Лабораторная работа №2

Операционные системы

Кучмар София Игоревна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

В рамках данной лабораторной работы мы выполним установку необходимого программного обеспечения, а именно установим Git и GitHub CLI (gh).

2 Задание

В процессе выполнения лабораторной работы мы произведем базовую настройку Git, сгенерируем SSH и PGP ключи для безопасной аутентификации. После этого мы настроим GitHub, добавив сгенерированный PGP ключ в нашу учетную запись и настроив автоматические подписи коммитов Git. Затем мы произведем настройку GitHub CLI (gh) для удобной работы с репозиториями. В завершение, мы создадим шаблон для рабочего пространства, создадим репозиторий курса на основе этого шаблона и настроим каталог курса для организации файлов. В итоге мы получим полностью настроенное окружение для контроля версий и совместной работы с кодом на платформе GitHub.

3 Выполнение лабораторной работы

Проведём установку программного обеспечения. Установим Git (рис. 3.1).

Рис. 3.1: Установим Git

Установим gh (рис. 3.2).

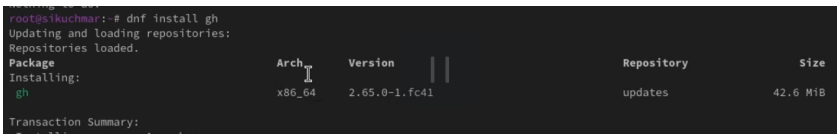

Рис. 3.2: Установим gh

Зададим имя и email владельца репозитория. Настроим utf-8 в выводе сообщений git. Зададим имя начальной ветки (будем называть её master). Установим параметр autocrlf и safecrlf (рис. 3.3).

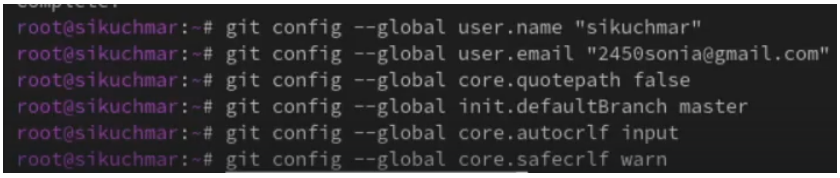

Рис. 3.3: Базовая настройка git

Сгенерируем ключи по алгоритму RSA с ключом размером 4096 бит (рис. 3.4).

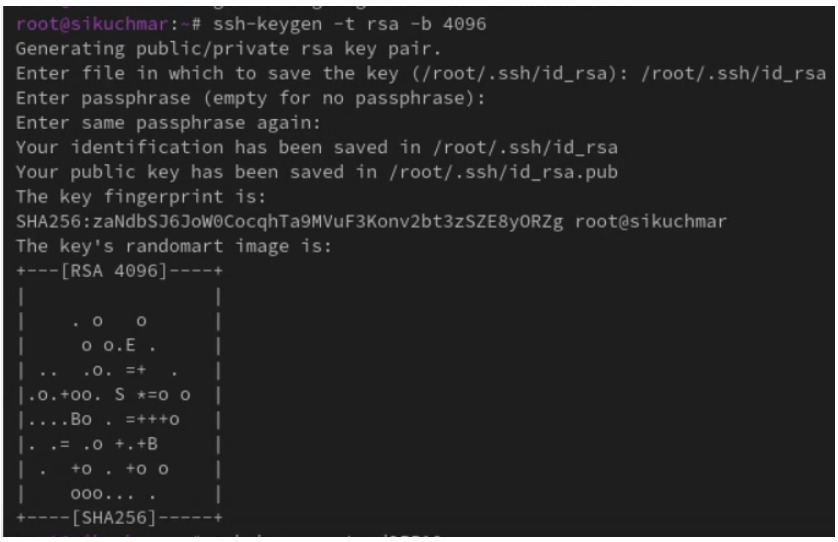

Рис. 3.4: Сгенерируем ключи по алгоритму RSA

Сгенерируем ключи по алгоритму ed25519 (рис. 3.5).

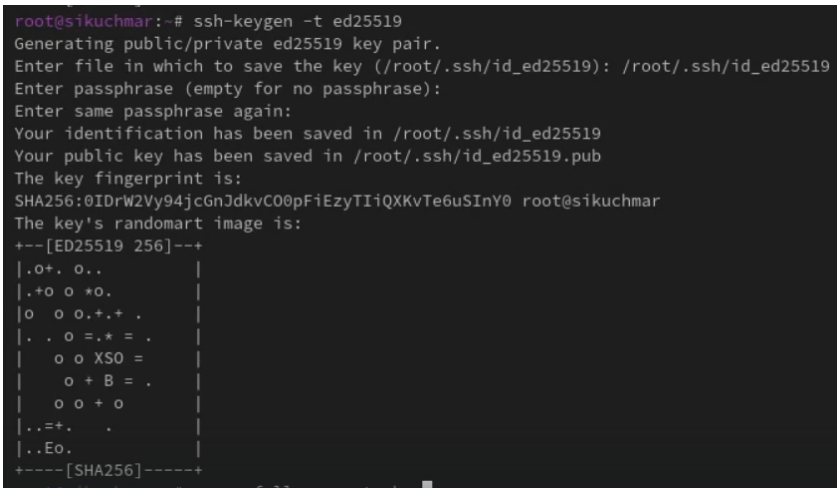

Рис. 3.5: Сгенерируем ключи по алгоритму ed25519

Сгенерируем PGP ключ (рис. 3.6).

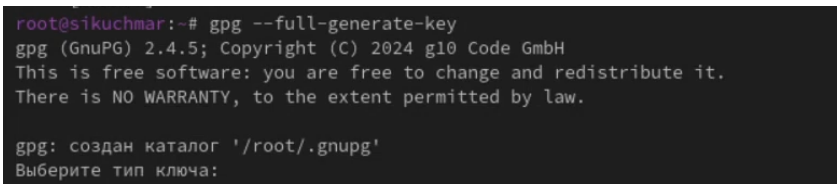

Рис. 3.6: Сгенерируем PGP ключ

Добавим его в наш github (рис. 3.7).

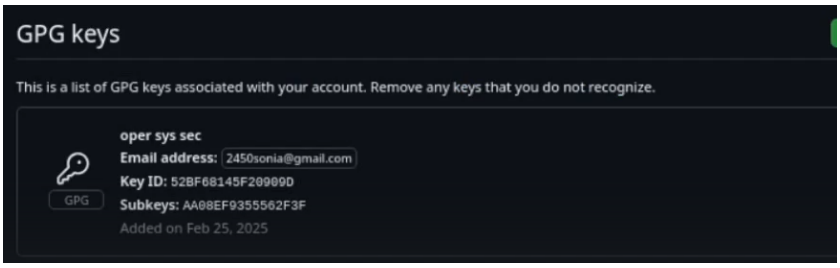

Рис. 3.7: Добавим его в наш github

Настройка автоматических подписей коммитов Git. Используя введённый email, укажем Git применять его при подписи коммитов (рис. 3.8).

Рис. 3.8: Настройка автоматических подписей коммитов Git

Для начала авторизуемся (рис. 3.9).

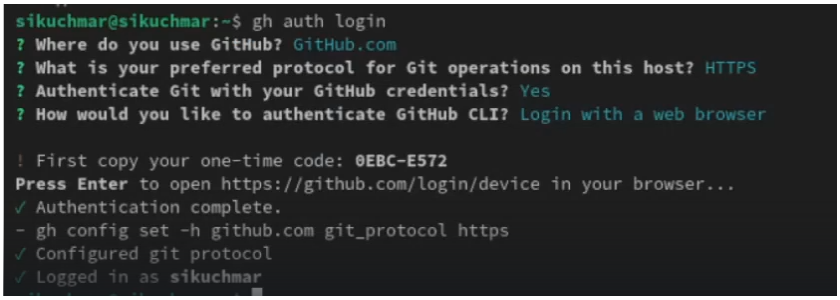

Рис. 3.9: Авторизация

Необходимо создать шаблон рабочего пространства. Для этого создадим папку (рис. 3.10).

Рис. 3.10: Создадим папку

Перейдём туда и создадим репозиторий (рис. 3.11).

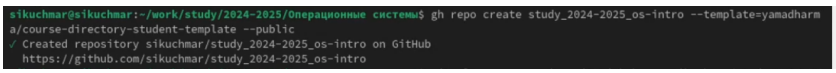

Рис. 3.11: Создадим репозиторий

После склонируем репозиторий (рис. 3.12).

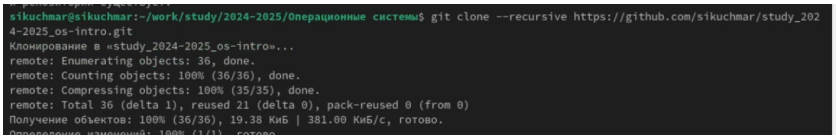

Рис. 3.12: Склонируем репозиторий

Перейдем в каталог курса. Удалим лишние файлы и создадим необходимые каталоги (рис. 3.13).

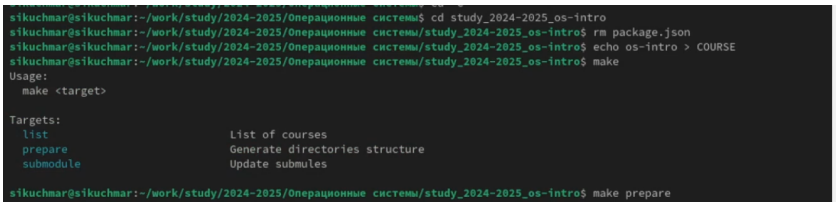

Рис. 3.13: Настройка каталога курса

Отправим файлы на сервер (рис. 3.14) и (рис. 3.15).

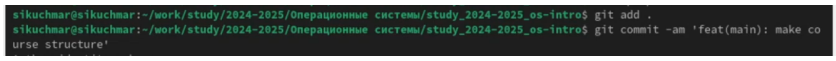

Рис. 3.14: git add и git commit

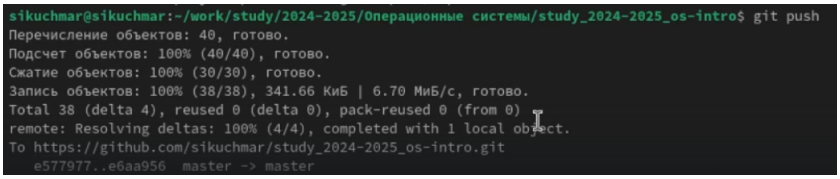

Рис. 3.15: git push

4 Контрольные вопросы

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

- VCS: Инструменты для отслеживания изменений файлов во времени.
- Задачи: Отслеживание, возврат, сравнение версий, совместная работа, ветвление, резервное копирование.

Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

- Хранилище: Все версии проекта + история.
- Commit: Снимок состояния проекта в момент времени.
- История: Последовательность коммитов.
- Рабочая копия: Локальная копия для работы.

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

- Централизованные: Единое хранилище (SVN, CVS).
- Децентрализованные: Полная копия хранилища у каждого (Git, Mercurial).

Опишите действия с VCS при единоличной работе с хранилищем.

1. Создать/Клонировать.
2. Изменить.

3. Проверить.
4. Добавить в индекс.
5. Закоммитить.
6. (Push для удаленного).

Опишите порядок работы с общим хранилищем VCS.

1. Клонировать.
2. Создать ветку.
3. Изменить.
4. Проверить.
5. Добавить в индекс.
6. Закоммитить.
7. Push.
8. Pull Request.
9. Ревью.
10. Слияние.
11. Удалить ветку.
12. Pull.

Каковы основные задачи, решаемые инструментальным средством git?

- Версионный контроль, совместная работа, управление ветками, отслеживание изменений, разрешение конфликтов, возврат к версиям.

Назовите и дайте краткую характеристику командам git.

- git init: Создать репо локально.
- git clone: Скачать репо удаленно.
- git add: Добавить в индекс сейчас.
- git commit: Зафиксировать изменения локально.
- git push: Отправить на сервер вверх.

- git pull: Получить с сервера вниз.
- git status: Состояние сейчас показывает.
- git log: История коммитов отображается.
- git branch: Ветки создать, переключить.
- git checkout: Переключиться между ветками.
- git merge: Объединить ветки вместе.
- git diff: Различия файлы покажет.

Приведите примеры использования при работе с локальным и удалённым репозиториями

- (Локально): git init, git add file.txt, git commit -m “Initial”, git log.
- (Удаленно): git clone URL, git remote add origin URL, git push origin main, git pull origin main.

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

- Независимые линии разработки.
- Нужны для: параллельной работы, экспериментов, разделения задач.

Как и зачем можно игнорировать некоторые файлы при commit?

- Исключение из коммитов временных, конфиденциальных и больших файлов.
- Сокращение размера репозитория и предотвращение конфликтов.

5 Выводы

В ходе выполнения лабораторной работы было успешно создано и настроено окружение для эффективной работы с системой контроля версий Git и платформой GitHub. В частности, были выполнены следующие задачи:

- Установлены необходимые инструменты: Git и GitHub CLI (gh).
- Выполнена базовая настройка Git и сгенерированы ключи SSH и PGP для безопасной аутентификации.
- Настроен аккаунт GitHub с добавлением PGP-ключа и включены автоматические подписи коммитов.
- Настроен GitHub CLI для удобного взаимодействия с репозиториями.
- Создан шаблон рабочего пространства, создан на его основе репозиторий курса и настроена структура каталогов для организации файлов.

В результате получено полностью готовое к использованию окружение для контроля версий и совместной разработки кода на платформе GitHub.