**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**

*дисциплина: Архитектура компьютеров “Операционные системы”*

Студент: Оганнисян Г.А.

Группа: НБИбд-03-24

№ ст. билета: 1132243806

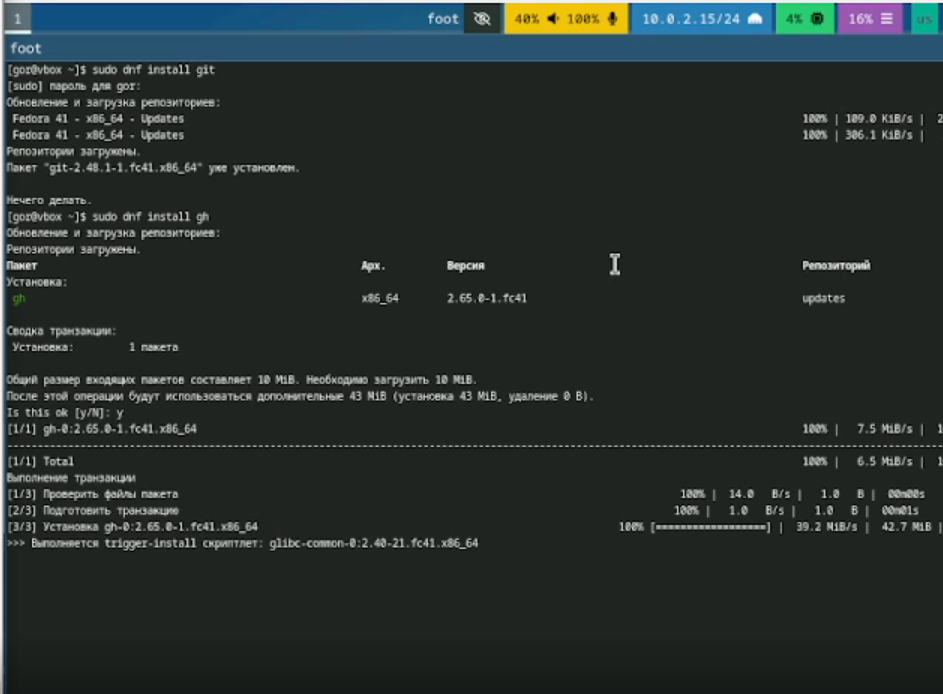
**МОСКВА**

2025 г.

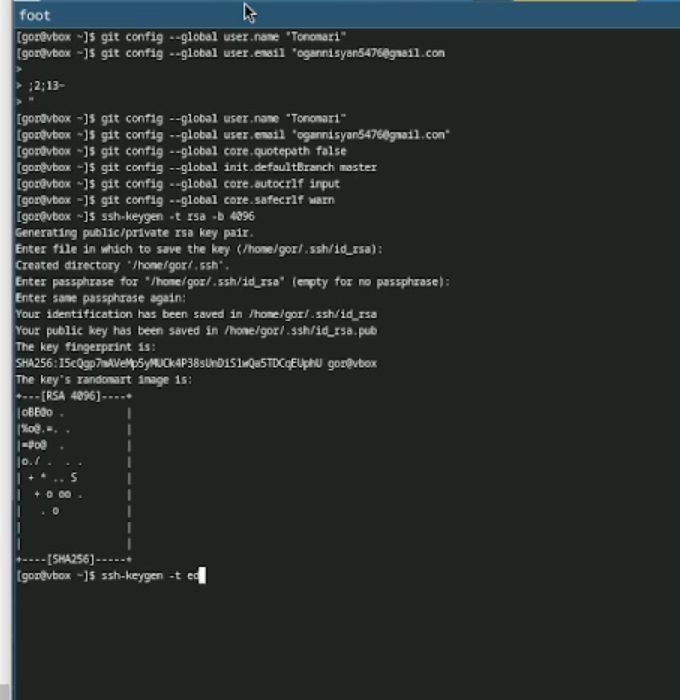
Цель работы

Целью данной работы является изучение идеологии git, применение средств контроля версий и освоение умений работы с git.

Описание результатов выполнения работы



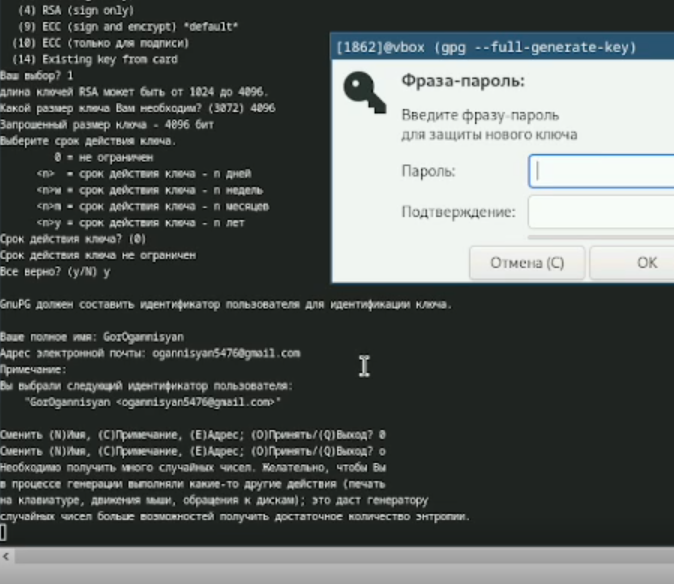
Устанавливаем Git и gh.



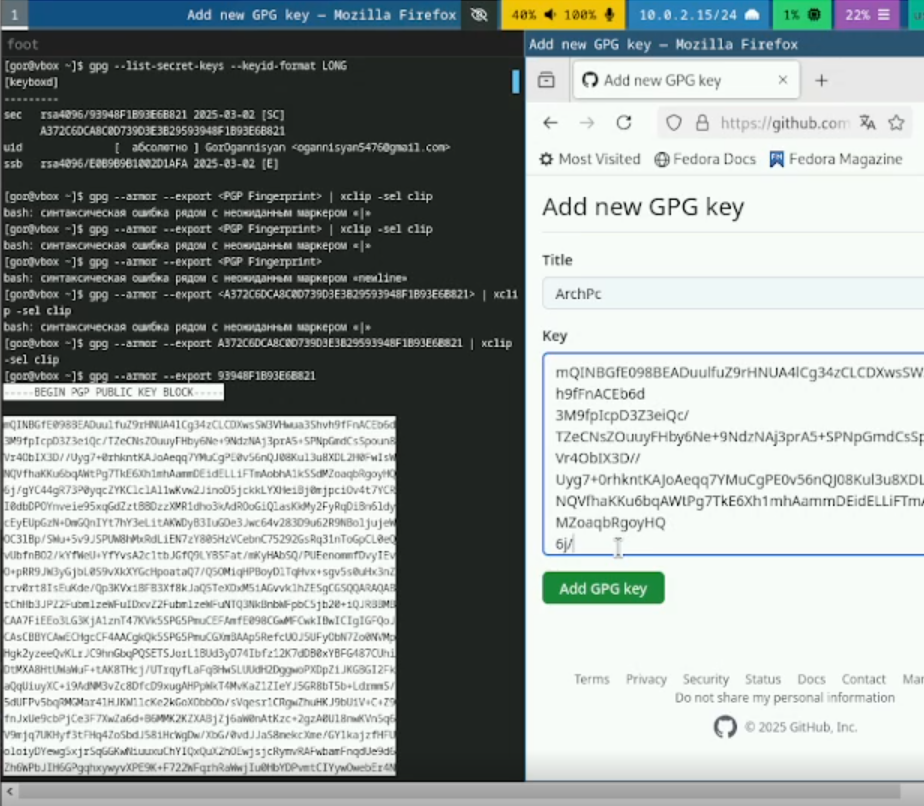
Делаем базовую настройку git.



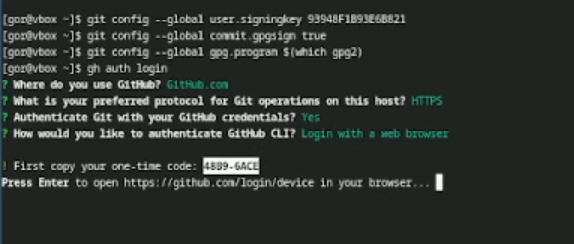
Создаем ключи ssh и pgp.



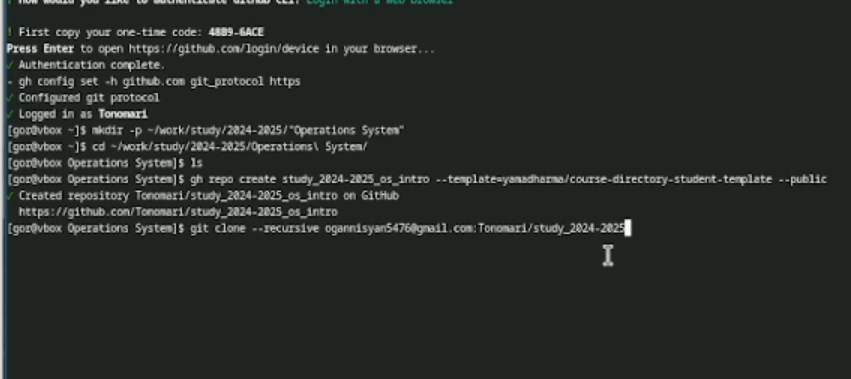
Делаем настройку github.



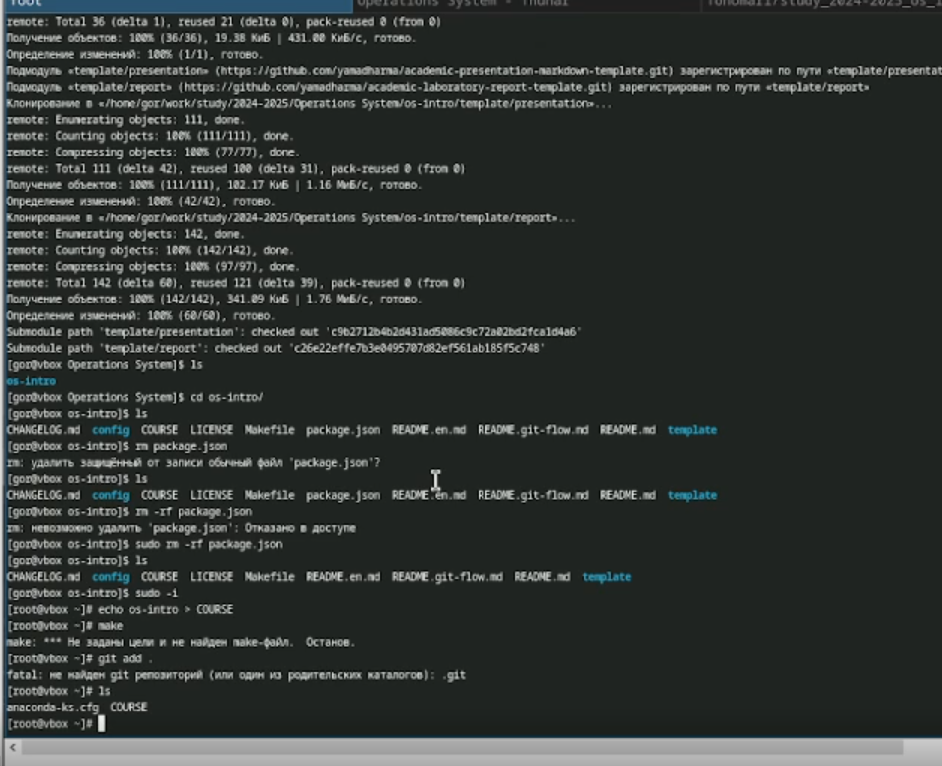
Добавляем pgp ключ в github.



Делаем настройку автоматический подписей коммитов и авторизуемся в gh.



Создаем репозиторий курса на основе шаблона.

Делаем настройку каталога курса.

Выводы, согласованные с задание работы

В данной лабораторной работе мы настроили git и gh для работы с github. Создали репозиторий для добавления своих работ туда.

Ответы на контрольные вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Системы контроля версий (VCS) — это инструменты, которые помогают разработчикам управлять изменениями в исходном коде и других файлах проекта. Они предназначены для отслеживания изменений, сохранения истории изменений, совместной работы над проектом и обеспечения возможности возврата к предыдущим версиям.
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.
   * Хранилище (Repository): Центральное место, где хранятся все версии файлов проекта.
   * Commit (Коммит): Зафиксированное изменение в хранилище, содержащее набор изменений и сообщение о том, что было изменено.
   * История (History): Последовательность всех коммитов, показывающая, как проект изменялся со временем.
   * Рабочая копия (Working Copy): Локальная копия проекта на компьютере разработчика, с которой он работает и вносит изменения.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.
   * Централизованные VCS (CVCS): Все изменения и история хранятся на центральном сервере. Примеры: Subversion (SVN), Perforce.
   * Децентрализованные VCS (DVCS): Каждый разработчик имеет полную копию хранилища, включая всю историю изменений. Примеры: Git, Mercurial.
4. Опишите действия с VCS при единоличной работе с хранилищем. При единоличной работе разработчик создает локальное хранилище, вносит изменения, фиксирует их с помощью коммитов и может возвращаться к предыдущим версиям при необходимости.
5. Опишите порядок работы с общим хранилищем VCS. Разработчики клонируют общее хранилище, работают с локальной копией, вносят изменения, коммитят их локально, а затем отправляют (push) изменения в общее хранилище. При необходимости они могут получать (pull) обновления от других разработчиков.
6. Каковы основные задачи, решаемые инструментальным средством git? Git решает задачи управления версиями, обеспечивая возможность отслеживания изменений, совместной работы, управления ветками и слияния изменений.
7. Назовите и дайте краткую характеристику командам git.
   * git init: Инициализация нового локального хранилища.
   * git clone: Клонирование удаленного хранилища.
   * git add: Добавление изменений в индекс (staging area).
   * git commit: Создание коммита.
   * git push: Отправка изменений в удаленное хранилище.
   * git pull: Получение изменений из удаленного хранилища.
   * git branch: Работа с ветками.
   * git merge: Слияние веток.
8. Приведите примеры использования при работе с локальным и удаленным репозиторием.
   * Локальный репозиторий: git init, git add, git commit.
   * Удаленный репозиторий: git clone, git push, git pull.
9. Что такое и зачем могут быть нужны ветви (branches)? Ветви позволяют разработчикам работать над различными задачами или функциями параллельно, не мешая друг другу. Они используются для разработки новых функций, исправления ошибок и экспериментов.
10. Как и зачем можно интегрировать некоторые файлы при commit? Интеграция определенных файлов при коммите позволяет разработчикам выборочно включать изменения в коммит, что помогает управлять изменениями более гибко и избегать включения ненужных или незавершенных изменений. Это делается с помощью команды git add перед git commit.