Отчёт по лабораторной работе №2

Операционные системы

Луангсуваннавонг Сайпхачан

Содержание

Список иллюстраций

Список таблиц

# 1 Цель работы

Цель работы - Изучение идеологии и применения инструментов контроля версий и овладение навыками работы с git.

# 2 Задание

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git.
5. Зарегистрироваться на Github.
6. Создать локальный каталог для выполнения заданий по предмету.

# 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.

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. Кроме того, 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.

# 4 Выполнение лабораторной работы

## 4.1 Установка программного обеспечения

Я установил git и gh в систему используя команду dnf install(рис. 1)

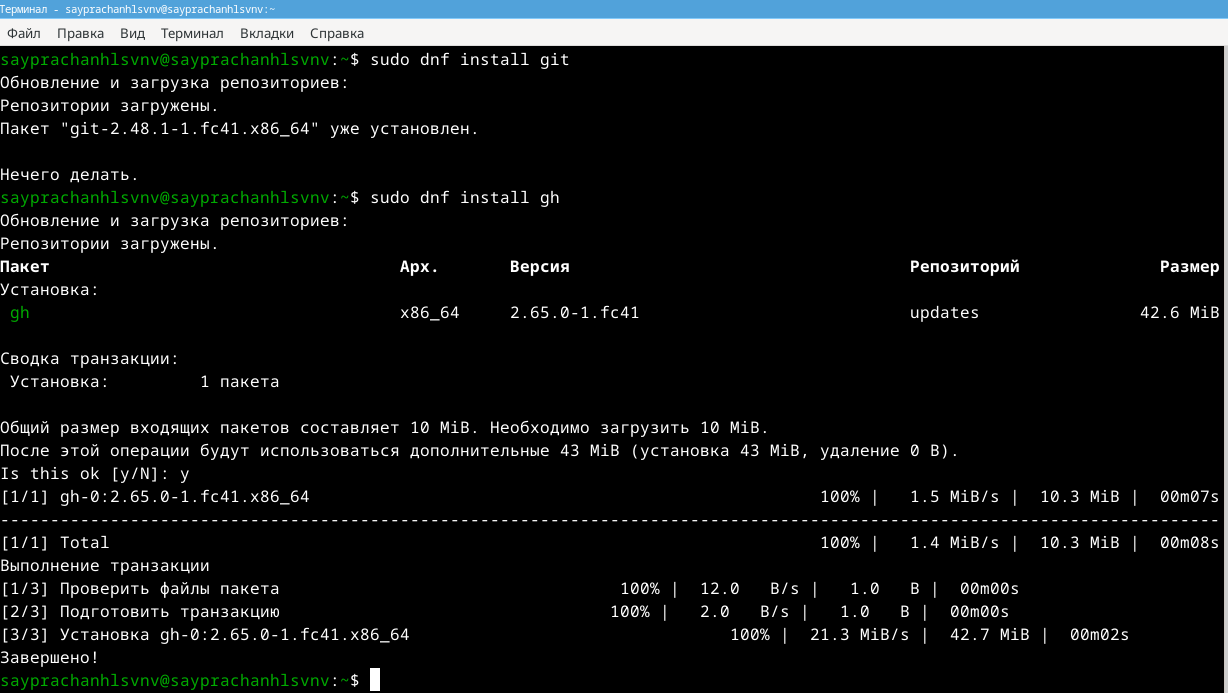


Рис. 1: Установка git и gh

## 4.2 Базовая настройка git

Затем я настраиваю имя и адрес электронной почты владельца репозитория, используя свое имя и адрес электронной почты (рис. 2)

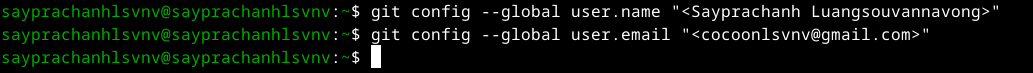


Рис. 2: Настройка имени и email владельца репозитория

Я настраиваю utf-8 при выводе git-сообщений о коррекции отображения (рис. 3)

Рис. 3: Настройка utf-8 при выводе сообщений git

Рис. 3: Настройка utf-8 при выводе сообщений git

Я задаю имя начальной ветви как (master) (рис. 4)

Рис. 4: Установка имени начальной ветви

Рис. 4: Установка имени начальной ветви

Я задаю параметры autocrlf и safecrlf для корректного отображения строки (рис. 5 и рис. 6)

Рис. 5: Настройка параметра autocrlf

Рис. 5: Настройка параметра autocrlf

Рис. 6: Настройка параметра safecrlf

Рис. 6: Настройка параметра safecrlf

## 4.3 Создание ключа SSH

Я создаю ssh-ключ размером 4096 бит по алгоритму rsa(рис. 7)

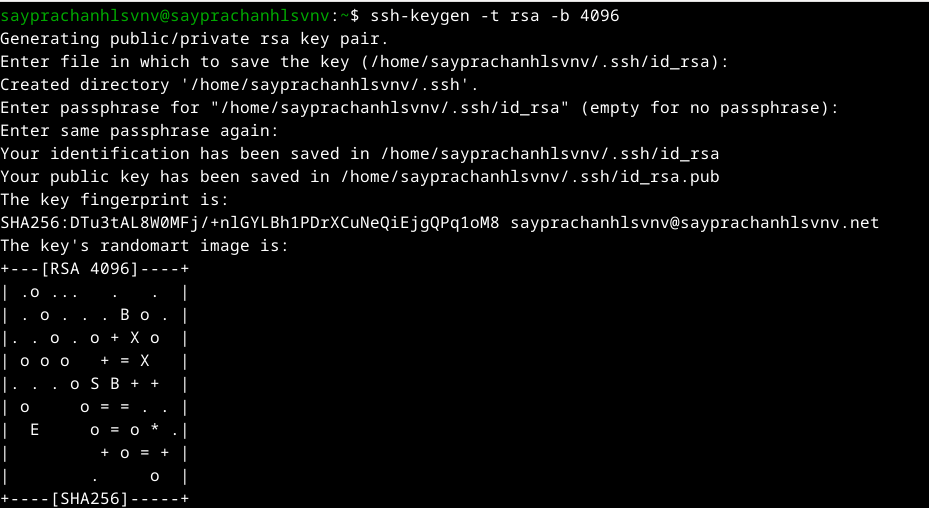


Рис. 7: создание ssh-ключа по алгоритму rsa

Я создаю ssh-ключ по алгоритму ed25519(рис. 8)

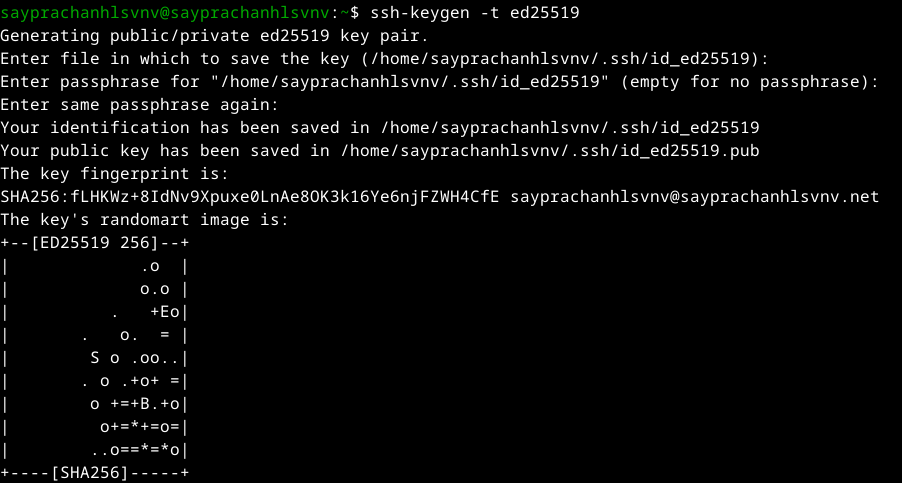


Рис. 8: создание ssh-ключа по алгоритму ed25519

## 4.4 Создание ключа GPG

Генерирую ключ GPG, выбираю 1 для типа ключа RSA и RSA , затем ввожу 4096 для необходимого размера ключа и оставляю неограниченный срок действия ключа, а также отвечаю на вопросы о личной информации(рис. 9)

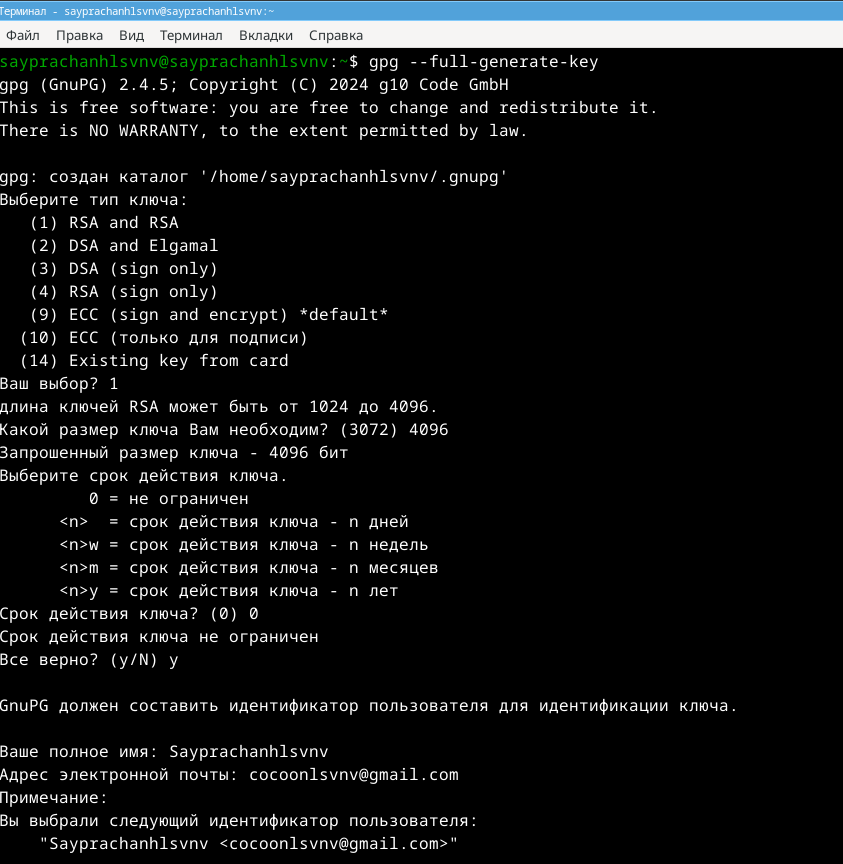


Рис. 9: Генерация ключа GPG

Я ввожу фразу-пароль для защиты нового ключа(рис. 10)

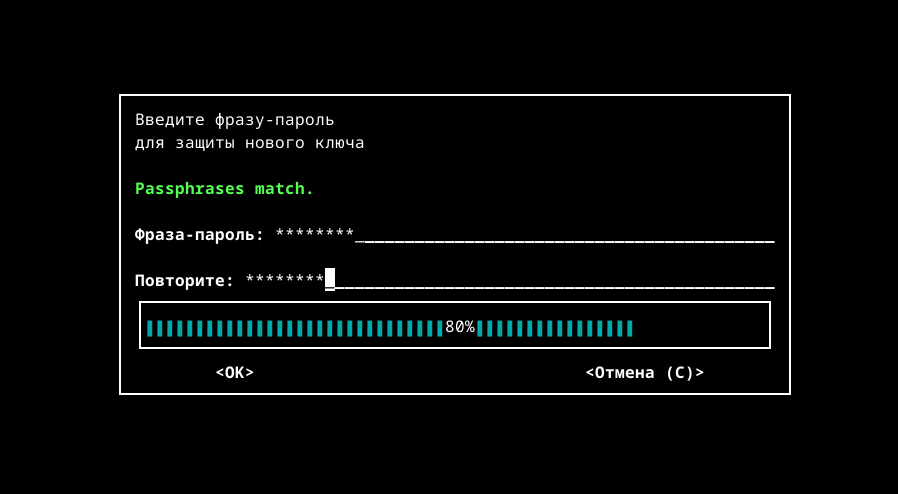


Рис. 10: Ввод фразу-пароля

## 4.5 Настройка GitHub

Я уже создал аккаунт на GitHub, а также настроил систему, поэтому просто вхожу в свой аккаунт.(рис. 11)

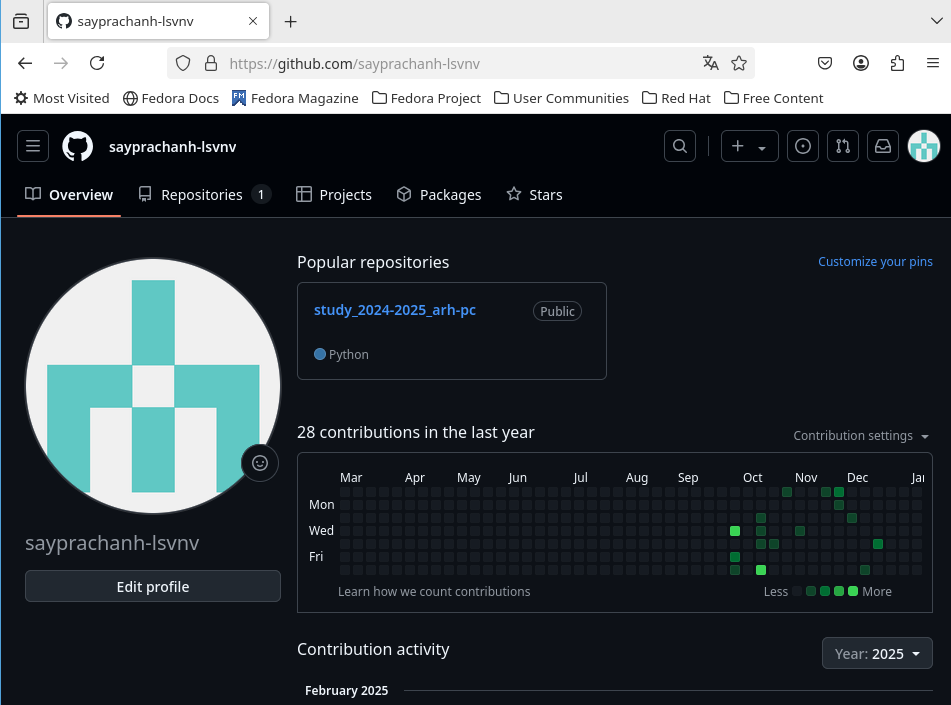


Рис. 11: Аккаунт на GitHub

## 4.6 Добавление GPG ключа в GitHub

Я отображаю список ключей в терминале и нахожу сгенерированные ключи GPG, ключ находится за обратной косой чертой, поэтому я копирую его(рис. 12)

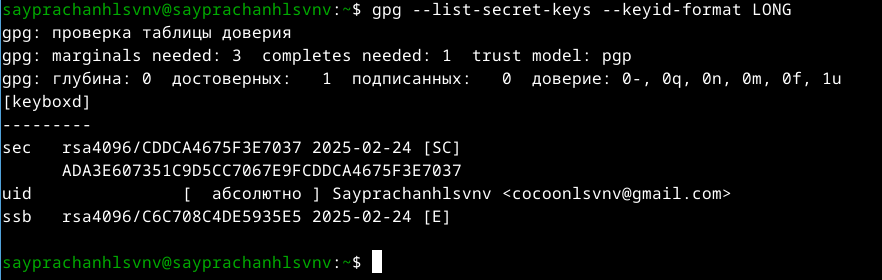


Рис. 12: Отображение списка ключей

Я самостоятельно копирую отпечаток GPG, затем с помощью утилиты xclip копирую его в буфер обмена(рис. 13)

Рис. 13: Копирование ключей GPG

Рис. 13: Копирование ключей GPG

Захожу в настройки свой аккаунт на GitHub, нахожу раздел GPG keys для ее добавления(рис. 14)

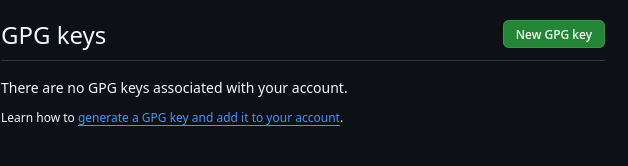


Рис. 14: Настройка GitHub

Нажимаю “New GPG Key” и вставляю ключ из буфера обмена(рис. 15)

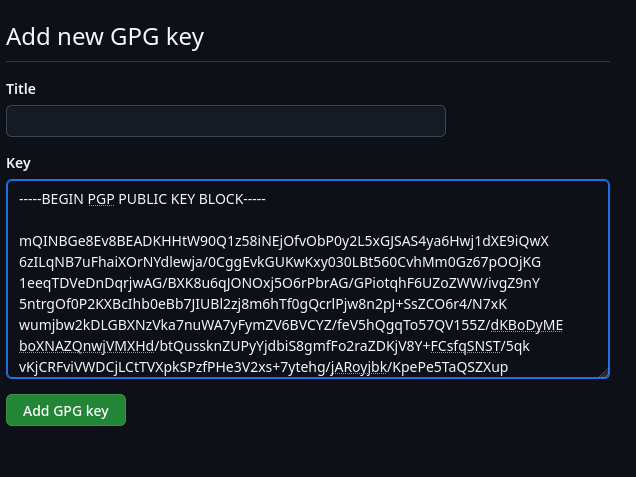


Рис. 15: Добавление ключа GPG

Я добавил ключи GPG(рис. 16)

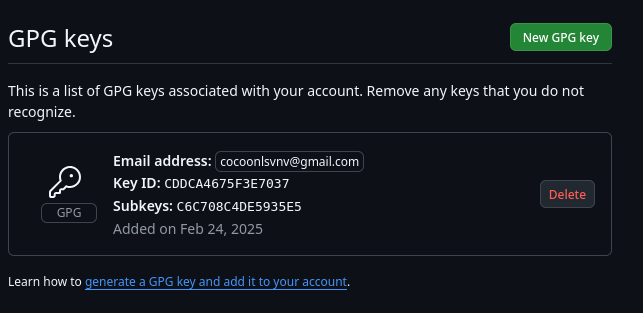


Рис. 16: Добавлен ключ GPG

## 4.7 Настройка автоматических подписей коммитов git

Используя ранее введенный адрес электронной почты, я указываю git, чтобы использовать его при создании подписи фиксации(рис. 17)

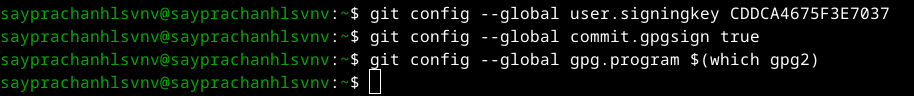


Рис. 17: Настройка подписи GPG

## 4.8 Настройка gh

Я начинаю авторизацию в gh, отвечаю на наводящие вопросы и выбираю Авторизоваться через браузер.(рис. 18)

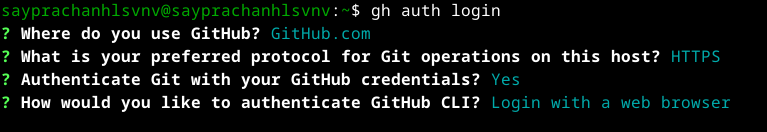


Рис. 18: Авторизация в gh

Затем я завершаю авторизацию на сайте(рис. 19)

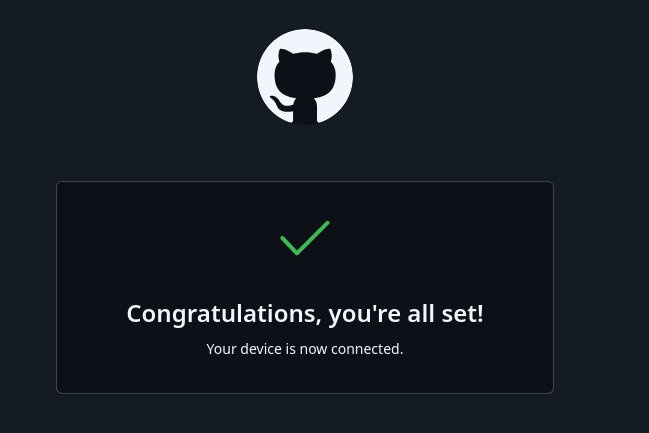


Рис. 19: Завершение авторизации через браузер

Вижу текст о завершении авторизации под именем sayprachanh-lsvnv(рис. 20)

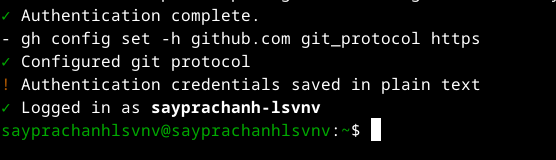


Рис. 20: Завершение авторизации

## 4.9 Сознание репозитория курса на основе шаблона

После этого я начинаю создавать каталог с помощью mkdir, присваивая ему имя: операционные системы. Используя cd, я вхожу в созданный каталог. А также создайте новый репозиторий на моем Github под названием study\_2024-2025\_os-intro(рис. 21)

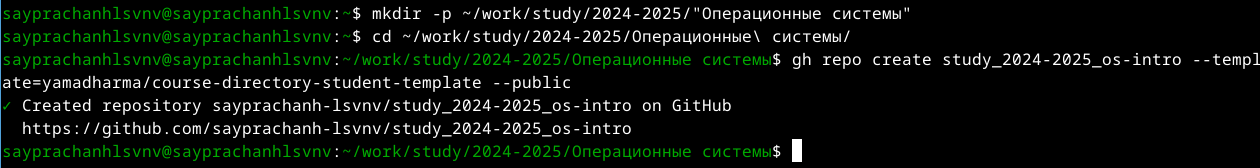


Рис. 21: Создание нового каталога и репозитория

Используя git clone, я клонирую удаленный репозиторий, который я создал ранее, в свой локальный репозиторий(рис. 22)

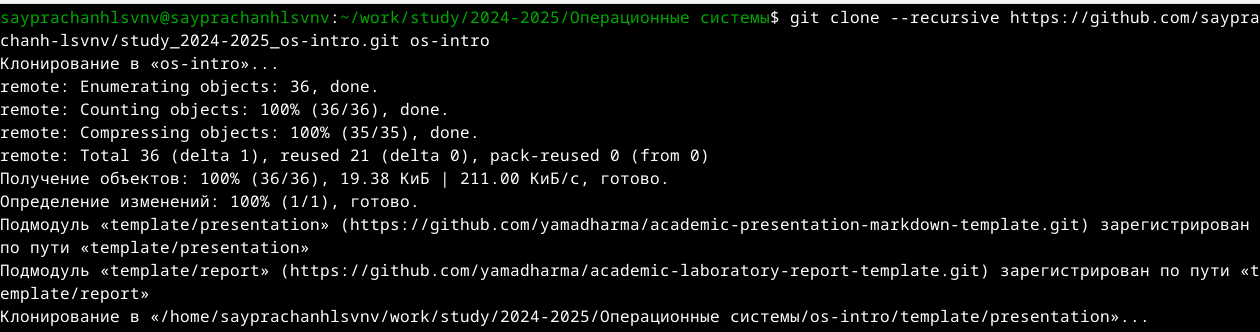


Рис. 22: Клонирование репозитория

## 4.10 Настройка каталога курса

Я перехожу в каталог с помощью command cd и проверяю содержимое каталога с помощью ls(рис. 23)

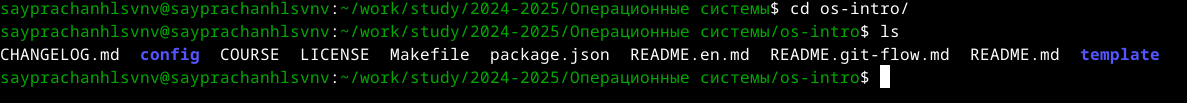


Рис. 23: Перемещение между каталогами

Используя команду rm, я удаляю файл package.json. Затем я создаю необходимые каталоги с помощью make и выбираю опцию “prepare” для создания структуры каталогов(рис. 24)

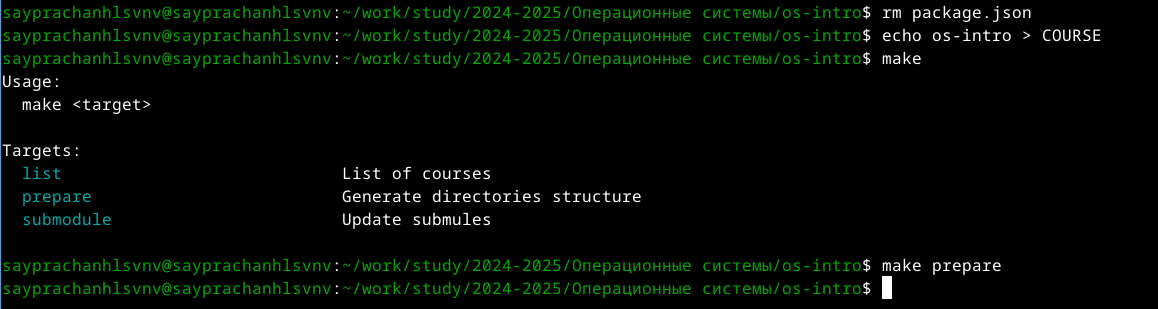


Рис. 24: Удаление файлов и создание каталогов

Я добавляю все файлы для отправки на сервер с помощью команды git add. и комментирую их с помощью git commit(рис. 25)

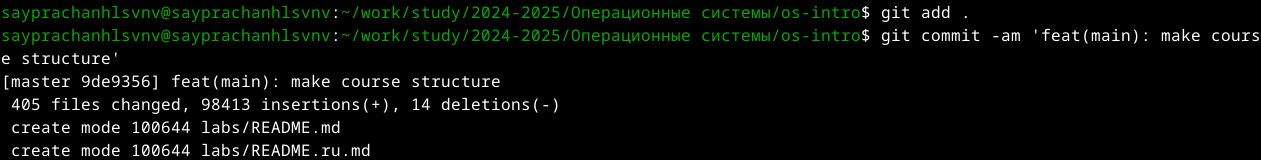


Рис. 25: Добавление файлов и комментирование к ним

Я отправляю файл на сервер с помощью команды git push(рис. 26)

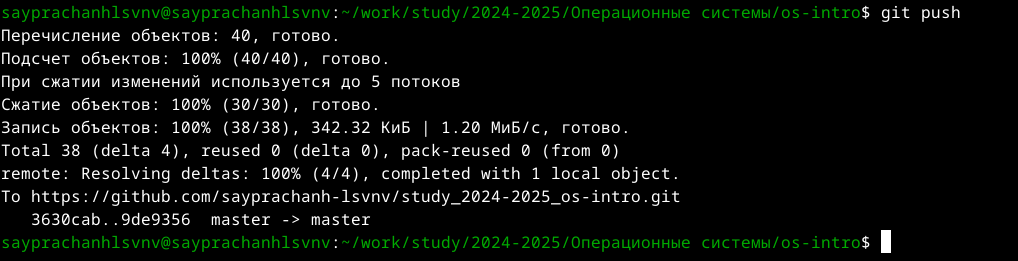


Рис. 26: Отправка файлов на сервер

# 5 Выводы

Во время выполнения этой лабораторной работы, я изучил идеологию и применяемые инструменты контроля версий, а также овладел навыками работы с git

# 6 Ответы на контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

Системы контроля версий (VCS) — это инструменты для управления изменениями в файлах. Они помогают:

Сохранять историю изменений. Отслеживать, кто и когда внёс изменения. Возвращаться к предыдущим версиям. Совместно работать над проектами.

1. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище (репозиторий) — место, где хранятся все версии файлов.

Commit — фиксация изменений в хранилище с комментарием. История — последовательность всех commit’ов. Рабочая копия — текущая версия файлов на компьютере разработчика.

1. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

Централизованные VCS (например, SVN): одно главное хранилище, с которым работают все. Децентрализованные VCS (например, Git): у каждого разработчика своя копия хранилища, изменения объединяются позже.

1. Опишите действия с VCS при единоличной работе с хранилищем.

Создать репозиторий (git init). Добавить файлы (git add). Фиксировать изменения (git commit). Просматривать историю (git log).

1. Опишите порядок работы с общим хранилищем VCS.

Клонировать репозиторий (git clone). Создать ветку для изменений (git branch). Фиксировать изменения (git commit). Отправить изменения на сервер (git push). Получить изменения от других (git pull).

1. Каковы основные задачи, решаемые инструментальным средством git?

Управление версиями файлов. Совместная работа над кодом. Создание и управление ветками. Резервное копирование и восстановление.

1. Назовите и дайте краткую характеристику командам git.

git init — создать новый репозиторий. git add — добавить файлы в индекс. git commit — зафиксировать изменения. git push — отправить изменения на сервер. git pull — получить изменения с сервера. git branch — управление ветками. git merge — объединить ветки.

1. Приведите примеры использования при работе с локальным и удалённым репозиториями.

Чтобы работать с локальным репозиторием, сначала создаём его с помощью команды git init. Затем добавляем файлы командой git add и фиксируем изменения командой git commit -m “Описание изменений”. Для работы с удалённым репозиторием клонируем его командой git clone, вносим изменения и отправляем их на сервер командой git push origin main.

1. Что такое и зачем могут быть нужны ветви (branches)?

Ветви — это отдельные линии разработки. Они нужны для: Параллельной работы над разными задачами. Изоляции экспериментальных изменений. Упрощения слияния изменений.

1. Как и зачем можно игнорировать некоторые файлы при commit?

Во время работы над проектом могут быть созданы файлы, которые не следует добавлять в репозиторий, например, временный файл. Чтобы игнорировать это, мы создаем файл.gitignore и указываем в нем шаблоны файлов или папок.

# Список литературы

[Лабораторная работа № 2](https://esystem.rudn.ru/mod/page/view.php?id=1224371)