**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**по лабораторной работе № 3**

*дисциплина: Архитектура компьютера*

Студент: Поляков Глеб Сергеевич

Группа: НПИбд 02 22

**МОСКВА**

2022 г.

**Содержание:**

1. **Цель работы 5**
2. **Задание 6**
3. **Теоретическое введение 7**
4. **Выполнение лабораторной работы 10**
5. **Вывод 17**

**Список иллюстраций:**

Рис. 1 10

Рис. 2 11

Рис. 3.1 12

Рис. 3.2 12

Рис. 3.3 12

Рис. 3.4 12

Рис. 4.1 13

Рис. 4.1 13

Рис. 5 14

Рис. 6.1 15

Рис. 6.2 15

Рис. 6.3 15

Рис. 6.4 15

Рис. 6.5 15

**Список таблиц:**

Табл. 1 7

Табл. 2 8-9

**1 Цель работы**

Целью работы является изучить идеологию и применение средств контроля версий. Приобрести практические навыки по работе с системой git.

**2 Задание**

Настройка github.

Базовая настройка git.

Создание SSH-ключа.

Создание рабочего пространства и репозитория курса на основе шаблона.

Создание репозитория курса на основе шаблона.

Настройка каталога курса.

Задания для самостоятельной работы.

**3. Теоретическое введение**

**3.2.1. Системы контроля версий. Общие понятия**

**Системы контроля версий (Version Control System, VCS)** применяются при работе нескольких человек над одним проектом. Обычно основное дерево про- екта 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 — со- хранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

Системы контроля версий поддерживают возможность отслеживания и разре- шения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависи- мости от настроек блокировка не позволяет 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 изме- нений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, ко- гда 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 цен- тральный репозиторий не является обязательным.

Среди классических VCS наиболее известны CVS, Subversion, а среди распре- делённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

**3.2.2. Система контроля версий Git**

**Система контроля версий Git** представляет собой набор программ команд- ной 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 копировани- ем или архивацией.

**3.2.3. Основные команды git**

Наиболее часто используемые команды git представлены в таблице 3.1. **Таблица 3.1.** Основные команды git

Команда

**git init** - создание основного дерева репозитория

**git pull** - получение обновлений (изменений) текущего дерева из центрального репозитория

**git push** - отправка всех произведённых изменений локального дерева в центральный репозиторий

**git status** - просмотр списка изменённых файлов в текущей директории

**git diff** - просмотр текущих изменения

**git add .** - добавить все изменённые и/или созданные файлы и/или каталоги

**git add имена\_файлов** - добавить конкретные изменённые и/или созданные файлы и/или каталоги

**git rm имена\_файлов** - удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории)

**git commit -am 'Описание коммита'** - сохранить все добавленные изменения и все изменённые файлы

Команда

**git checkout -b** имя\_ветки - создание новой ветки, базирующейся на текущей

**git checkout** имя\_ветки - переключение на некоторую ветку (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой)

**git push** отправка изменений конкретной ветки в центральный репозиторий

**origin** имя\_ветки

**git merge --no-ff** имя\_ветки - слияние ветки с текущим деревом

**git branch -d** имя\_ветки удаление локальной уже слитой с основным деревом ветки принудительное удаление локальной ветки

**git branch -D** имя\_ветки - удаление ветки с центрального репозитория

**git push**

**origin:имя\_ветки**

Описание

**3.2.4. Стандартные процедуры работы при наличии центрального репозитория**

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):

**git checkout master**

**git pull**

**git checkout** **-b** имя\_ветки

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. Для этого необходимо проверить, какие файлы изменились к текущему моменту:

**git status**

и при необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.

Затем полезно просмотреть текст изменений на предмет соответствия прави- лам ведения чистых коммитов:

**git diff**

Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями:

**git add** имена\_файлов

**git rm** имена\_файлов

Если нужно сохранить все изменения в текущем каталоге, то используем:

**git add .**

Затем сохраняем изменения, поясняя, что было сделано:

**git commit -am "Some commit message"**и отправляем в центральный репозиторий: **git push origin имя\_ветки** или **git push**

1. **Выполнение лабораторной работы**
2. Настройка GitHub

**Graphical user interface, application

Description automatically generated**

Рис. 1. Созданная учетная запись GitHub

1. Базовая настройка git

Text

Description automatically generated

Рис. 2.1 Настройка локальной учетной записи gitText

Description automatically generatedText

Description automatically generated

Рис. 2.2 Настройка вывода путей Рис. 2.3 Настройка проверки обратимости изменений

Text

Description automatically generated

Рис. 2.3 Настройка вывода

1. Создание SSH ключа

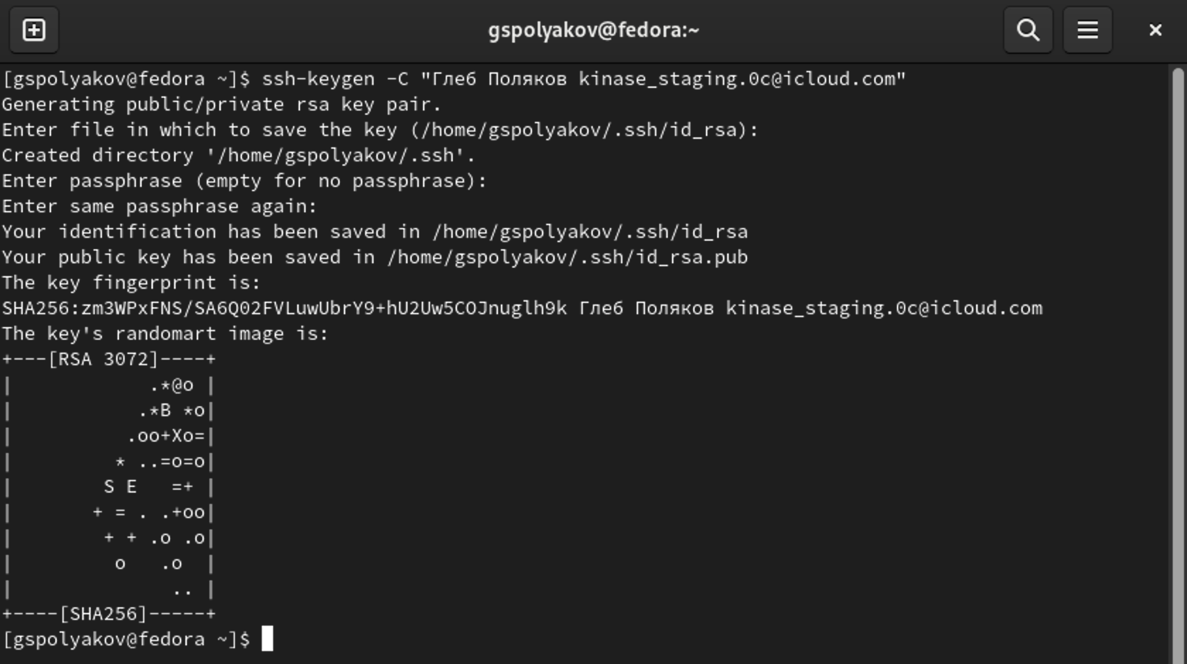


Рис. 3.1 Создание SSH ключа

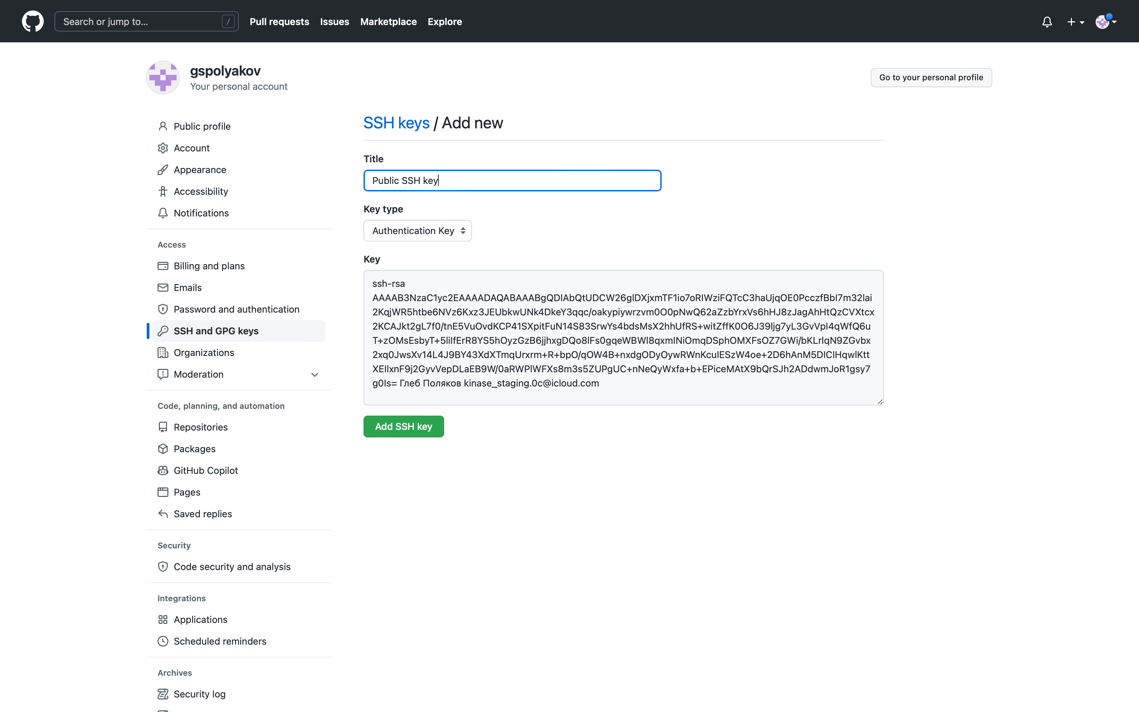


Рис. 3.2 Добавление SSH ключа в учетную запись GitHub

1. Создание рабочего пространства и репозитория курса на основе шаблона

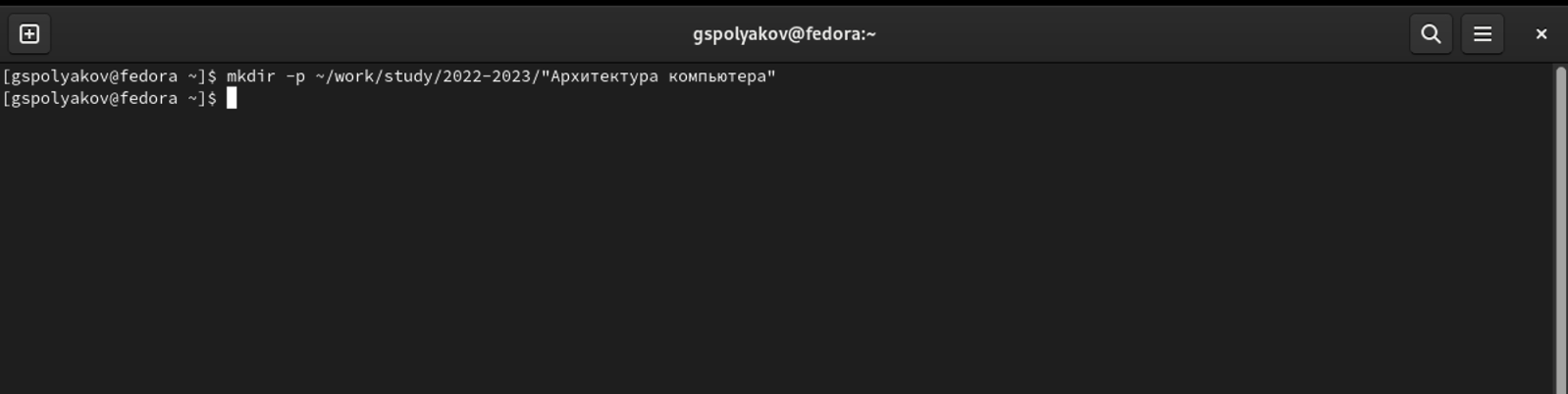


Рис. 4 Создание структуры и директорий

1. Создание репозитория курса на основе шаблона

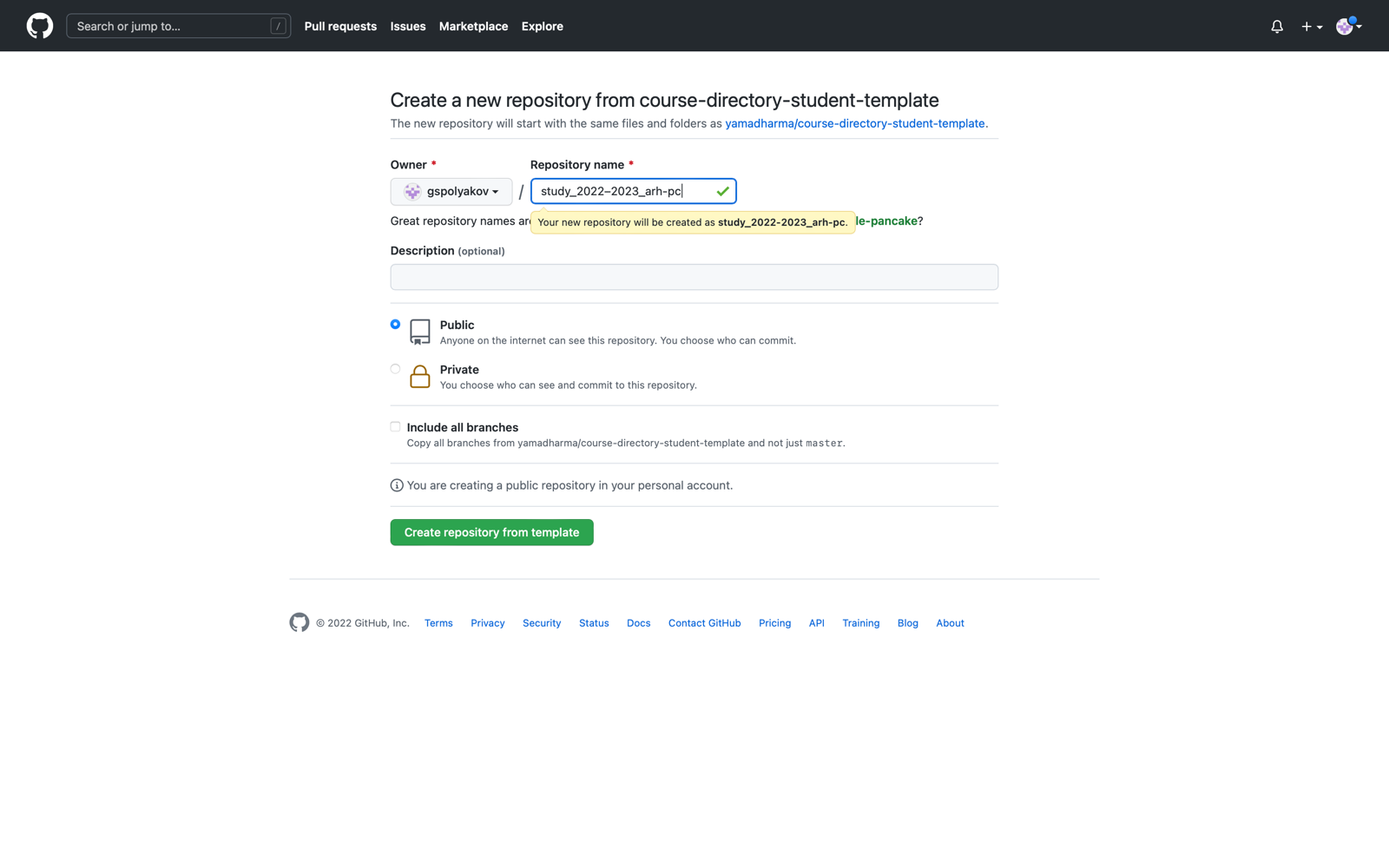


Рис. 5.1 Создание репозитория по шаблону на GitHub

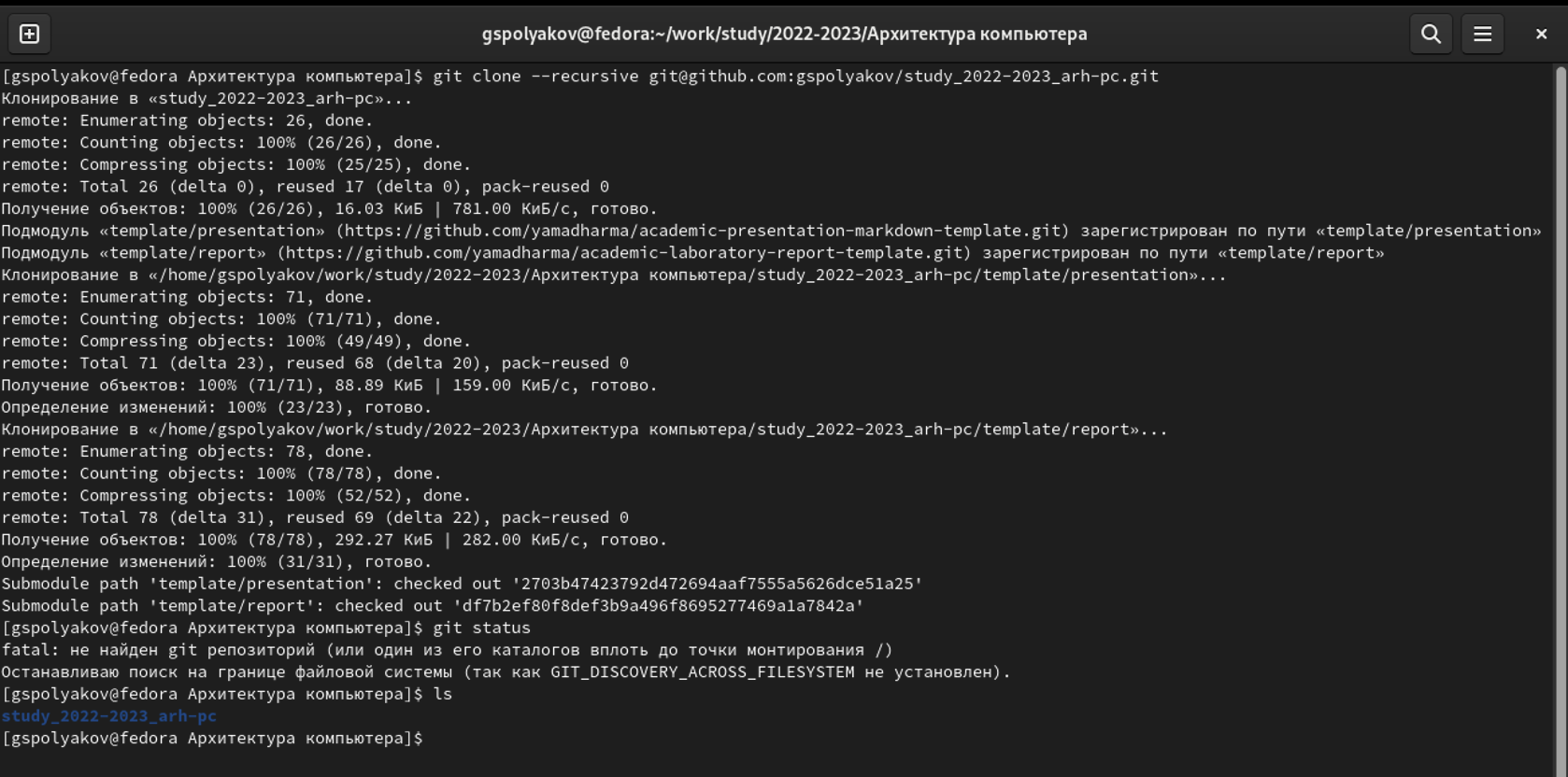


Рис. 5.2 Клонирование репозитория

1. Настройка каталога курса

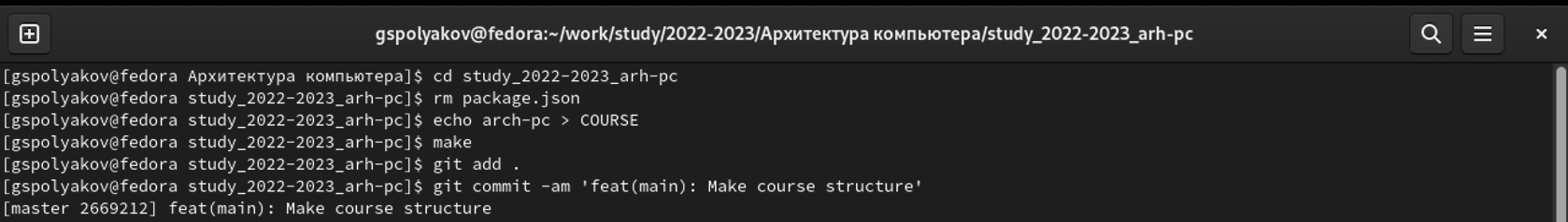


Рис. 6 Настройка каталога курса

**5 Выводы**

Выполняя лабораторную работу, я приобрел практические навыки работы с операционной системой на уровне командной строки (организация файловой системы, навигация по файловой системе, создание и удаление файлов и директорий).