Отчёт по лабораторной работе №2:

Первоначальна настройка git

Коротков Андрей Романович

Table of Contents

# 1 Цель работы

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

# 2 Задания

1. Создать базовую конфигурацию для работы с git.
2. Создать ключи SSH и PGP.
3. Настроить подписи git.
4. Создать локальный каталог для выполнения заданий по предмету.

# 3 Выполнение лабораторной работы

### 3.0.1 Задание 1

Для начала установим программное обеспечение коммандами **dnf install git** и **dnf install gh**

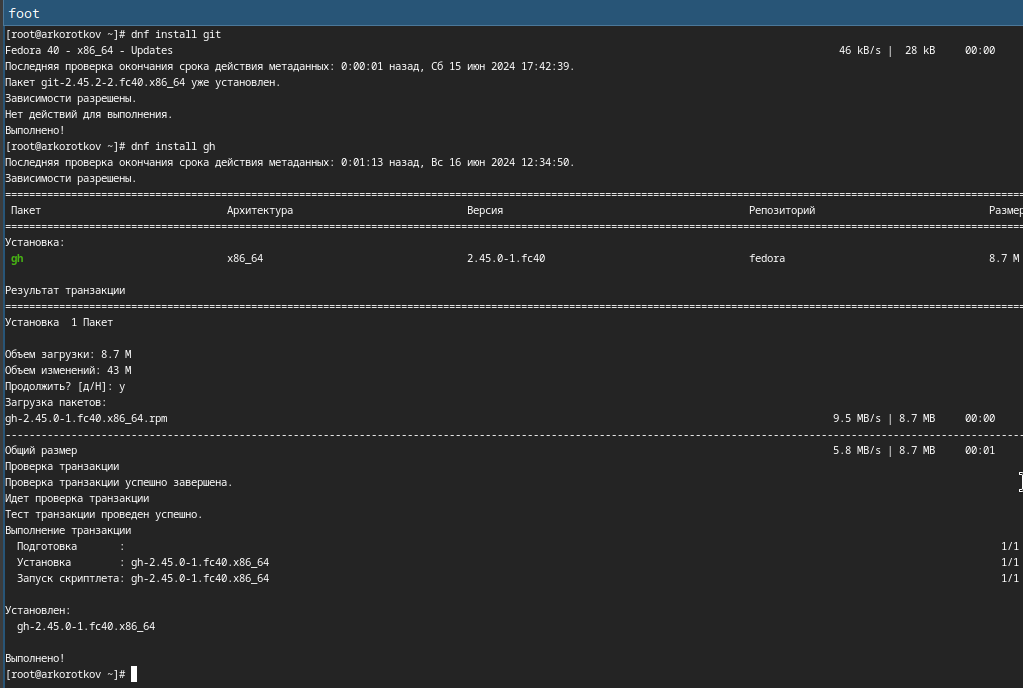


Рис. 1: Установка программного обеспечения

Проведём базовую настройку:

* Зададим имя и email владельца репозитория командами **git config –global user.name “Name Surname”** и **git config –global user.email “work@mail”**

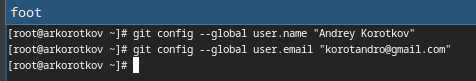


Рис. 2: Настройка имени и почты владельца

* Настроим **utf-8** в выводе сообщений git командой **git config –global core.quotepath false**
* Зададим имя начальной ветки командой **git config –global init.defaultBranch master**
* Настроим параметр **autocrlf** командой **git config –global core.autocrlf input**
* Настроим параметр **safecrlf** командой **git config –global core.safecrlf warn**

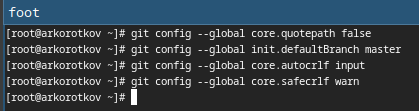


Рис. 3: Настройка конфига

### 3.0.2 Задание 2

Создадим ключ *ssh* о алгоритму *rsa* с размером 4096 бит командой **ssh-keygen -t rsa -b 4096**.

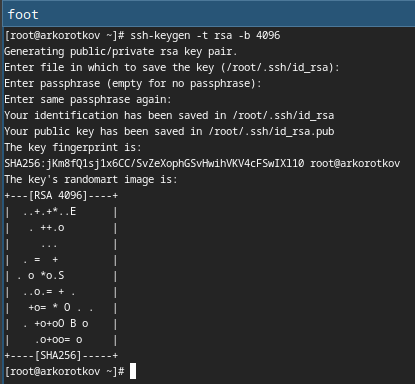


Рис. 4: Создание ключа ssh(rsa)

Создадим ключ *ssh* о алгоритму *ed25519* командой **ssh-keygen -t ed25519**

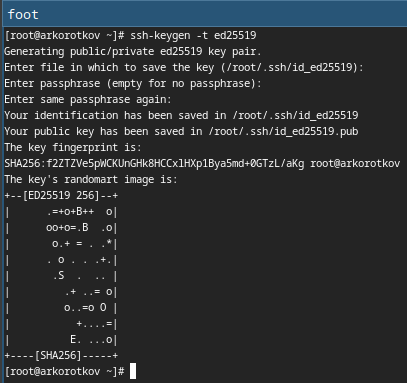


Рис. 5: Создание ключа ssh(ed25519)

Создадим ключ *pgp* командой **gpg –full-generate-key**. Из предложенных опций выбираем:

* тип *RSA* and *RSA*;
* размер 4096;
* выбаем срок действия;

Вводим личную кинформацию и комментарий.

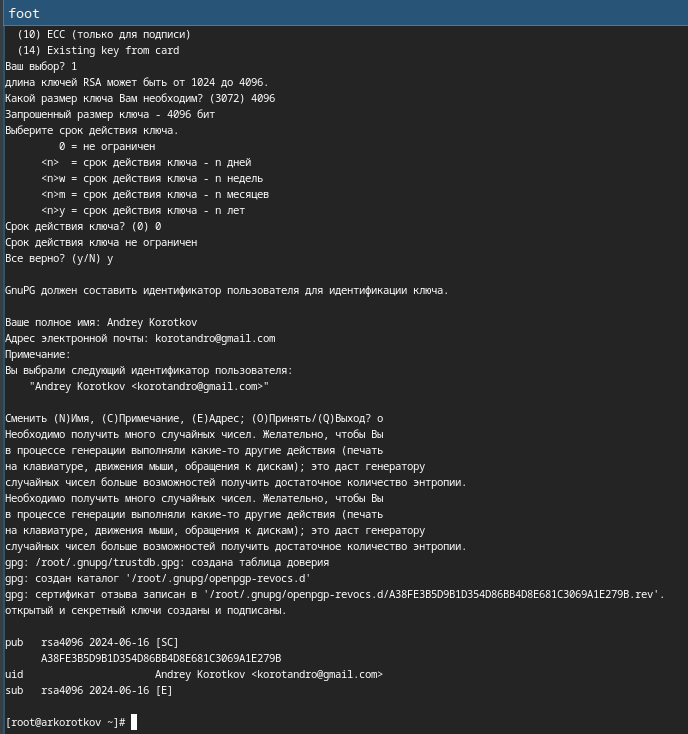


Рис. 6: Создание ключа pgp

### 3.0.3 Задание 3

#### 3.0.3.1 Добавление PGP ключа в GitHub

Копируем приватный ключ в файл командой **gpg –armor –export почта\_владельца > /файл.txt**

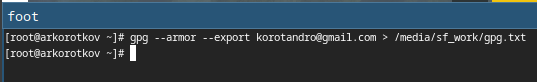


Рис. 7: Копирование ключа pgp

Переходим на https://github.com/settings/keys и добавляем полученный ключ.

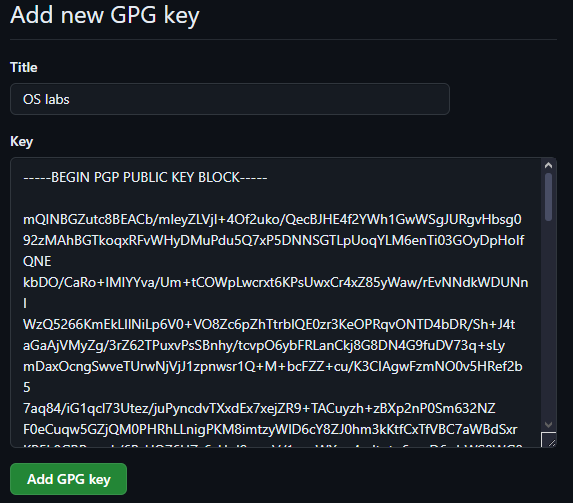


Рис. 8: Добавление ключа pgp

#### 3.0.3.2 Настройка автоматических подписей коммитов git

Используя введёный email, укажим Git применять его при подписи коммитов используя команды:

* **git config –global user.signingkey**
* **git config –global commit.gpgsign true**
* **git config –global gpg.program $(which gpg2)**

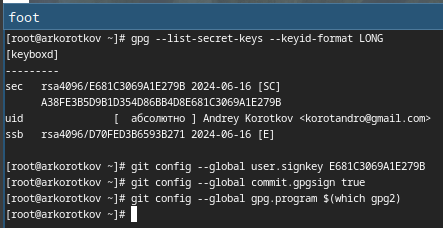


Рис. 9: Настройка подписей

#### 3.0.3.3 Настройка gh

Авторизация командой **gh auth login**

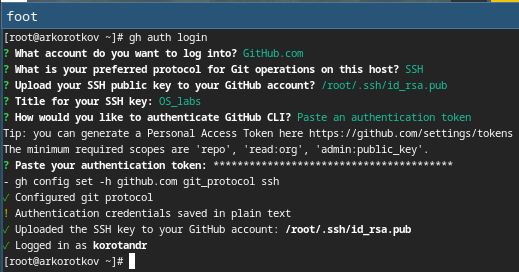


Рис. 10: Настройка подписей

### 3.0.4 Задание 4

* Создадим каталог для репозитория командой **mkdir -p ~/study/2024/OS**
* Перейдём в созданный каталог командой **cd ~/study/2024/OS**
* Создадим репозиторий на основе шаблона командой **gh repo create study\_2024\_os-intro –template=yamadharma/course-directory-student-template –public**
* Клонируем созданный репозиторий в каталог командой **git clone –recursive git@github.com:korotandr/study\_2024\_os-intro.git os-intro**

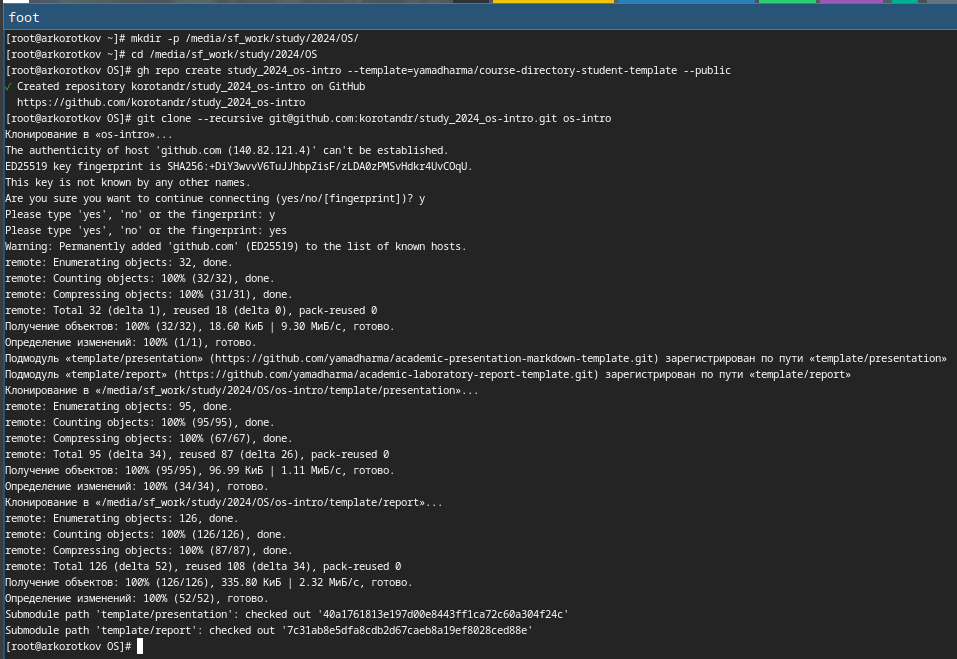


Рис. 11: Создание репозитория

Настройка каталога курса:

* Перейти в каталог курса командой **cd os-intro**
* Удалить лишние файлы командой **rm package.json**
* Создать необходимые каталоги командой **make prepare**

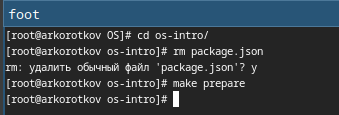


Рис. 12: Создание структуры каталога

Отправка файлов на сервер последовательностью команд:

* **git add .**
* **git commit -am ‘feat(main): make course structure’**
* **git push**

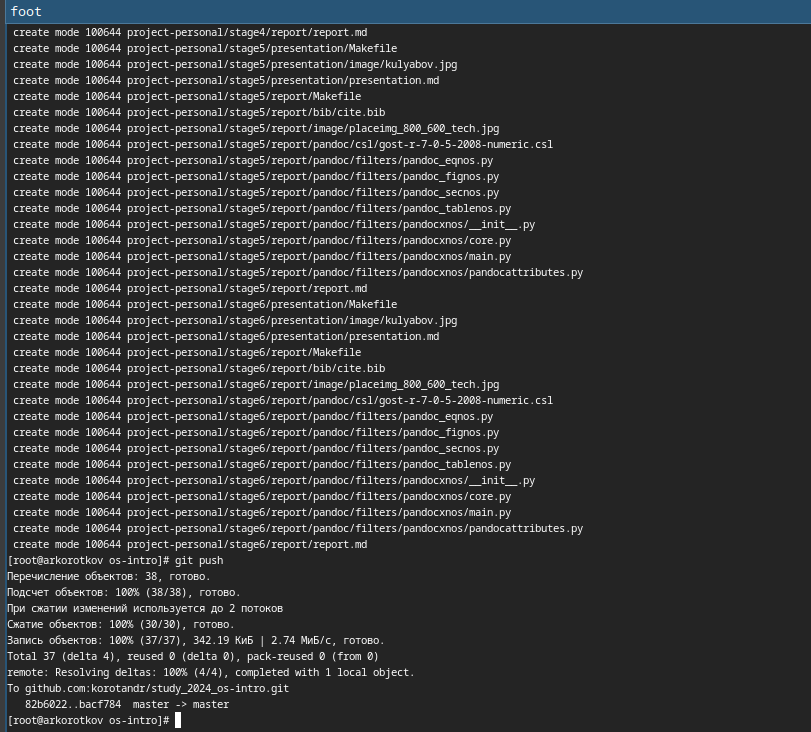


Рис. 13: Отправка файлов на сервер

# 4 Выводы

В ходе данной лабороторной работы я смог изучить идеологию и применение средств контроля версий, а так же освоил основные аспекты работы с git.

# 5 Ответы на контрольные вопросы

1.Системы контроля версий (Version Control Systems, VCS) — это программные инструменты, предназначенные для управления изменениями в исходном коде, документации и других данных в процессе разработки. Основные задачи:

* Отслеживание изменений
* Совместная работа
* Управление версиями
* Восстановление версий
* Резервное копирование и восстановление

2.Хранилище: Это центральное место, где сохраняется вся история изменений проекта, включая все версии файлов, метаданные изменений (например, авторы и временные метки), и другие данные, необходимые для управления версиями.

Commit: Это операция, при которой изменения, внесённые в файлы рабочей копии, сохраняются в хранилище как отдельная версия (коммит).

История: Это последовательность всех коммитов, сохранённых в хранилище.

Рабочая копия: Это текущая версия файлов проекта на локальной машине разработчика, с которой он непосредственно работает.

Взаимосвязь этих понятий:

* Разработчик клонирует или извлекает репозиторий, создавая рабочую копию на своём компьютере.
* В рабочей копии он вносит изменения в файлы проекта.
* После завершения работы он фиксирует (commit) изменения, добавляя новый коммит в историю репозитория.
* Хранилище обновляется, и новая версия проекта становится доступной для других участников команды.
* История изменений позволяет отслеживать все внесённые изменения и при необходимости откатываться к предыдущим версиям.

3.Централизованные системы контроля версий (CVCS)

* Единое хранилище
* Обмен данными через сервер
* Простота администрирования
* Примеры:
  + CVS (Concurrent Versions System)
  + Subversion (SVN)

Децентрализованные системы контроля версий (DVCS)

* Полные копии репозитория
* Работа офлайн
* Масштабируемость и гибкость
* Примеры:
  + Git
  + Mercurial

4.Примерный процесс работы:

* Инициализация или клонирование репозитория
* Работа с файлами
* Отслеживание изменений
* Коммит изменений
* Синхронизация
* Просмотр истории

5.Примерный процесс работы в команде:

* Клонирование репозитория
* Обновление перед началом работы
* Работа с файлами и коммиты
* Регулярная синхронизация
* Разрешение конфликтов
* Push изменений
* Просмотр истории и анализ

6.Основные задачи, решаемые инструментальным средством **git**:

* Отслеживание изменений
* Управление версиями
* Совместная работа
* Создание и управление ветками
* Объединение изменений (Merge)
* Разрешение конфликтов
* История изменений
* Обеспечение целостности данных
* Резервное копирование и восстановление
* Работа офлайн
* Интеграция с CI/CD

7.**git init** — создание нового репозитория в текущей директории.

**git clone URL** — клонирование удалённого репозитория на локальную машину.

**git add file** — добавление файла в коммит.

**git commit -m “Сообщение”** — создание коммита с описанием изменений.

**git branch branch-name** — создание новой ветки.

**git checkout branch-name** — переключение на другую ветку.

**git merge branch-name** — объединение изменений из указанной ветки в текущую.

**git push origin branch-name** — отправка локальных изменений в удалённый репозиторий.

**git pull origin branch-name** — извлечение изменений из удалённого репозитория и объединение их с локальными.

**git log** — просмотр истории коммитов.

**git fetch** — синхронизация с удалённым репозиторием

8.Работа с локальным репозиторием:

* git init
* git add file1.txt file2.txt
* git commit -m “Initial commit”
* git status
* git log
* git branch new-feature
* git checkout new-feature
* git checkout master
* git merge new-feature
* Разрешение конфликтов

Работа с удалённым репозиторием:

* git clone https://github.com/user/repository.git
* git remote add origin https://github.com/user/repository.git
* git push origin master
* git pull origin master
* git remote -v
* git checkout -b new-feature
* git push origin new-feature
* git push origin –delete new-feature
* git fetch

9.Ветви (branches) в системах контроля версий, являются важным инструментом для управления разработкой проекта. Они позволяют разработчикам работать над различными задачами, функциями или исправлениями независимо друг от друга.

10.Игнорирование некоторых файлов при commit в системах контроля версий позволяет исключить из репозитория временные, сгенерированные или конфиденциальные файлы, которые не должны быть частью версионного контроля. Для этого в Git используется специальный файл .gitignore.