Лабораторная работа №2

Отчёт

Сергеев Даниил Олегович

Содержание

# 1 Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git. [1]

# 2 Задание

* Создать базовую конфигурацию для работы с git.
* Создать ключ SSH.
* Создать ключ PGP.
* Настроить подписи git.
* Зарегистрироваться на Github.
* Создать локальный каталог для выполнения заданий по предмету.

# 3 Выполнение лабораторной работы

## 3.1 Установка необходимого ПО (git, gh)

Установим git и gh. (рис. 1)

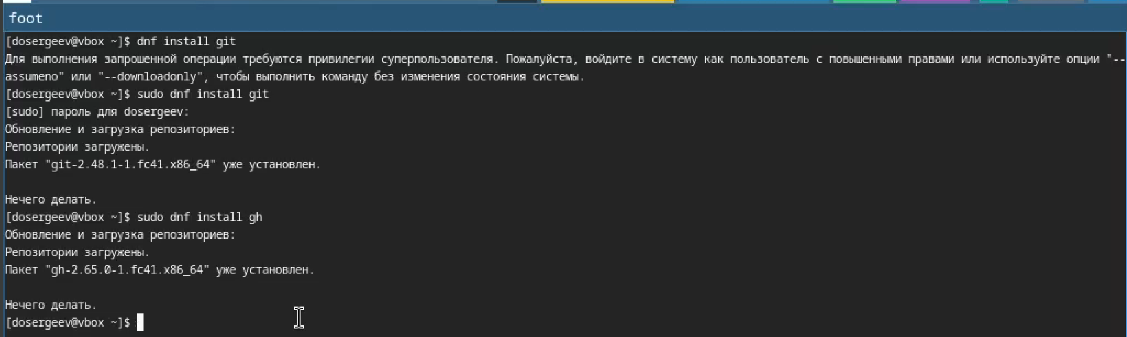


Рис. 1: Процесс установки git и gh.

Необходимые пакеты уже установлены, поэтому продолжим.

## 3.2 Базовая настройка git

Зададим имя и почту владельца репозитория, настроим utf-8 в выводе сообщений git, зададим имя начальной ветки master, параметр autocrlf и safecrlf. (рис. 2)

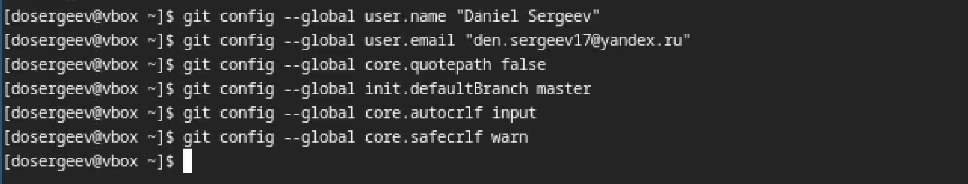


Рис. 2: Настройка параметров git.

## 3.3 Создание ключа ssh

Cоздадим ключ ssh по алгоритму ed25519. (рис. 3)

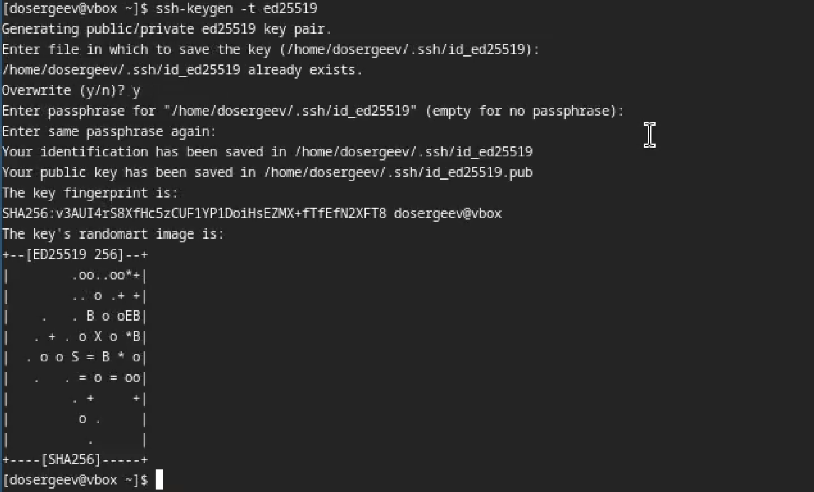


Рис. 3: Генерация ключа ssh.

Выведем ключ, скопируем и вставим в гит.

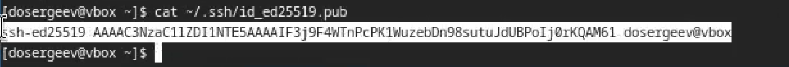


Рис. 4: Копирование ключа ssh.

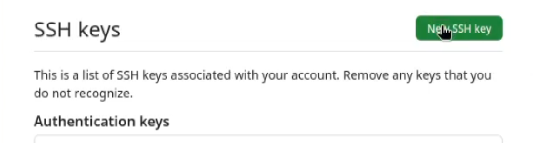


Рис. 5: Добавялем ключ ssh в гите.

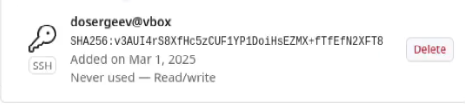


Рис. 6: Вид ключа ssh в гите.

## 3.4 Создание ключа gpg

сгенерируем ключ с помощью команды gpg –full-generate-key (рис. 7). В предложенных вариантах выберем: - тип RSA and RSA. - размер 4096. - срок действия по умолчанию. - имя. - адрес электронной почты. - без комментария.

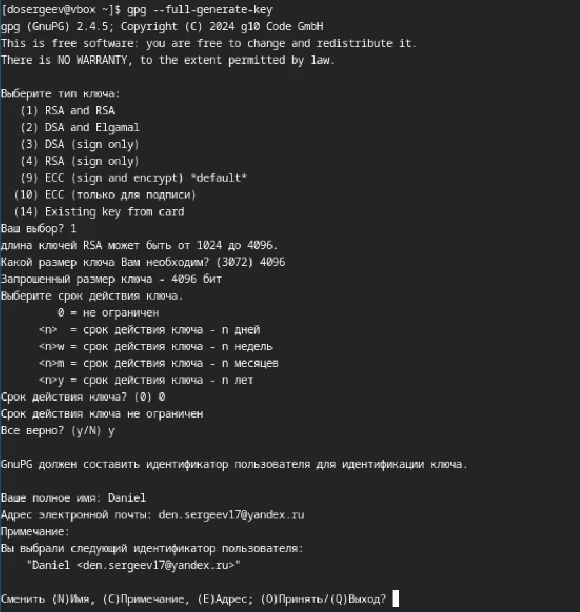


Рис. 7: Генерация ключа gpg.

## 3.5 Добавление ключа gpg в GitHub

Выведем список ключей. (рис. 8)

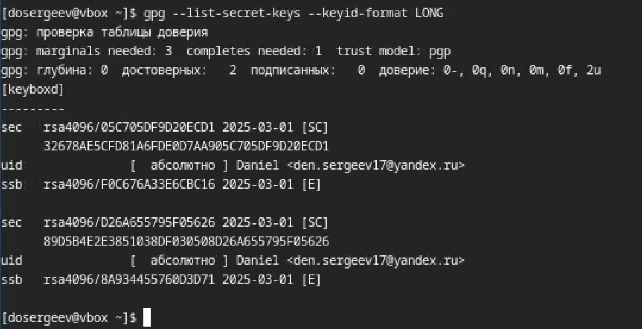


Рис. 8: Список ключей gpg.

Cкопируем сгенерированный ключ в буфер обмена с помощью его отпечатка

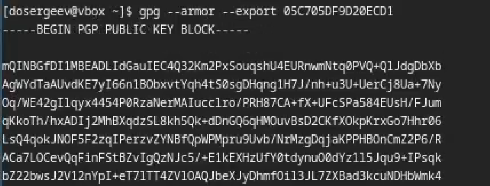


Рис. 9: Копирование ключа gpg.

Добавим ключ gpg в Github.

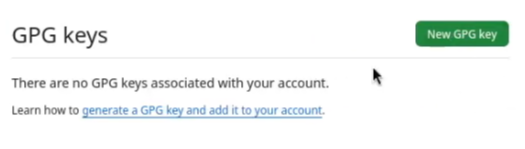


Рис. 10: Добавляем ключ gpg в гите.

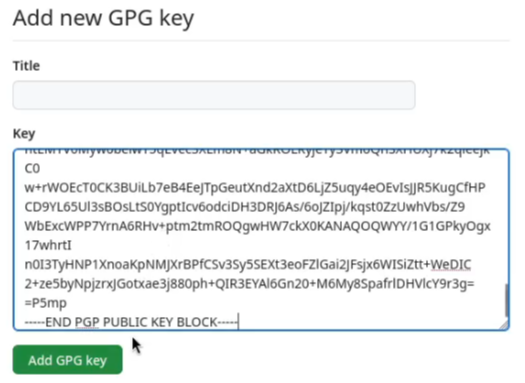


Рис. 11: Вводим в окно для ключа gpg.

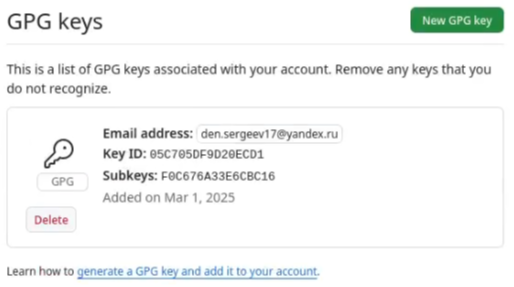


Рис. 12: Вид ключа gpg в гите.

## 3.6 Настройка автоматических подписей коммитов git

Указываем git-у применять адрес почты при подписи коммитов.

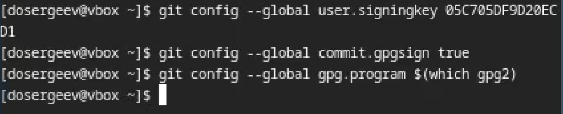


Рис. 13: Настройка подписей коммитов git.

Авторизуемся через браузер.

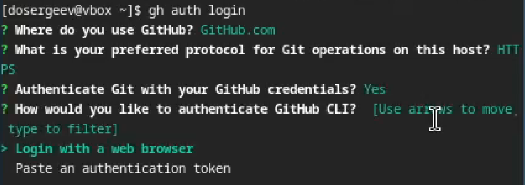


Рис. 14: Выбираем вход через браузер.

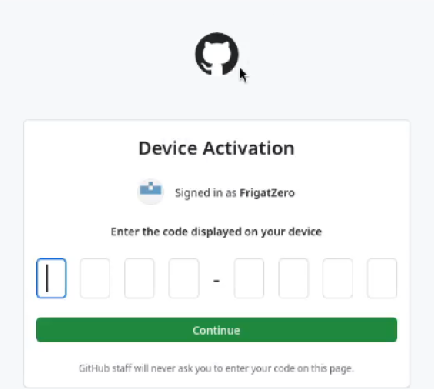


Рис. 15: Вводим ключ.

## 3.7 Создание репозитория курса на GitHub

Создадим репозиторий гит и каталог курса. (рис. 16)

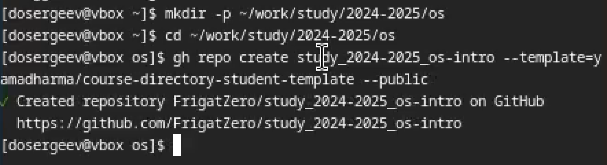


Рис. 16: Создание репозитория курса.

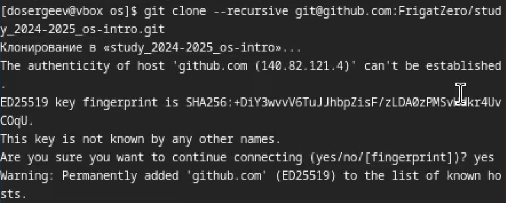


Рис. 17: Клонирование удалённого репозитория в локальный.

Перейдем в каталог курса и удалим лишние файлы. Создадим необходимые каталоги с помощью make prepare.

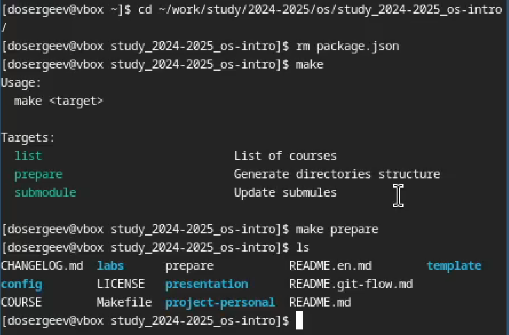


Рис. 18: Удаление лишних файлов и создание необходимых каталогов make.

Отправим файлы на сервер.

Сохраняем изменения.

Рис. 19: Сохраняем изменения.

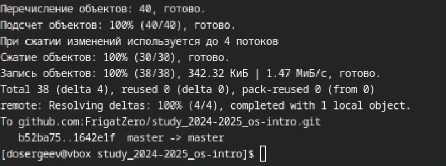


Рис. 20: Отправляем файлы на сервер.

# 4 Ответы на контрольные вопросы

1.Системы контроля версий - это инструменты, позволяющие организовать работу над проектом разработки, выкладывать его обновления и контролировать релизы и изменения кода. Они предназначаются для отслеживания изменений, защиты исходного кода от удаления и изменения, возможности отката изменений и командной работы для 10 и более человек.

* Хранилище - репозиторий в котором храняться файлы проекта в различных версиях.
* Commit - комментарий внесённых изменений в репозитории.
* История - история изменений файлов проекта.
* Рабочая копия - копия, созданная из определенной версии репозитория, которую модифицирует разработчик.

3.Централизованные системы контроля версий имеют единый сервер под хранение проекта, для изменения которых необходимо скачать необходимые файлы, изменить и вернуть обратно на сервер. Пример центроализованной VCS: Subversion. Децентрализованные системы полностью копируют удалённый репозиторий в локальный. При этом внесенные изменения отправляются на сервер в качестве новой версии. Пример: git.

4.Создается репозиторий для работы с проектом, при необходимости файлы обновляются локально и отправляются на сервер в качестве новой версии.

5.Репозиторий копируется локально. После внесения изменений файлы загружаются на сервер в качестве отдельной версии. После этого измененная ветка может быть объединена с текущей и отправлена в релиз.

6.Хранение файлов проекта, отслеживание версий, защита от изменений, работа в команде.

* git clone – клонирование проекта с сервера в указанный локальный репозиторий.
* git add – добавляет все изменённые или созданные файлы или каталоги.
* git commit – сохраняет изменения репозитория с комментарием.
* git push – загружает добавленные изменения на сервер
* git pull – получить последние изменения с сервера
* git rm – удаляет файл из индекса репозитория
* git status – просмотривает список измененных файлов в текущем репозитории

8.С локальным: git add . , git commit -am commit, git push - добавление всех измененнных файлов в текущем каталоге, их сохранение с комментарием commit, git pull - получаем последние изменения с удалённого репозитория. С удалённым: git push - отправляем сохраненные изменения на сервер.

9.Ветки - это различные версии исходного репозитория, являющиеся копиями с внесенными изменениями. Нужны для параллельной работы над проектом. Могут быть объединенны для внесения изменений. Их можно проигнорировать, добавив имя в .gitignore.

10.Мы можем игнорировать некоторые файлы при коммитах, когда не хотим добавлять их в удаленный репозиторий для чистоты или упрощения.

# 5 Вывод

В результате выполнения лабораторной работы я изучил как применять средства контроля версий для работы с удалённым репозиторием и освоил умения по работе с git.

# Список литературы

1. Kulyabov. [Лабораторная работа № 2. Первоначальная настройка git](https://esystem.rudn.ru/mod/page/view.php?id=1224095). RUDN.