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: Юсупова Алина Руслановна

Группа: НКАбд-06-25

МОСКВА

2025 г.

Содержание

1 Цель работы

Целью работы является изучение идеологии и применения средств контроля версий, приобретение практических навыков по работе с системой контроля версий git.

2 Задания

4.1 Техническое обеспечение

4.2 Базовая настройка Git

4.3 Создание SSH ключа

4.4 Создание рабочего пространства и репозитория курса на основе шаблона.

4.5 Создание репозитория курса на основе шаблона

4.6 Настройка каталога курса

5 Задания для самостоятельной работы

3 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 (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости 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.

4 Выполнение лабораторной работы

4.1 Техническое обеспечение

Лабораторная работа была выполнена на домашнем компьютере под управлением операционной системы Linux Mint 22.2 (рис.4.1.1).

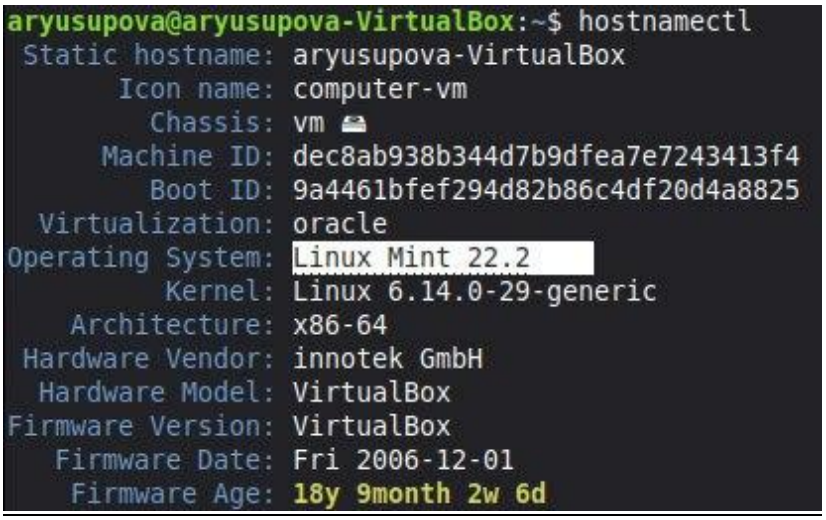

Рис.4.1.1. Операционная система.

4.2 Базовая настройка Git

Для начала зарегистрируюсь на GitHub (рис. 4.2.1).

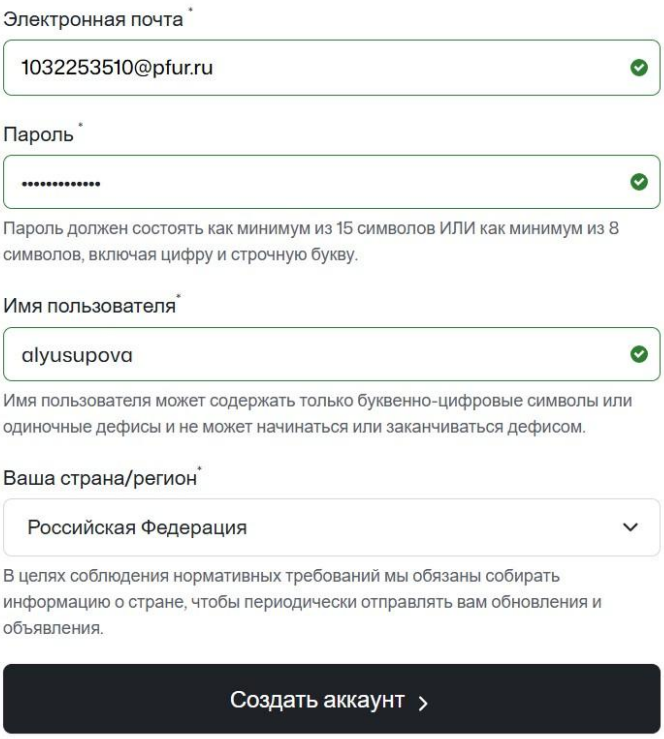

Рис.4.2.1. Регистрация на GitHub.

Затем необходимо установить ПО Git, используя команду sudo apt install git, т.к на моём компьютере его нет(рис.4.2.2).

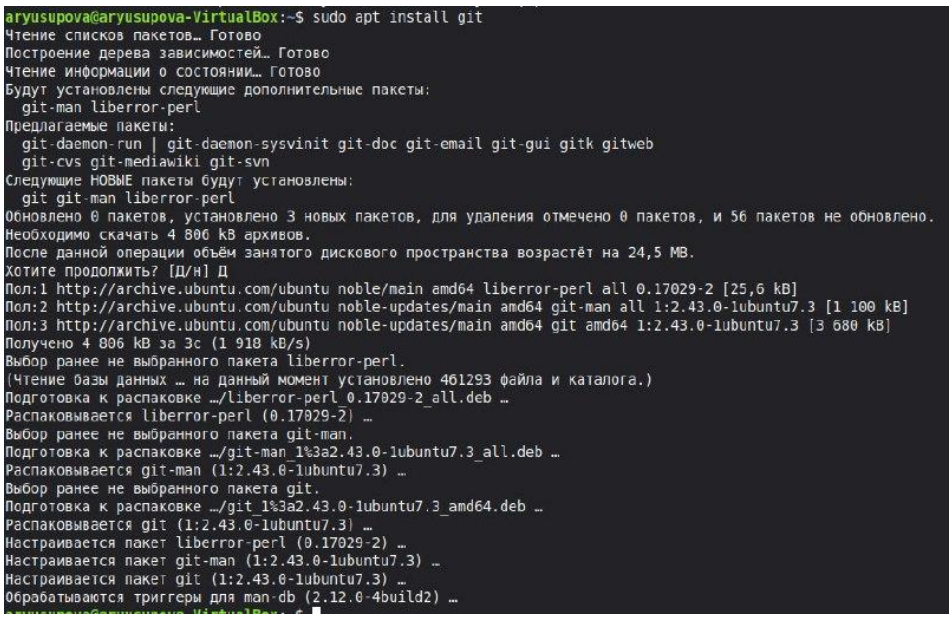

Рис.4.2.2. Установка ПО Git.

Далее я проведу предварительную конфигурацию Git, для этого открываю терминал и ввожу команды, указав имя и e-mail владельца репозитория (рис. 4.2.3).

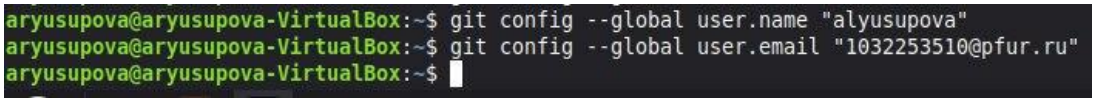

Рис.4.2.3. Предварительная конфигурация Git.

Также настраиваю параметры utf-8, имя начальной ветки, autocrlf и safecrlf(рис. 4.2.4).

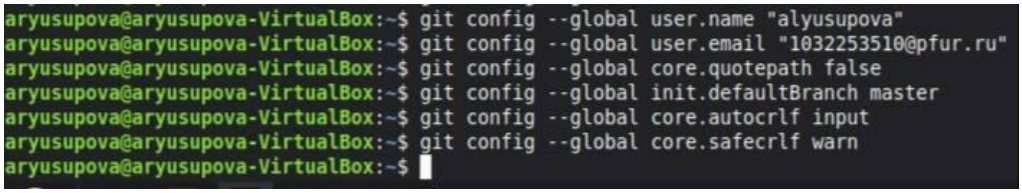

Рис.4.2.4. Настройка параметров Git.

4.3 Создание SSH ключа

После этого генерирую пару ключей, они сохраняются в каталоге ~/.ssh/. и понадобится для последующей идентификации пользователя на сервере (рис.4.3.1).

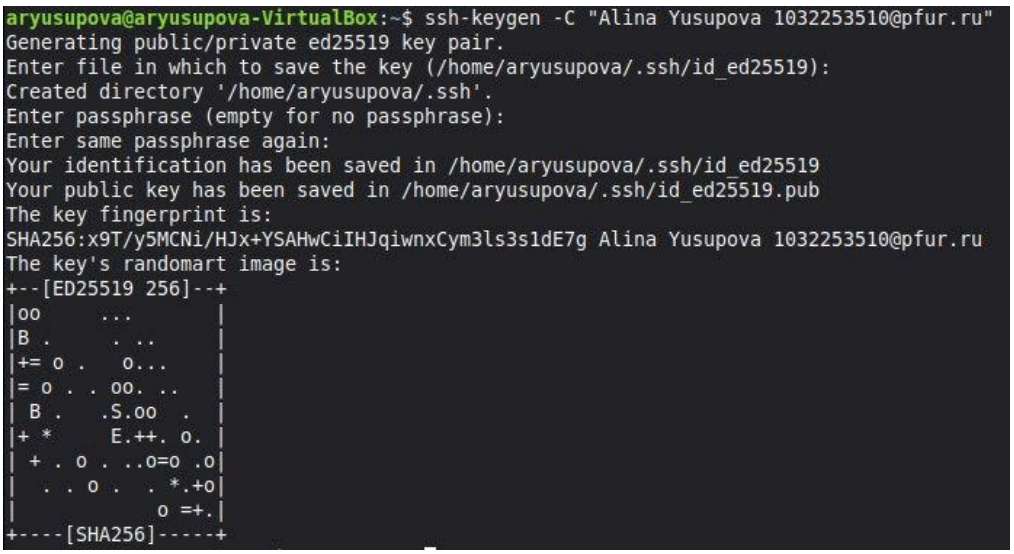

Рис.4.3.1. Генерация ключей.

Загружу ключ, который сгенерировала. Для этого копирую ключ из локальной консоли в буфер обмена (рис. 4.3.2). Затем захожу на сайт https://github.org/ под своей учётной записью и во вкладке настройки, выбираю “SSH и GPG ключи”, далее - “Новый SSH ключ”. В поле “Имя ключа” указываю “Title”, а в поле “Ключ” вставляю скопированный ключ (рис. 4.3.3). Удостоверимся, что Ключ появился в github (рис. 4.3.4).

Рис. 4.3.2. Копирование сгенерированного ключа.

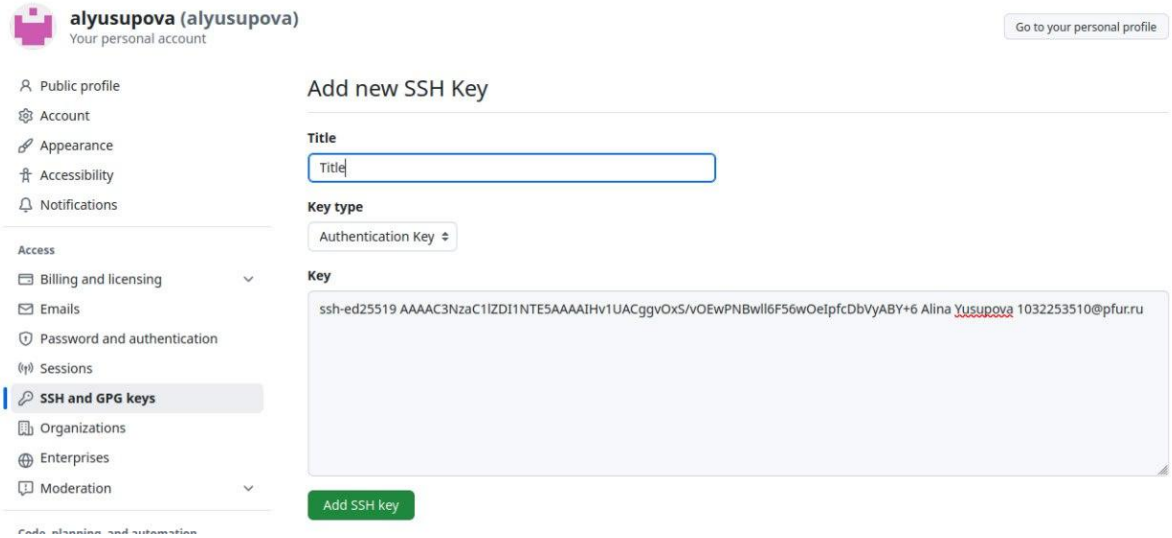

Рис.4.3.3. Загрузка SSH ключа на GitHub.

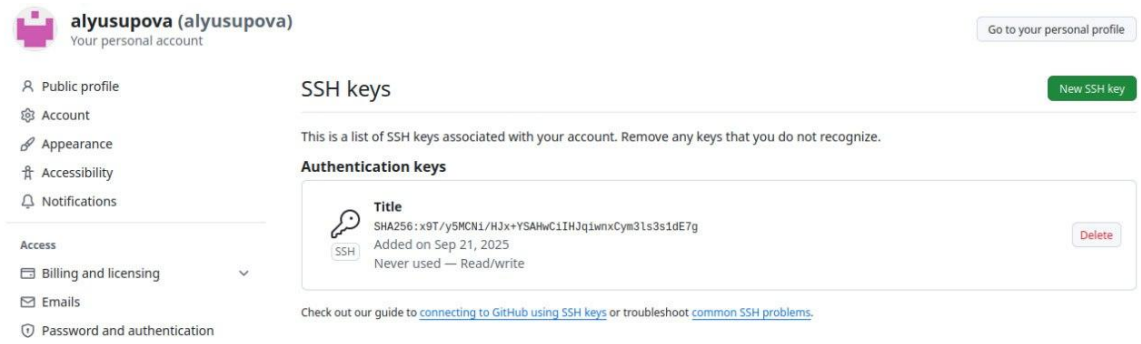

Рис.4.3.4. Сохранённый ключ в GitHub.

4.4 Создание рабочего пространства и репозитория курса на основе шаблона.

Затем я открыла терминал и создал каталог для предмета “Архитектура компьютера”, придерживаясь структуры рабочего пространства, т.е чтобы оно удовлетворяло следующей иерархии: ~/work/study// (рис.4.4.1).

Рис.4.4.1. Создание рабочего пространства (каталога).

4.5. Создание репозитория курса на основе шаблона.

Перехожу на страницу репозитория с шаблоном курса https://github.com/yamadharma/course-directory-student-template. Выбираю “Выбрать этот шаблон”, из падующего списка нажимаю на “Создать новый репозиторий”(рис.4.5.1), указываю имя (study_2025-2026_arh-pc) и создаю репозиторий (рис.4.5.2).

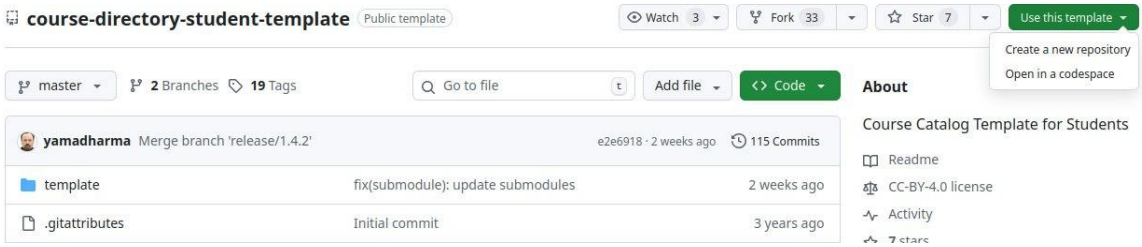

Рис. 4.5.1. Создание репозитория по шаблону.

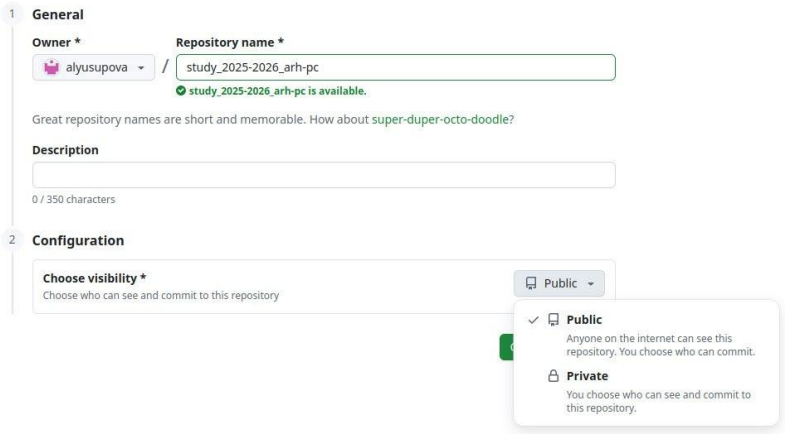

Рис.4.5.2. Создание репозитория по шаблону.

Сгенерированный репозиторий на основе шаблона клонирую на свой рабочий компьютер, для этого беру ссылку для клонирования через интерфейс GitHub (рис. 4.5.3) ,затем ввожу в терминале git clone и сразу проверяю успешность операции с помощью команды ls. . (рис 4.5.4).

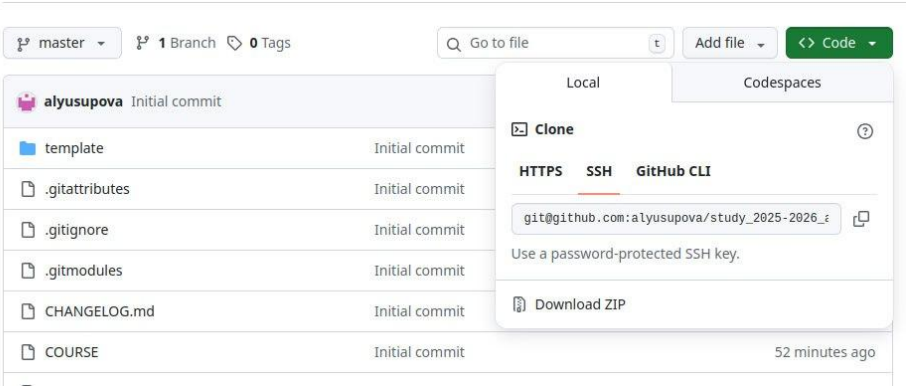

Рис. 4.5.3. Копирование ссылки для клонирования.

```
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера$ git clone --recursive git@github.com:alyusupova/study_2025-2026_arh-pc.git
Клонирование в «study_2025-2026_arh-pc»...
The authenticity of host 'github.com (140.82.121.3)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 38 (delta 1), reused 27 (delta 1), pack-reused 0 (from 0)
Получение объектов: 100% (38/38), 23.45 КиБ | 1.12 МиБ/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/home/aryusupova/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc/template/presentation»...
remote: Enumerating objects: 161, done.
remote: Counting objects: 100% (161/161), done.
remote: Compressing objects: 100% (111/111), done.
remote: Total 161 (delta 60), reused 142 (delta 41), pack-reused 0 (from 0)
Получение объектов: 100% (161/161), 2.65 МиБ | 3.07 МиБ/с, готово.
Определение изменений: 100% (60/60), готово.
Клонирование в «/home/aryusupova/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc/template/report»...
remote: Enumerating objects: 221, done.
remote: Counting objects: 100% (221/221), done.
remote: Compressing objects: 100% (152/152), done.
remote: Total 221 (delta 98), reused 180 (delta 57), pack-reused 0 (from 0)
Получение объектов: 100% (221/221), 765.46 КиБ | 2.51 МиБ/с, готово.
Определение изменений: 100% (98/98), готово.
Submodule path 'template/presentation': checked out '6efd5c4ee78e4456caff3dc7062cfcad26058ca6'
Submodule path 'template/report': checked out '89a9622199b4df88227b9b3fa3d4714c85f68dd2'
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера$ ls
study_2025-2026_arh-pc
```

Рис.4.5.4. Клонирование репозитория и проверка.

4.6 Настройка каталога курса

Перехожу в каталог курса и удаляю лишние файлы (рис. 4.6.1). Также создаю необходимые каталоги (рис. 4.6.2).

```
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера$ cd study_2025-2026_arh-pc
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$ rm package.json
```

Рис.4.6.1. Переход в каталог и удаление ненужных файлов.

```
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$
 echo arch-pc > COURSE
```

Рис.4.6.2. Создание необходимых каталогов.

Отправляю файлы на сервер, делаю снимок сделанных изменений и push'у их на свой репозиторий в GitHub. (рис 4.6.3).

```
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$ git add .
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$ git commit -am 'feat(main): make course structure'
[master 044f54e] feat(main): make course structure
 2 files changed, 1 insertion(+), 14 deletions(-)
 delete mode 100644 package.json
```

Рис.4.6.3. Отправка изменений на удаленный репозиторий.

Затем проверю правильность создания иерархии рабочего стола в локальном репозитории (рис. 4.6.4) и на странице github (рис. 4.6.5).

Рис.4.6.4. Проверка правильности создания иерархии рабочего пространства.

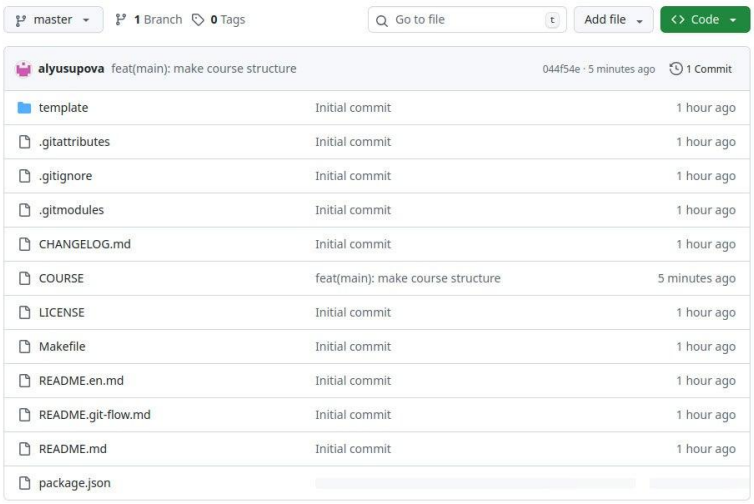

Рис.4.6.5. Проверка репозитория на github.com.

5 Задания для самостоятельной работы.

Создаю в локальном репозитории файл отчета 2-ой лабораторной работы в соответствующей папке, также копирую отчет первой лабораторной работы в папку, предназначенную для него (рис. 5.1).

```
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/s
tudy_2025-2026_arh-pc$ touch labs/lab02/report/ЛО_Юсупова_отчёт.pdf
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/s
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$
cp /home/aryusupova/Загрузки/ЛО1_Юсупова_отчёт.pdf labs/lab01/report
```

Рис.5.1. Создание и копирование файлов отчётов в локальном репозитории.

Затем загружаю файлы на GitHub(рис. 5.2). Не забываю проверить наличие файлов на GitHub(рис.5.3, рис.5.4).

Проверка прошла успешно, файлы скопировались в GitHub.

```
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$
git add .
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$
git commit -am 'uploaded previous reports'
[master 285d32c] uploaded previous reports
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 labs/lab01/report/ЛО1_Юсупова_отчёт.pdf
 create mode 100644 labs/lab02/report/ЛО2_Юсупова_отчёт.pdf
aryusupova@aryusupova-VirtualBox:~/work/study/2025-2026/Архитектура компьютера/study_2025-2026_arh-pc$
git push
Перечисление объектов: 10, готово.
Подсчет объектов: 100% (10/10), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (7/7), готово.
Запись объектов: 100% (9/9), 2.31 МиБ | 3.07 МиБ/с, готово.
Всего 9 (изменений 1), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:alyusupova/study_2025-2026_arh-pc.git
   044f54e..285d32c  master -> master
```

Рис. 5.2. Загрузка файлов на GitHub.

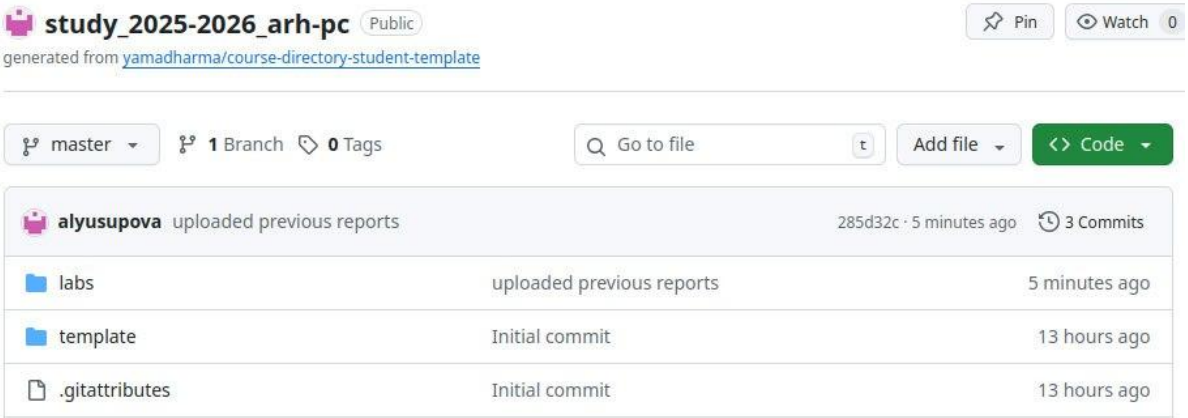

Рис.5.3. Проверка файлов на GitHub.

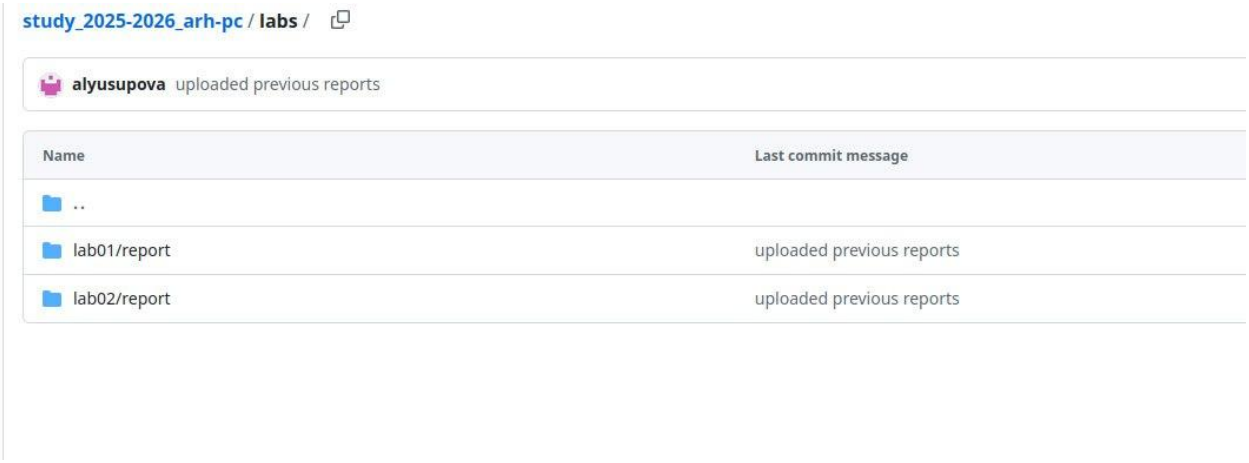

Рис.5.3. Проверка файлов на GitHub.

6 Выводы

При выполнении лабораторной работы я изучил идеологию и применение средств контроля версий. Также я приобрёл практические навыки по работе с системой git.

7 Ответы на вопросы для самопроверки

1)Системы контроля версий (VCS – Version Control Systems) – это программное обеспечение, которое отслеживает изменения в файлах (чаще всего исходном коде программ, но также и в других документах) с течением времени. Проще говоря, VCS позволяет сохранять "снимки" вашего проекта на разных этапах его разработки, создавая историю всех внесенных правок.

2)Хранилище (репозиторий) в системе контроля версий (VCS) — это система, которая обеспечивает хранение всех существовавших версий файлов. Часто говорят об удалённом репозитории (копия кода на каком-то сервере) и локальном репозитории (копия на компьютере разработчика).

Commit — это запись изменений. С помощью коммитов изменения, внесённые в рабочую копию, заносятся в хранилище.

История — это список предыдущих изменений. Благодаря истории можно отследить изменения, вносимые в репозиторий.

Рабочая копия — это копия файла, с которой непосредственно ведётся работа (находится вне репозитория). Перед началом работы рабочую копию можно получить из одной из версий, хранящихся в репозитории.

3) Централизованные СКВ (например, CVS, Subversion) имеют один общий репозиторий, в котором работают все пользователи. В отличие от них, распределённые СКВ (такие как Git, Bazaar, Mercurial) используют несколько репозиториев, позволяя обмениваться изменениями между ними, а центральный репозиторий может и вовсе не требоваться.

4) При единоличной работе с VCS, вы выполняете следующие действия:

Инициализация репозитория:

* Если проект новый: git init (создает локальное хранилище).
* Если проект уже существует: git init в корне проекта.

Создание/изменение файлов:

* Я работаю с файлами в вашей локальной папке проекта.

Отслеживание изменений:

* git add <файл>: добавить конкретный файл для следующего коммита.

* git add .: добавить все измененные и новые файлы в текущей директории. Сохранение изменений (Commit):

* git commit -m "Описание изменений": сохранить добавленные файлы как новую версию в локальном хранилище.

Просмотр истории:

* git log: посмотреть список всех коммитов.

Возврат к предыдущим версиям:

* git checkout <хэш_коммита>: переключиться на состояние проекта из указанного коммита.

* git checkout -- <файл>: откатить изменения в конкретном файле к состоянию последнего коммита.

Создание и управление ветками (опционально):

* git branch <название_ветки>: создать новую ветку.

* git checkout <название_ветки>: переключиться на другую ветку.

* git merge <название_ветки>: объединить изменения из другой ветки в текущую.

Синхронизация си удаленным репозиторием (если используется):

* git remote add origin <URL_репозитория>: подключить локальное хранилище к удаленному.

* git push origin <название_ветки>: отправить локальные коммиты в удаленное хранилище.

* git pull origin <название_ветки>: загрузить изменения из удаленного хранилища.

5) Порядок работы с общим хранилищем VCS (системой контроля версий) предполагает совместную разработку и требует соблюдения определенных правил и процедур для эффективного сотрудничества и предотвращения конфликтов. Вот основные шаги и рекомендации:

1. Настройка рабочей среды (единожды для каждого разработчика):

• Клонирование репозитория: получить локальную копию (репозиторий) проекта из общего (удалённого) хранилища.

* git clone <URL_общего_репозитория> (Git)

• Настройка: Установить VCS (например, Git), настроить параметры аутентификации (ключи SSH, токен доступа), сконфигурировать имя пользователя и email.

• Результат: Полная локальная копия репозитория на компьютере разработчика.

2. Получение последних изменений (регулярно):

• Обновление локального репозитория: Перед началом работы над новой задачей или внесением изменений необходимо получить самые свежие изменения из общего репозитория. Это предотвращает конфликты и гарантирует, что вы работаете с актуальной версией кода.

* git pull origin main (или git pull origin master, если основная ветка называется master) (Git)

• Разрешение конфликтов (при необходимости): Если при обновлении возникают конфликты (т.е., изменения, внесенные вами и другими разработчиками, затрагивают одни и те же строки кода), их необходимо разрешить вручную. Для этого нужно отредактировать конфликтные файлы, указав, какие изменения следует сохранить.

• Результат: Локальный репозиторий содержит все последние изменения, внесенные другими разработчиками.

3. Создание ветки для новой задачи (рекомендуется):

• Изоляция изменений: Создание отдельной ветки для каждой новой функции, исправления ошибки или эксперимента позволяет изолировать изменения и избежать влияния на основную кодовую базу.

* git branch <имя_ветки> (создать новую ветку) (Git)

* git checkout <имя_ветки> (переключиться на новую ветку) (Git)

* git checkout -b <имя_ветки> (создать и переключиться на ветку одновременно) (Git)

• Выбор имени ветки: Имя ветки должно быть информативным и отражать суть задачи (например, feature/add-user-authentication, bugfix/resolve-login-issue).

• Результат: Создана новая ветка, в которой можно безопасно вносить изменения, не затрагивая основную ветку (например, main или master).

4. Внесение изменений и коммиты:

• Регулярные коммиты: Вносите изменения небольшими порциями и делайте коммиты как можно чаще. Каждый коммит должен представлять собой логически завершенную часть работы.

* git add . (или git add <имя_файла>) (добавить измененные файлы в staging area) (Git).

* git commit -m "Описание изменений" (зафиксировать изменения с кратким описанием) (Git)

• Содержательные сообщения к коммитам: Описывайте, что именно было изменено и почему. Хорошее описание помогает другим разработчикам (и вам в будущем) понимать историю изменений.

• Результат: Локальный репозиторий содержит серию коммитов с описанием изменений, внесенных в рамках текущей задачи.

5. Отправка изменений в общий репозиторий (push):

• Публикация ветки: После завершения работы над задачей необходимо отправить изменения в общий репозиторий.

* git push origin <имя_ветки> (Git).

• Создание Pull Request (Merge Request): В большинстве случаев, после отправки ветки в общий репозиторий, необходимо создать Pull Request (PR) или Merge Request (MR). PR/MR – это запрос на включение вашей ветки в основную ветку проекта. PR/MR позволяет другим разработчикам просмотреть ваш код, оставить комментарии и предложить исправления.

6. Code Review (обзор кода):

• Review: Другие разработчики просматривают ваш код в PR/MR, проверяют его на наличие ошибок, соответствие стандартам кодирования и общее качество.

• Комментарии и исправления: На основе результатов обзора кода вносятся необходимые исправления и изменения.

• Итерации: Процесс Code Review может повторяться несколько раз.

7. Слияние ветки (Merge):

• После одобрения: После того, как код был одобрен и все необходимые исправления внесены, ветку можно слить (merge) с основной веткой.

• Выполнение слияния: Слияние обычно выполняется через веб-интерфейс

(GitHub, GitLab, Bitbucket) или с помощью командной строки.

• git merge (локально) или через UI: Происходит интеграция изменений из ветки в основную ветку.

8. Удаление ветки (после слияния):

• Очистка: После успешного слияния ветку, как правило, удаляют, чтобы не создавать путаницу.

* git branch -d <имя_ветки> (удалить локальную ветку) (Git)
* git push origin --delete <имя_ветки> (удалить удаленную ветку) (Git)

9. Регулярная синхронизация:

• Повторение процесса: Повторяйте шаги 2-8 для каждой новой задачи, чтобы поддерживать локальный репозиторий в актуальном состоянии и эффективно сотрудничать с другими разработчиками.

Важные принципы при работе с общим репозиторием:

• Коммуникация: Обсуждайте изменения и планируйте работу с другими участниками команды.

• Code Review: Всегда проводите Code Review для повышения качества кода и обмена знаниями.

• Тестирование: Пишите и запускайте тесты для проверки работоспособности кода.

• Соглашения: Соблюдайте принятые в команде соглашения о стиле кодирования, именовании веток и коммитов.

• Разрешение конфликтов: Будьте готовы к разрешению конфликтов и делайте это оперативно.

Соблюдение этих правил и процедур поможет обеспечить эффективную совместную разработку, минимизировать конфликты и поддерживать высокое качество кода в общем репозитории VCS.

6) Централизованные СКВ (например, CVS, Subversion) имеют один общий репозиторий, в котором работают все пользователи. В отличие от них, распределённые СКВ (такие как Git, Bazaar, Mercurial) используют несколько репозиториев, позволяя обмениваться изменениями между ними, а центральный репозиторий может и вовсе не требоваться.

7) git init: Инициализирует новый Git-репозиторий в текущей директории.

git clone <URL>: Создает локальную копию удаленного репозитория.

git add <файл>: Добавляет изменения в файле в индекс для следующего коммита.

git commit -m "<сообщение>": Сохраняет изменения из индекса в локальный репозиторий с описанием.

git status: Показывает текущее состояние рабочей директории (измененные, новые, добавленные файлы).

git log: Отображает историю коммитов.

git diff: Показывает различия между рабочей директорией и индексом, или между индексами и коммитами.

git checkout <ветка>: Переключает рабочую директорию на указанную ветку или коммит.

git branch: Управляет ветками (создает, удаляет, перечисляет).

git merge <ветка>: Объединяет изменения из указанной ветки в текущую.

git pull: Загружает изменения из удаленного репозитория и объединяет их с текущей веткой.

git push: Отправляет локальные коммиты в удаленный репозиторий.

git remote: Управляет удаленными репозиториями.

git reset: Отменяет коммиты или возвращает файлы в определенное состояние.

git revert: Создает новый коммит, отменяющий изменения предыдущего коммита.

git stash: Временно сохраняет незафиксированные изменения, чтобы очистить рабочую директорию.

8) Приведите примеры использования при работе с локальным и удалённым репозиториями.

Работа с локальным репозиторием:

1. Инициализация: git init (создать новый репозиторий).
2. Создание/изменение файлов: Работаете с файлами.
3. Статус: git status (увидеть, что изменилось).
4. Добавление: git add <файл> (подготовить к коммиту).

5. Коммит: git commit -m "Добавлена новая функция" (сохранить локально).
6. Просмотр истории: git log (увидеть коммиты).
7. Создание ветки: git checkout -b новая_фича (для новой фичи).
8. Слияние: git merge main (объединить ветку в main).

Работа с удаленным репозиторием (например, на GitHub):

1. Клонирование: git clone <URL_репозитория> (скопировать с сервера).
2. Pull: git pull origin main (получить последние изменения с сервера).
3. Push: git push origin main (отправить локальные коммиты на сервер).
4. Работа с ветками на удалёнке:

* git push origin <новая_ветка> (отправить локальную ветку на сервер).

* git checkout -b <ветка_с_сервера> origin/<ветка_с_сервера> (создать локальную ветку на основе удалённой).

Список литературы

1. https://esystem.rudn.ru/mod/page/view.php?id=1030492
2. https://esystem.rudn.ru/mod/resource/view.php?id=1030495
3. https://esystem.rudn.ru/mod/resource/view.php?id=1030496
4. https://github.com/evdvorkina/study_2022-2023_arh-pc/tree/master