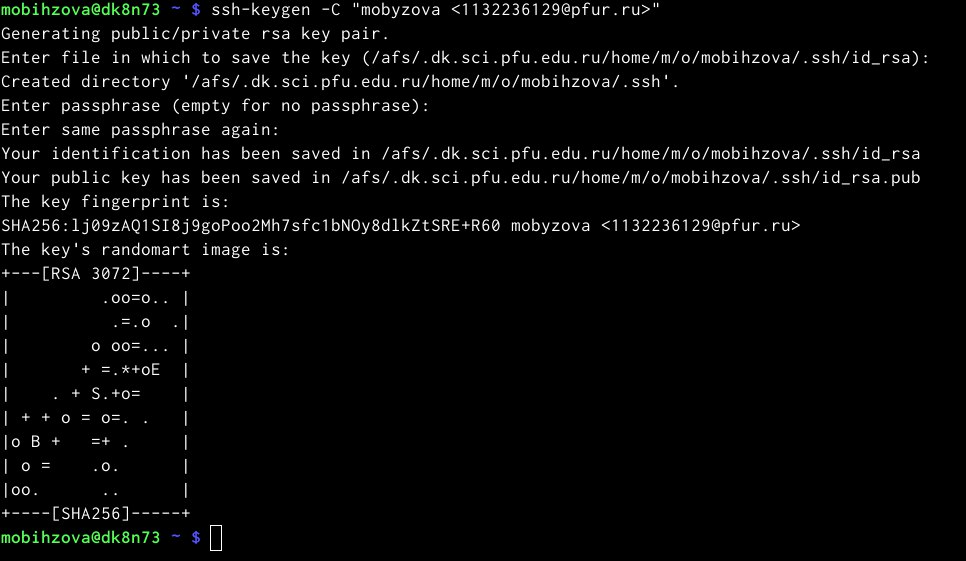
Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый). Для этого вводим команду ssh-keygen -C “Имя Фамилия, work@email”, указывая имя владельца и электронную почту владельца. Ключ автоматически сохранится в каталоге ~/.ssh/ (рис. 9).  
  
****

Рисунок 9 . Создание SSH-ключа.

Далее копируем открытый ключ из директории, в которой он был сохранен, с помощью утилиты xclip (рис. 10).

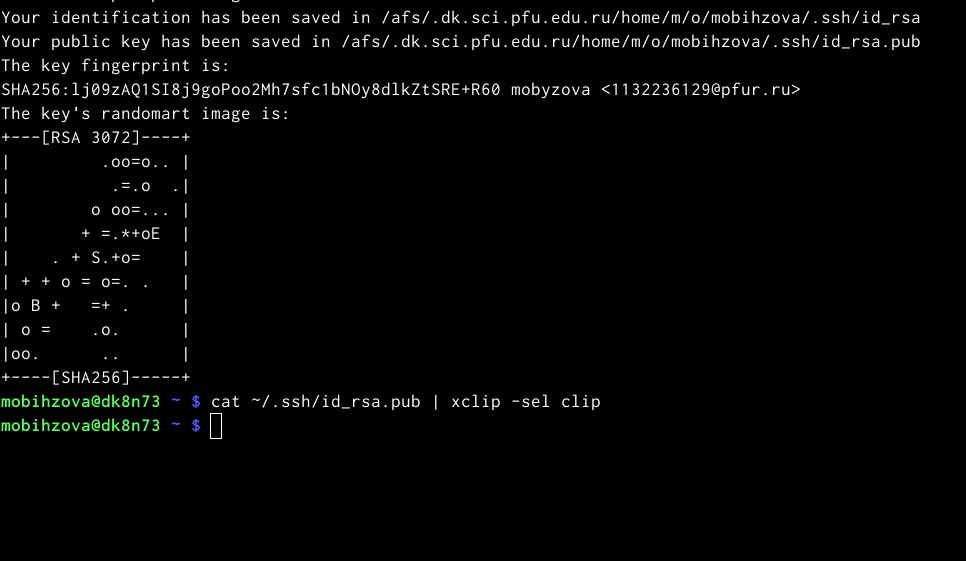


Рисунок 10. Копирование содержимого файла.

Далее необходимо загрузить сгенерённый открытый ключ. Для этого заходим на сайт http: //github.org/ под своей учётной записью и переходим в меню Setting. После этого выбираем в боковом меню SSH and GPG keys и нажимаем кнопку New SSH key. Вставляем ключ в появившееся на сайте поле и указываем для ключа имя (Title). Нажимаем «Add SSH-key», чтобы завершить добавление ключа (рис. 11).

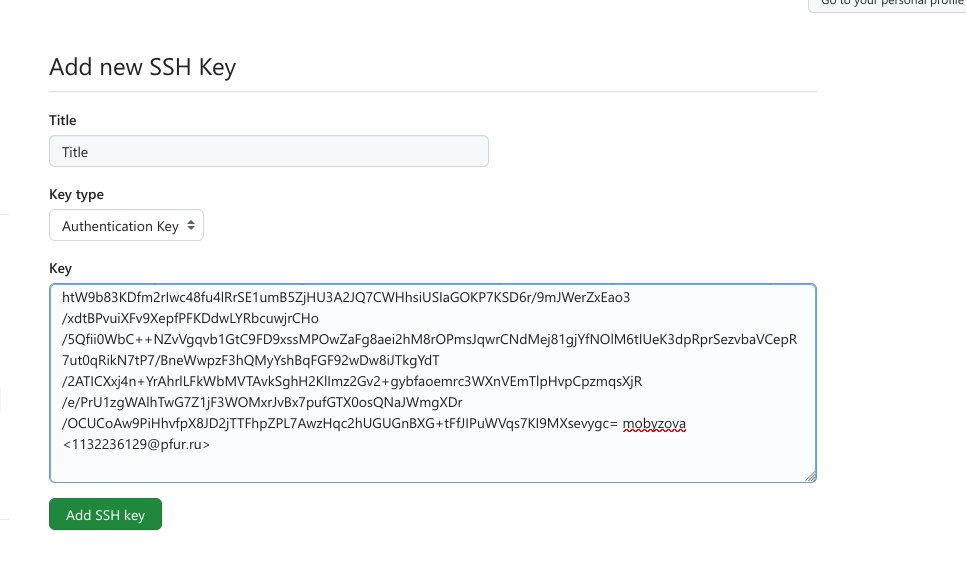


Рисунок 11. Добавление ключа.

1. **Создание рабочего пространства и репозитория курса на основе шаблона.**

После проделанных действий возвращаемся в терминал. Создаём каталог для предмета «Архитектура компьютера» с помощью утилиты mkdir с ключом -p. Далее проверяем работу команды (рис. 12-13) любым удобным способом.

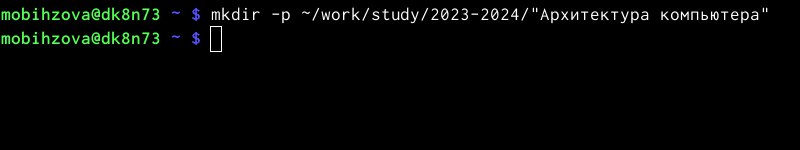


Рисунок 12. Создание рабочего пространства.

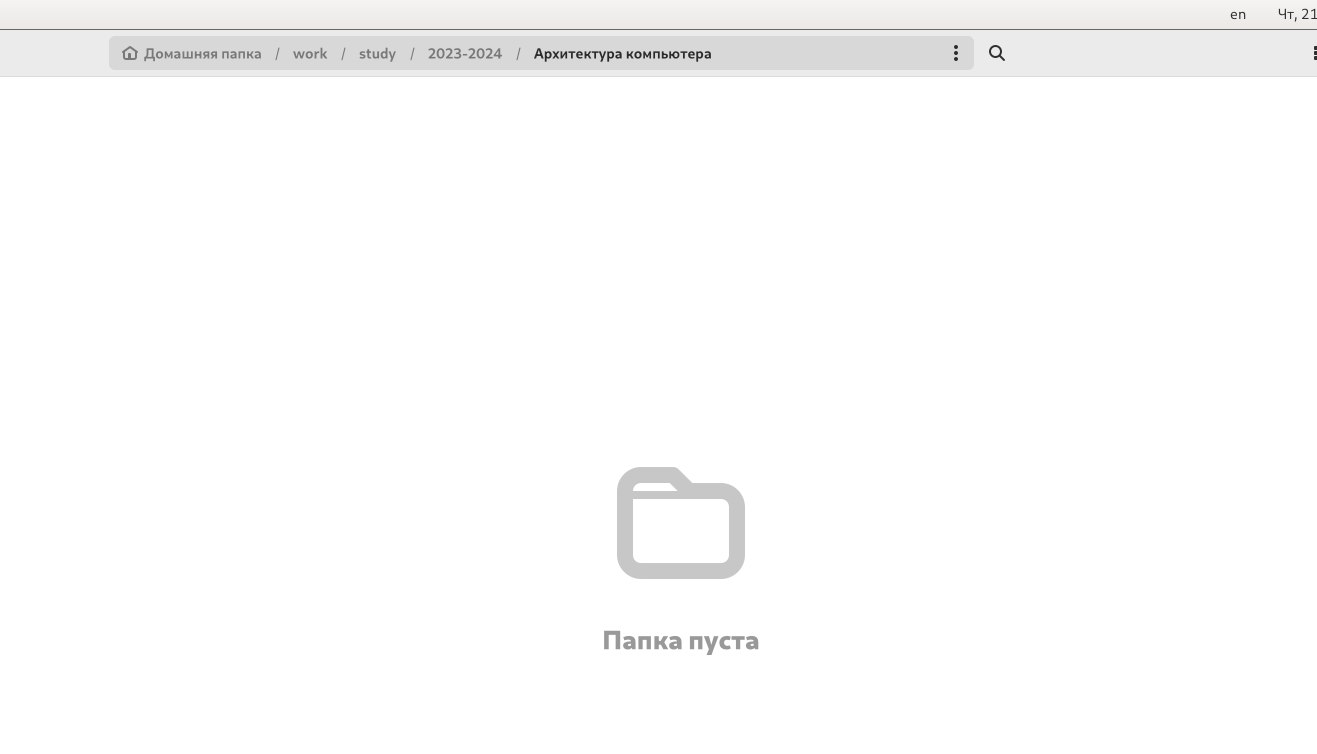


Рисунок 13. Проверка работы команды.

1. **Создание репозитория курса на основе шаблона.**

Далее в браузере переходим на страницу репозитория с шаблоном курса по адресу https://github.com/yamadharma/course-directory-student-template. Теперь выбираем «Use this template», чтобы использовать этот шаблон для своего репозитория (рис. 14).

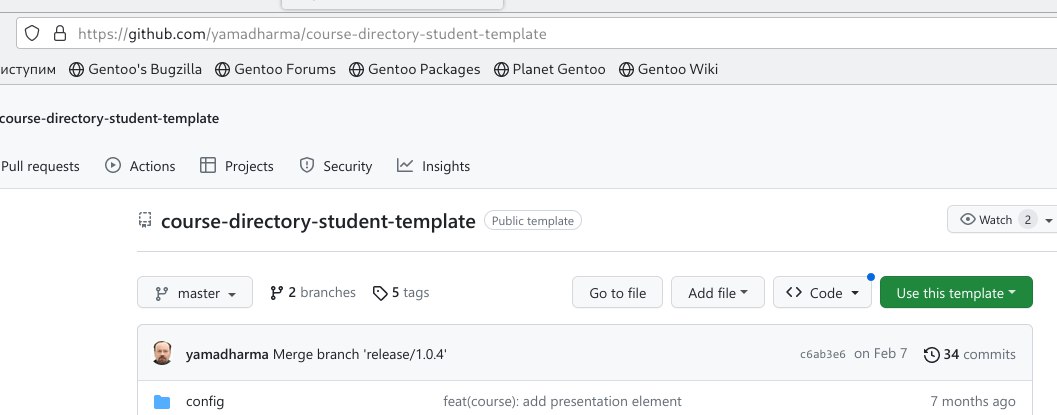


Рисунок 14. Страница шаблона для репозитория.

В открывшемся окне задаем имя репозитория (Repository name) «study\_2023–2024\_arh-pc» и создаем репозиторий (кнопка Create repository from template) (рис. 15).

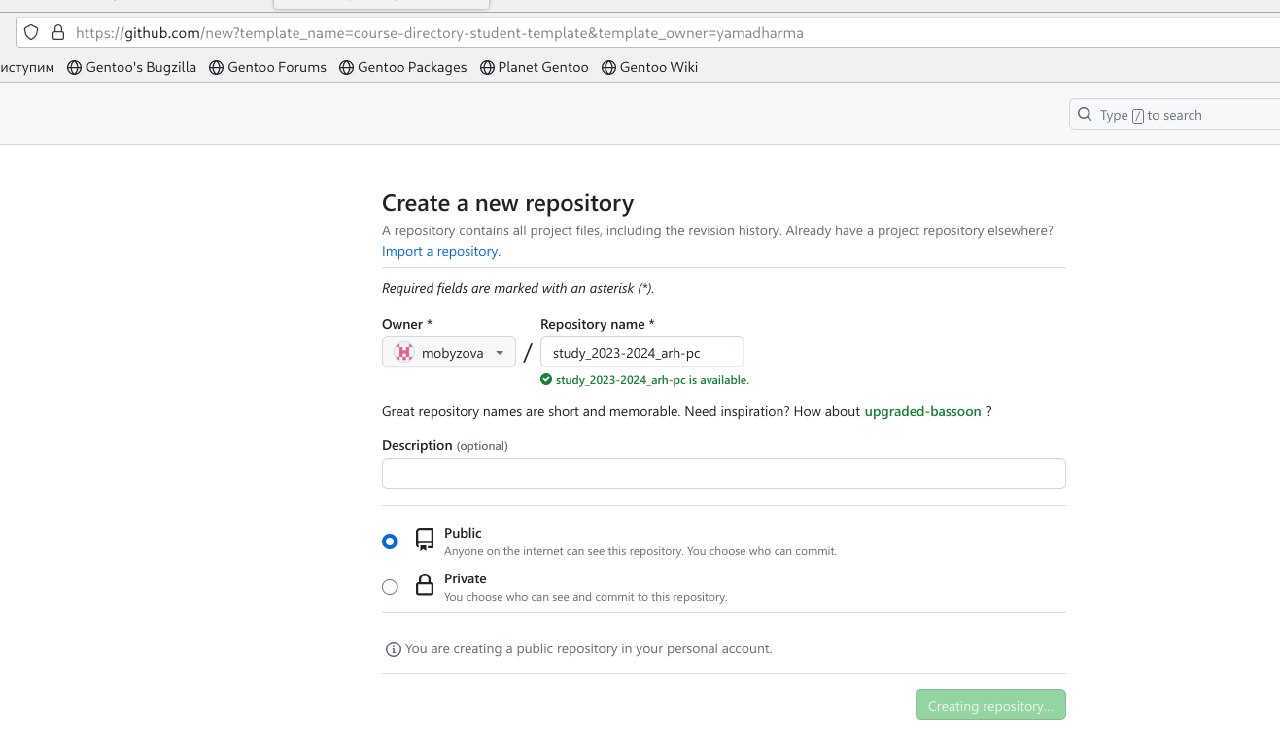


Рисунок 15. Окно создания репозитория.

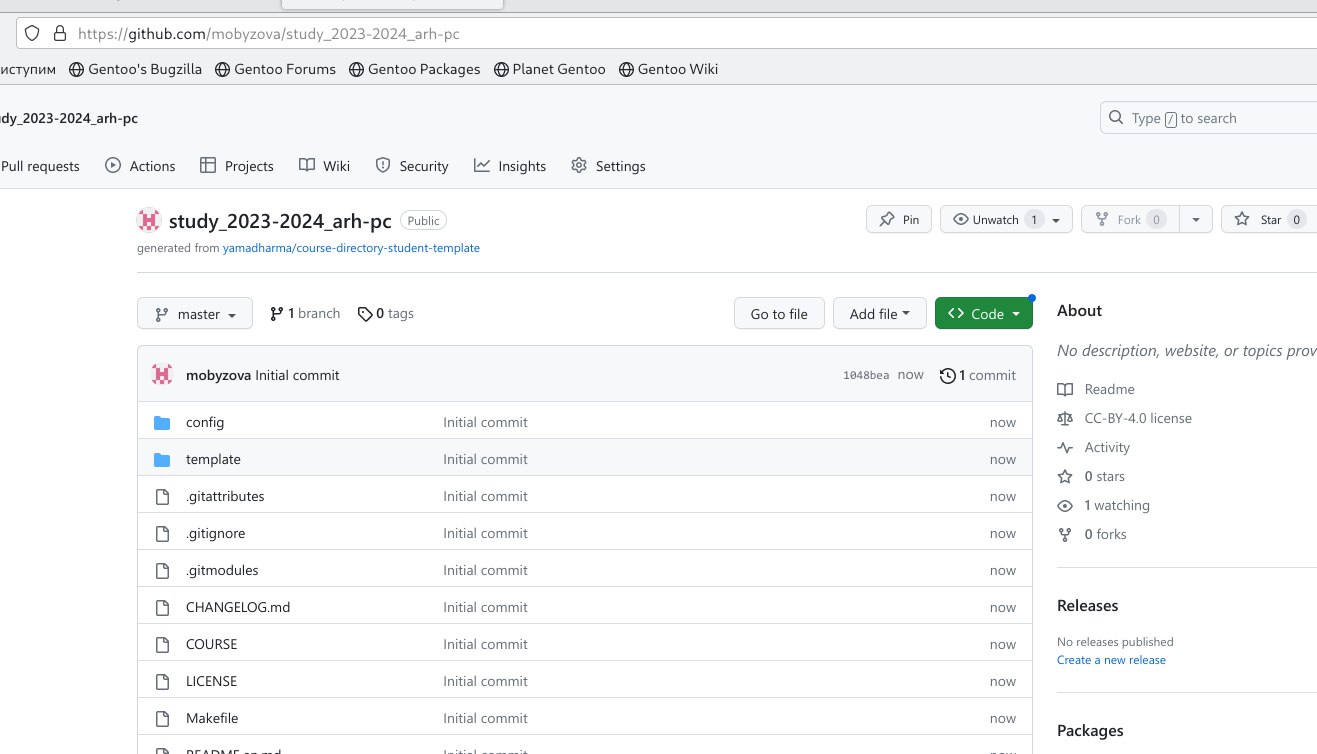
Таким образом, репозиторий успешно создан (рис. 16).  
  


Рисунок 16. Созданный репозиторий.

Для дальнейшей работы через терминал переходим в созданный каталог курса с помощью уже известной нам утилиты cd (рис. 17).

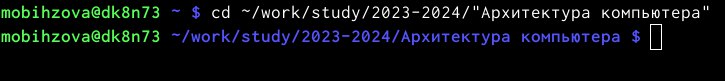


Рисунок 17. Перемещение между директориями.

Теперь клонируем созданный репозиторий с помощью команды git clone –recursive git@github.com:/study\_2023–2024\_arh-pc.git arch-pc, предварительно скопировав ссылку для клонирования на странице созданного репозитория: сначала переходим в окно «code», далее выбираем в окне вкладку «SSH» (рис. 18-19).

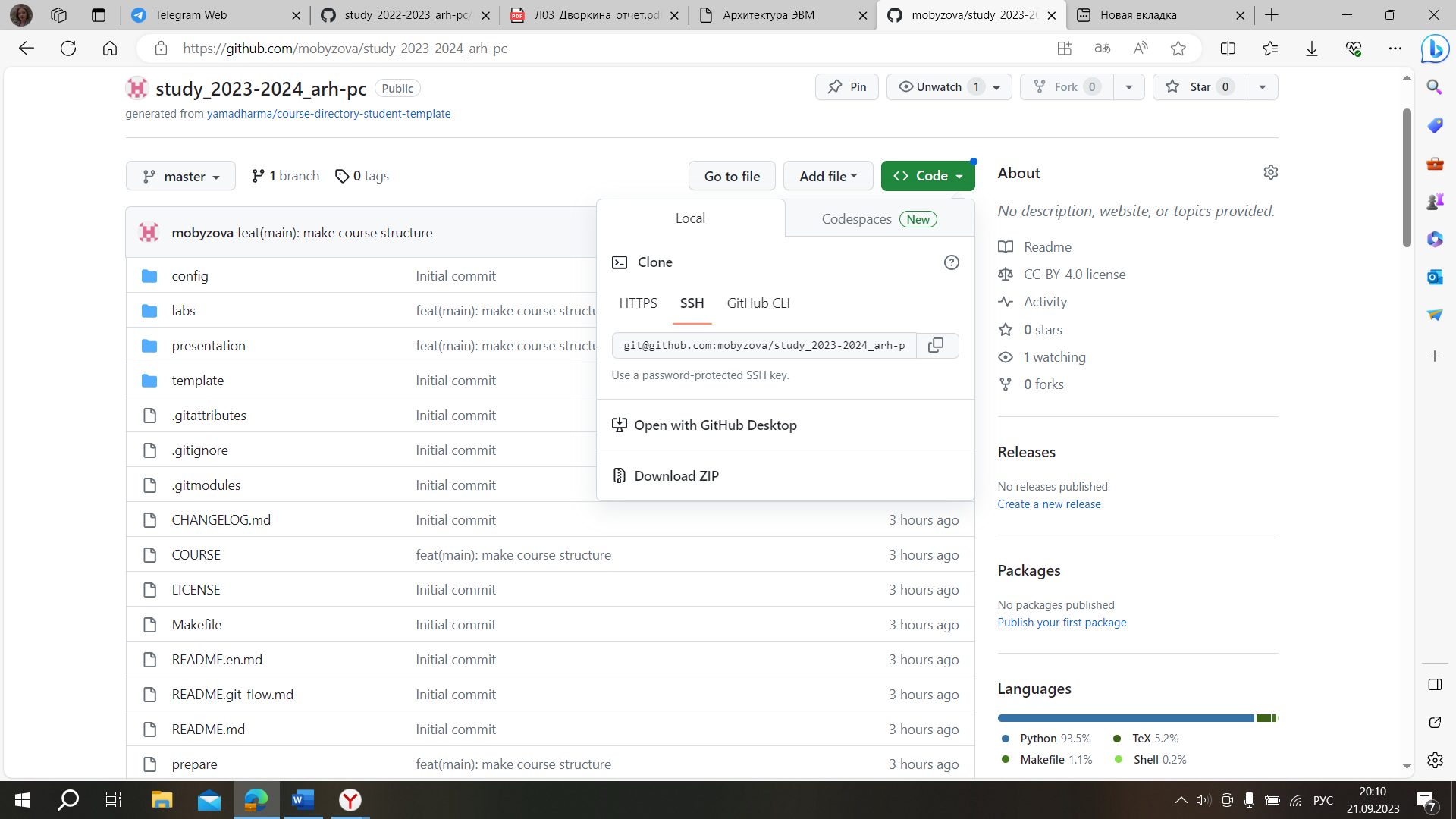


Рисунок 18. Окно с ссылкой для копирования репозитория.

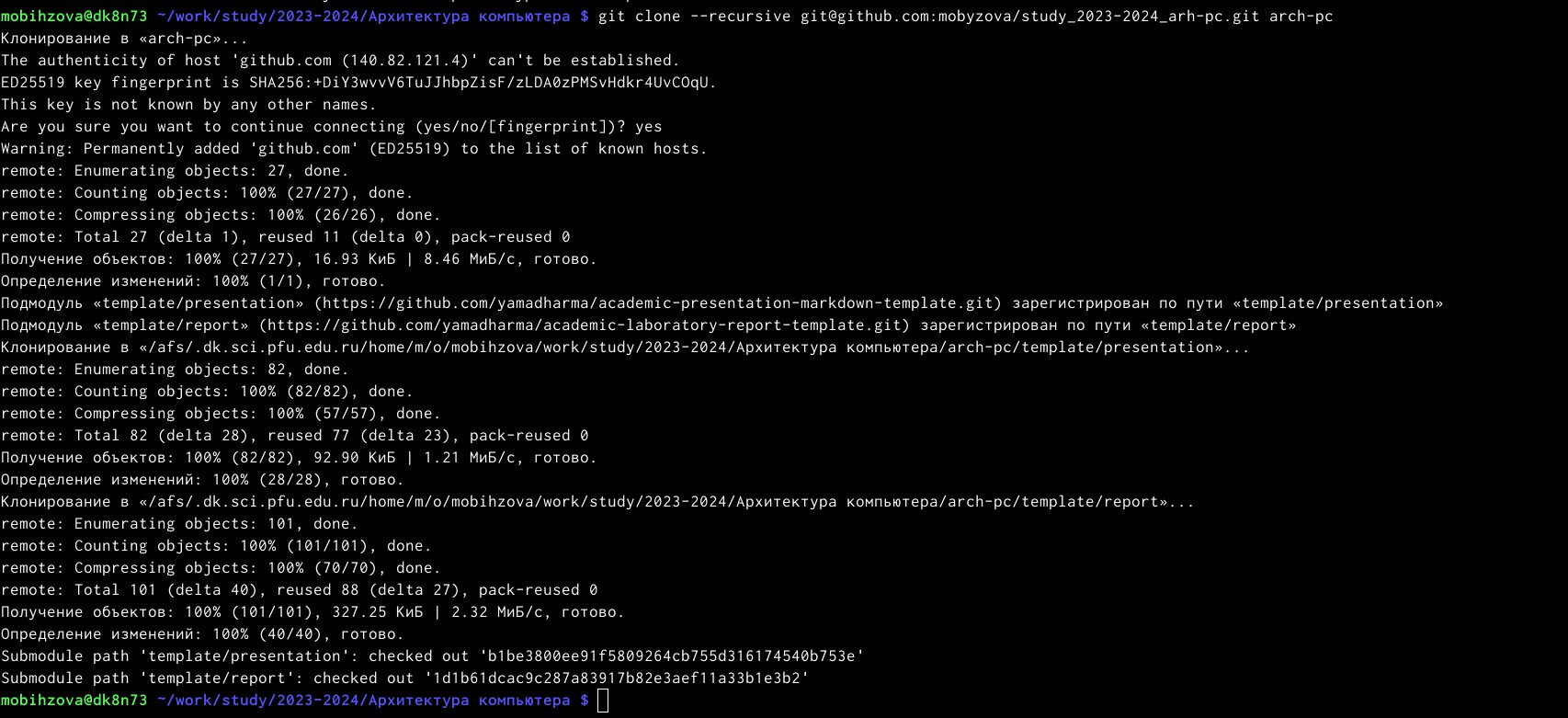


Рисунок 19. Клонирование репозитория.

Таким образом, репозиторий успешно скопирован.

1. **Настройка каталога курса.**

Для дальнейшей работы переходим в каталог курса с помощью утилиты cd (рис. 20).

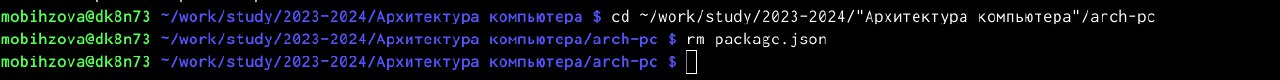


Рисунок 20. Перемещение между директориями.

Теперь удалим лишние файлы с помощью утилиты rm (рис. 21).

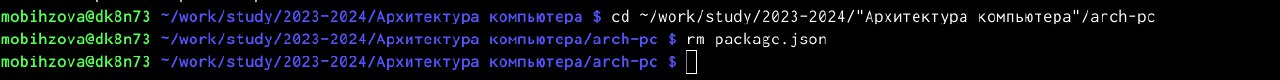


Рисунок 21. Удаление файлов.

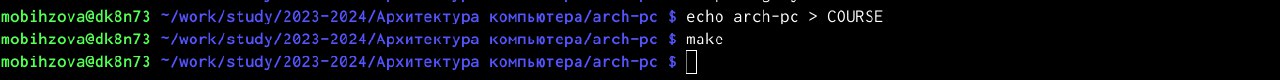
Создадим все необходимые каталоги при помощи ряда команд (рис. 22).  
  


Рисунок 22. Создание каталогов.

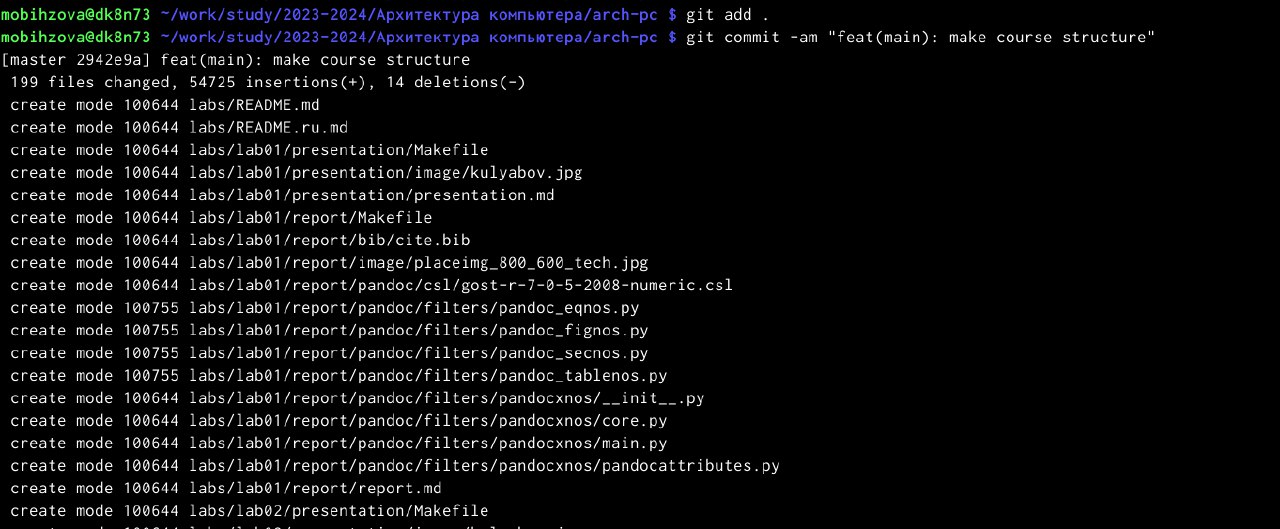
Отправим созданные нами каталоги на наш сервер с локального репозитория: для этого добавляем все созданные ранее каталоги при помощи git add, комментируем и сохраняем изменения на сервере как добавление курса с помощью git commit (рис. 23).   
  


Рисунок 23. Добавление и сохранение изменений на сервере.

При помощи push отправляем всё на сервер (рис. 24).

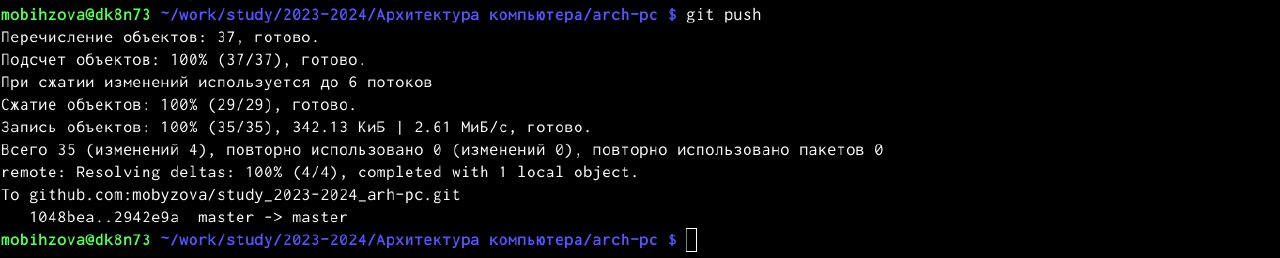


Рисунок 24. Выгрузка изменений на сервер.

В конце концов проверим правильность создания иерархии рабочего пространства в локальном репозитории и на странице github (рис. 25-26).

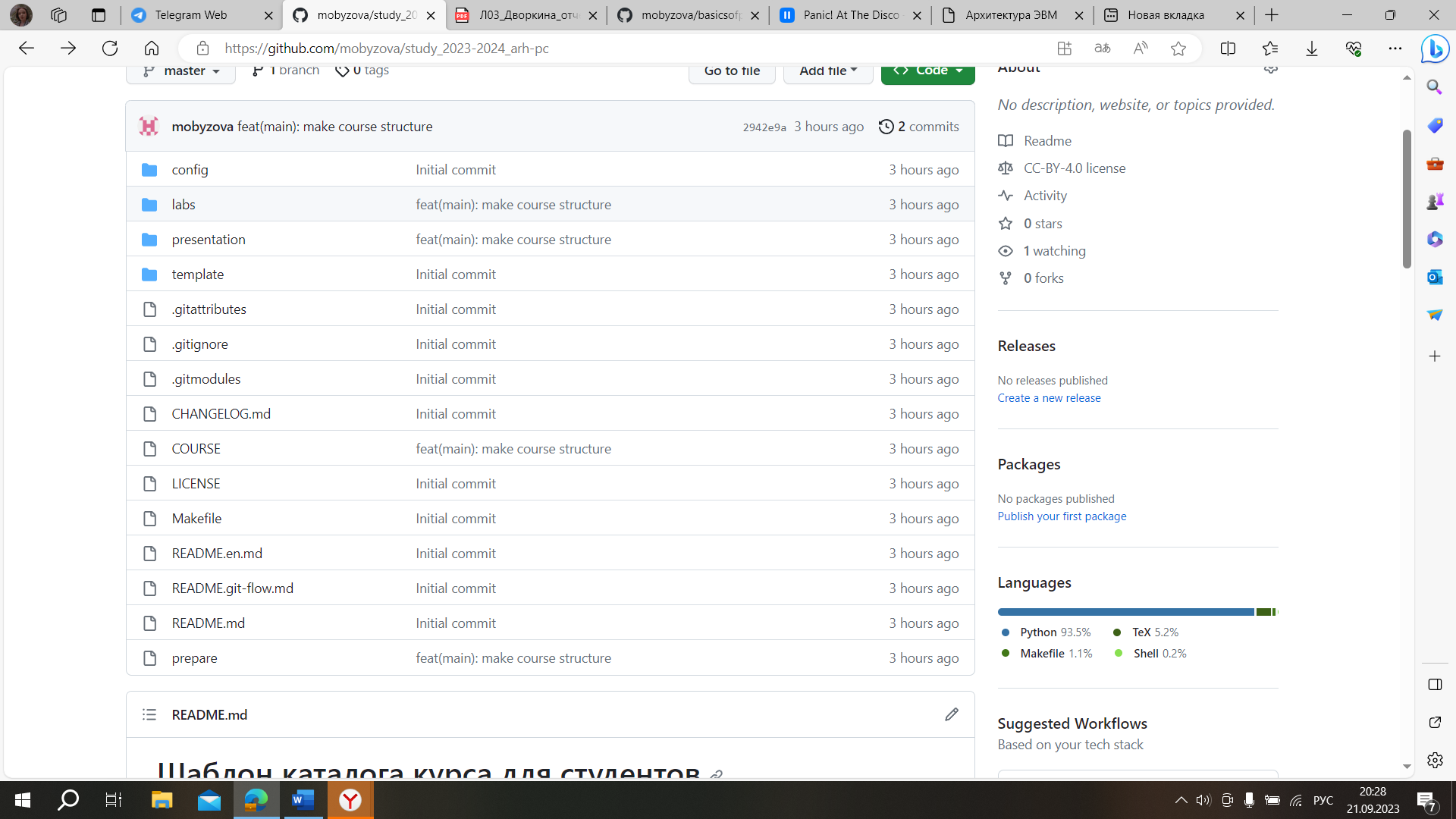


Рисунок 25. Страница репозитория (1).

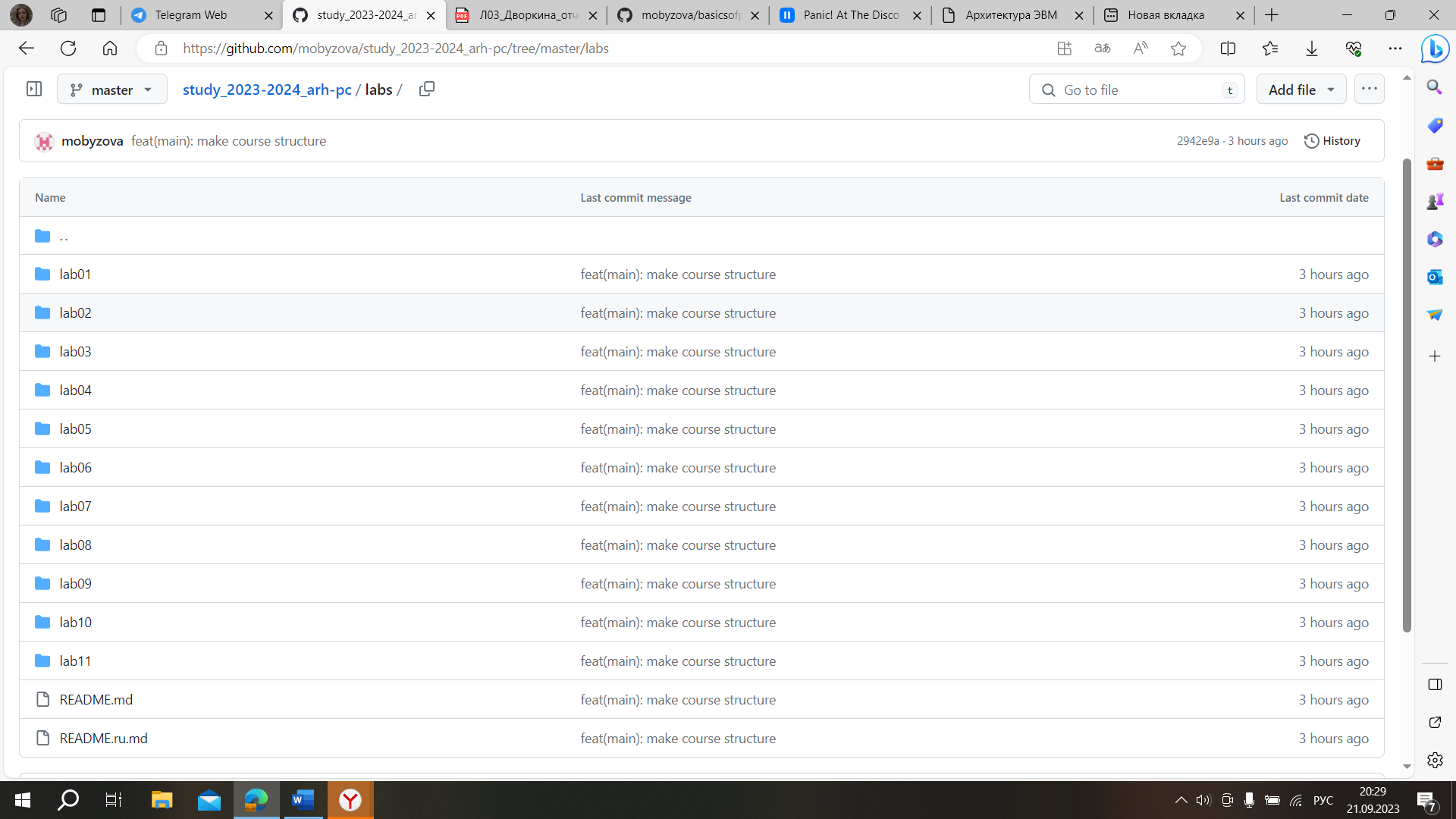


Рисунок 26. Страница репозитория (2).

**Описание результатов выполнения заданий для самостоятельной работы.**1.Для создания отчета по выполнению лабораторной работы в соответствующем каталоге рабочего пространства переходим в директорию labs/lab02/report с помощью утилиты cd. Создаем в каталоге файл для отчета по второй лабораторной работе с помощью утилиты touch (рис. 27-28).

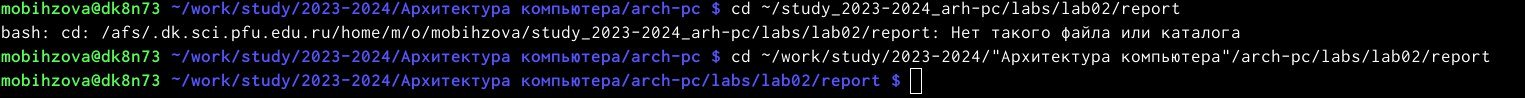
****

Рисунок 27. Перемещение между директориями.

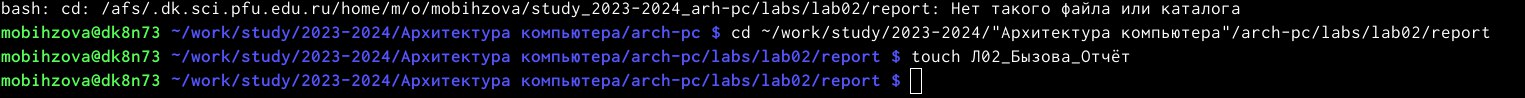
****

Рисунок 28. Создание файла.

2. Для копирования отчетов по выполнению предыдущих лабораторных работ в соответствующие каталоги созданного рабочего пространства проверим местонахождение файла с отчетом по первой лабораторной работе - он должен находиться в подкаталоге домашней директории «Загрузки», для проверки используем команду ls (рис. 29). Затем копируем необходимый файл с помощью утилиты cp (рис. 30). Проверяем правильность выполнения команды cp с помощью ls (рис. 31).

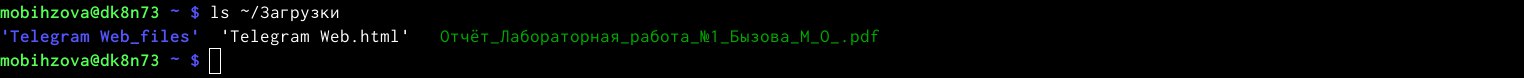


Рисунок 29. Проверка местонахождения файла.

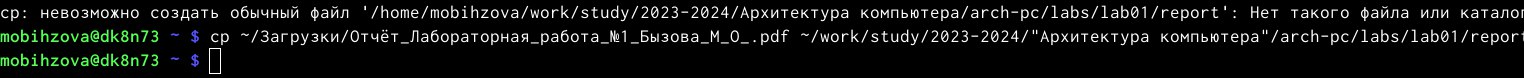


Рисунок 30. Копирование файл.

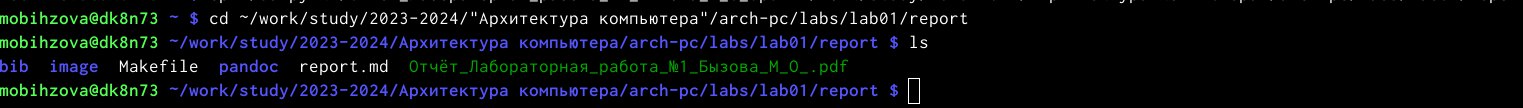


Рисунок 31. Проверка выполнения команды.

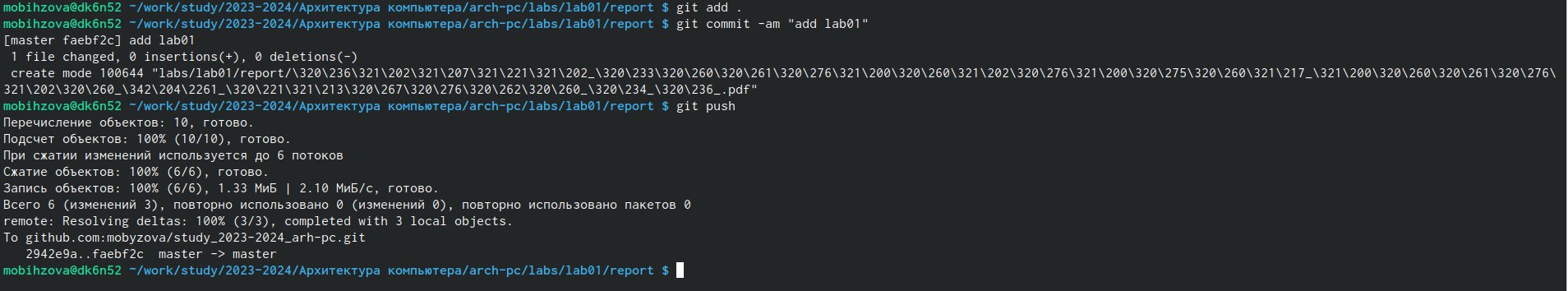
3. Далее с помощью ранее изученных команд загружаем файлы на GitHub: добавляем с помощью git add нужный файл, сохраняем изменения с помощью git commit, отправляем в центральный репозиторий сохраненные изменения командой git push. Проверяем правильность работы (рис. 32-35).   
  


Рисунок 32. Загрузка файла с отчетом по лабораторной работе №1 на GitHub.

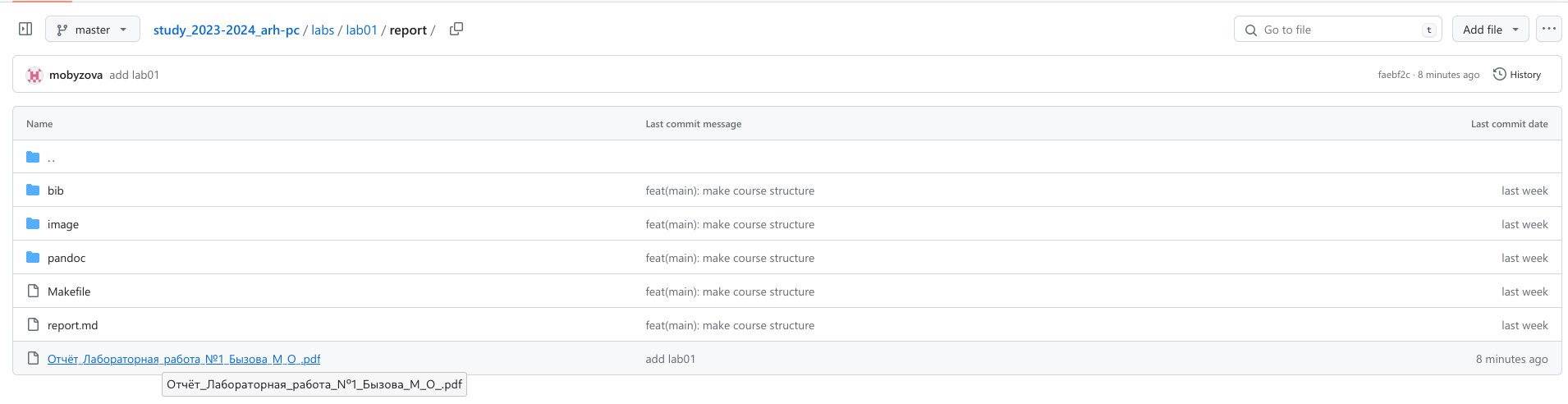


Рисунок 33. Проверка правильности работы команд.

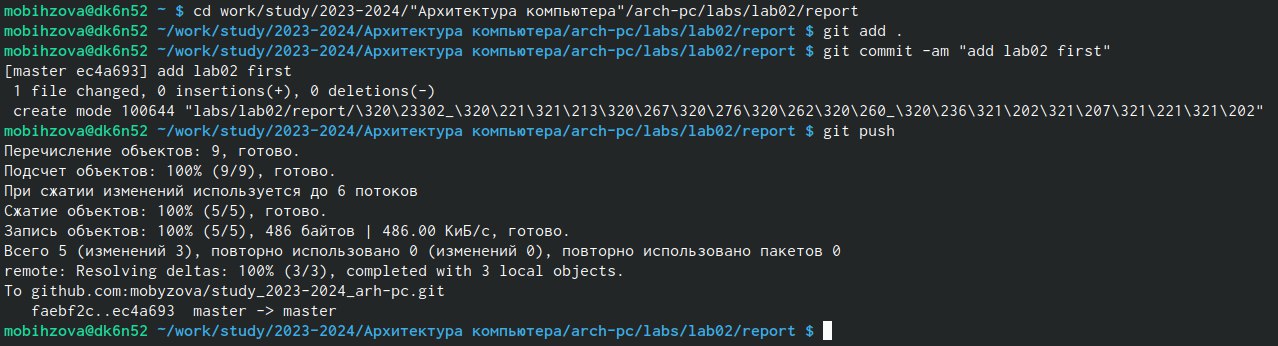


Рисунок 34. Загрузка файла с отчетом по лабораторной работе №2 на GitHub.

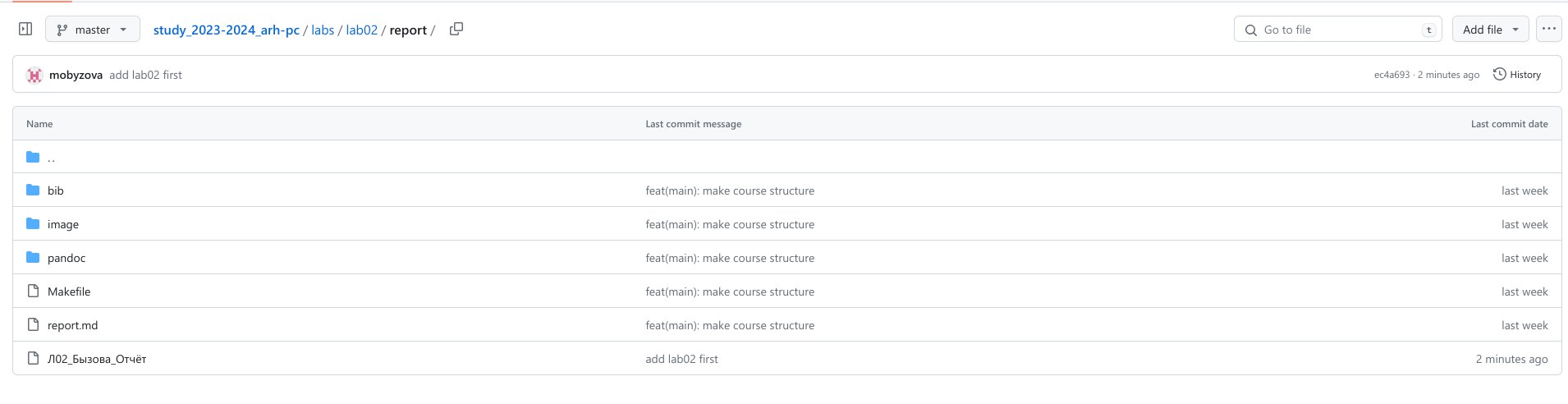


Рисунок 35. Проверка правильности работы команд.

Мы видим, что пояснение к совершенным действиям отображается, а отчеты по лабораторным работам находятся в соответствующих каталогах репозитория: отчет по первой - в lab01/report, по второй – в lab02/report. Убеждаемся, что всё выполнено верно.

**Выводы:**В ходе выполнения данной лабораторной работы были получены практические навыки работы с системой git, а также была изучена идеология и применение средств контроля версий посредству выполнения различных заданий, а именно:   
  
1. Настройки GitHub.   
2. Базовой настройки Git.  
3. Создания SSH-ключа.   
4. Создания рабочего пространства и репозитория курса на основе шаблона.   
5. Создания репозитория курса на основе шаблона.   
6. Настройки каталога курса.   
7. Выполнения заданий для самостоятельной работы для подведения итогов и закрепления полученных знаний.