# **Лабораторная работа:** Базовая настройка Git

## **ФИО студента:**

Рафи Кази ар

НКАбд-03-24

1032238132@pfur.ru

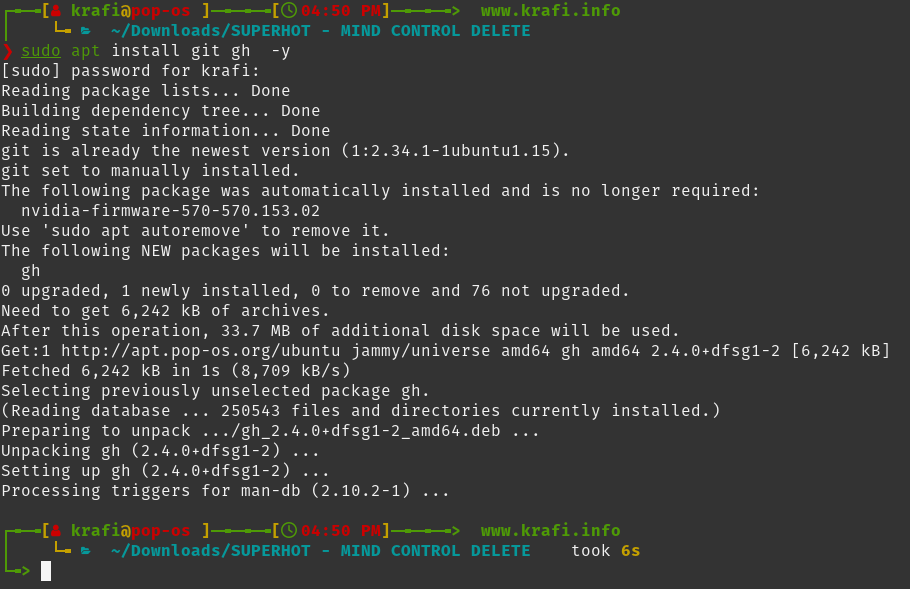
Кулябов Дмитрий Сергеевич

Moscow

1 — "Базовая настройка Git"

### **1. Установка программного обеспечения**

sudo apt install git gh -y



### **2. Базовая настройка Git**

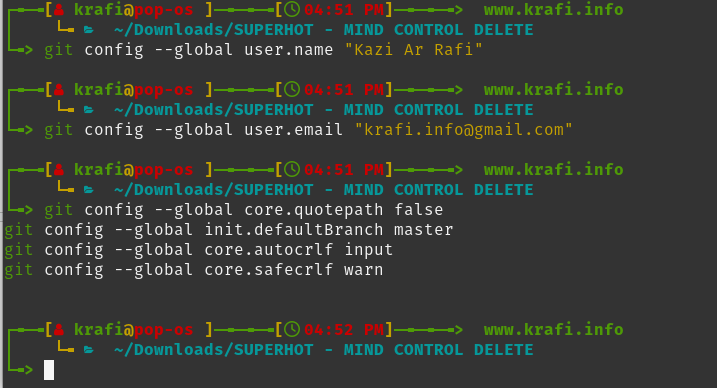
git config --global user.name "Kazi Ar Rafi”

git config --global user.email "krafi.info@gmail.com"

git config --global core.quotepath false

git config --global init.defaultBranch master

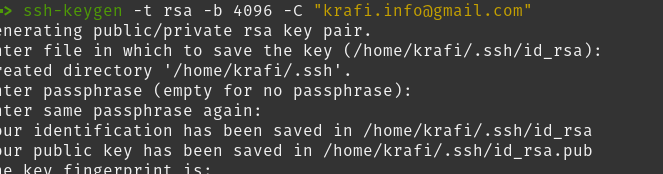
git config --global core.autocrlf input

git config --global core.safecrlf warn

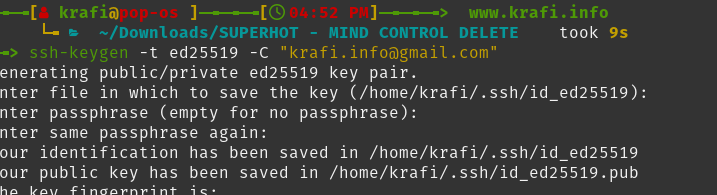
### **3. Создание SSH-ключей**

#### **RSA (4096 бит):**

ssh-keygen -t rsa -b 4096 -C "ваш[\_email@example.com](mailto:_email@example.com)"

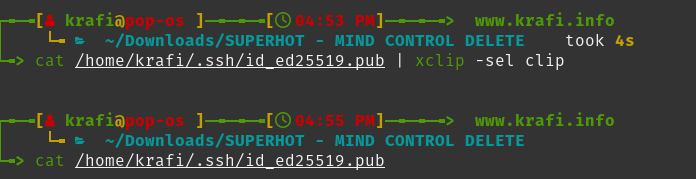


#### **Ed25519 (современный \ modern):**

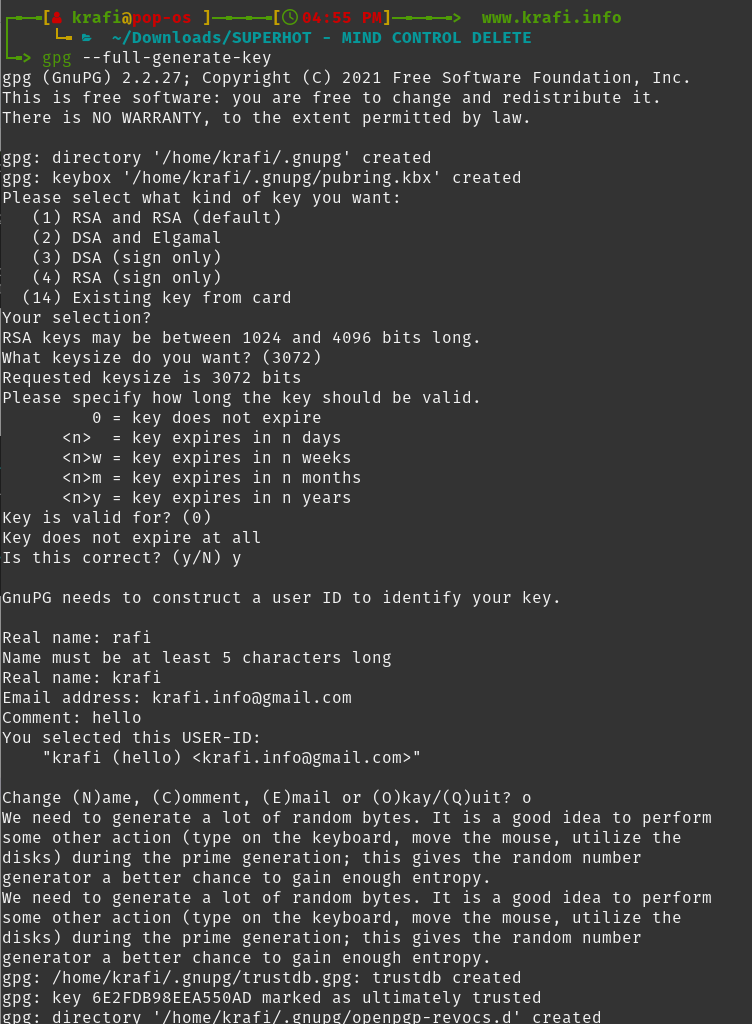
ssh-keygen -t ed25519 -C "ваш[\_email@example.com](mailto:_email@example.com)"

#### **Копирование публичного ключа:**

cat ~/.ssh/id\_ed25519.pub | xclip -sel clip

1. Settings → SSH and GPG keys .
2. Нажмите New SSH key , вставьте скопированный ключ.

### **4. Создание PGP-ключа (create):**

gpg --full-generate-key

* Тип: RSA and RSA
* Размер: 4096
* Срок действия: 0 (без ограничений)
* Укажите имя, email (должен совпадать с тем, что указан в git config --global user.email), комментарий.

#### **Получение отпечатка ключа:**

#### **Get key fingerprint:**

gpg --list-secret-keys --keyid-format LONG

#### **Экспорт публичного ключа: Export public key:**

gpg --armor --export 1234ABCD5678EFGH90IJKLNMOPQ2RSTUVWXYZ123 | xclip -sel clip

#### **Добавление ключа в GitHub:**

Settings → SSH and GPG keys .

New GPG key , вставьте содержимое из буфера обмена.

### **5. Настройка автоматических подписей коммитов**

### **Configuring Automatic Commit Signing**

git config --global user.signingkey 1234ABCD5678EFGH90IJKLNMOPQ2RSTUVWXYZ123

git config --global commit.gpgsign true

git config --global gpg.program $(which gpg)

### **6. Авторизация через GitHub CLI**

gh auth login

### **7. Создание рабочего пространства**

#### **Создание каталога проекта:**

mkdir -p ~/work/study/2022-2023/"Операционные системы"

cd ~/work/study/2022-2023/"Операционные системы"

#### **Создание репозитория на основе шаблона:**

#### **Create repo based on template:**

gh repo create study\_2022-2023\_os-intro --template=yamadharma/course-directory-student-template --public

git clone --recursive git@github.com:<your\_username>/study\_2022-2023\_os-intro.git os-intro

#### **Настройка структуры курса:**

#### **Configure course structure:**

cd ~/work/study/2022-2023/"Operating Systems"/os-intro

rm package.json

echo os-intro > COURSE

make

#### **Отправка изменений:**

#### **Push changes:**

git add .

git commit -am 'feat(main): make course structure'

git push

## **📚 Контрольные вопросы**

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

Ответ:  
Системы контроля версий (VCS) позволяют отслеживать изменения файлов проекта, сохранять историю изменений, работать над проектом нескольким разработчикам одновременно, а также восстанавливать предыдущие версии.

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

Ответ:

* Хранилище (репозиторий) — это место, где хранится вся история изменений проекта.
* Commit — это зафиксированное состояние изменений.
* История — это последовательность коммитов, показывающая развитие проекта.
* Рабочая копия — это локальная копия проекта, с которой работает пользователь.

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

Ответ:

* Централизованные VCS : есть один главный сервер, все работают с ним (например, SVN).
* Децентрализованные VCS : каждый участник имеет полную копию репозитория (например, Git, Mercurial).

### **4. Опишите действия с VCS при единоличной работе с хранилищем.**

Ответ:  
При единоличной работе вы:

* Создаете локальный репозиторий (git init).
* Добавляете файлы (git add).
* Сохраняете изменения (git commit).
* Может быть, отправляете их на удалённый сервер (git push).

### **5. Опишите порядок работы с общим хранилищем VCS.**

Ответ:

* Клонируете репозиторий (git clone).
* Переключаетесь на нужную ветку (git checkout).
* Вносите изменения, добавляете их (git add) и делаете коммит (git commit).
* Отправляете изменения (git push).
* Если есть изменения у других участников — получаете их (git pull).

### **6. Каковы основные задачи, решаемые инструментальным средством git?**

Ответ:

* Управление версиями кода.
* Совместная разработка.
* Откат изменений.
* Работа с ветками.
* Поддержка истории изменений.

**7. Назовите и дайте краткую характеристику командам git.**

Ответ:

* git init — Инициализация нового репозитория
* git clone — Клонирование существующего репозитория
* git add — Добавление файлов в индекс (подготовка к коммиту)
* git commit — Сохранение изменений с комментарием (фиксация изменений в истории)
* git status — Показывает текущее состояние репозитория (какие файлы изменены, добавлены и т.д.)
* git push — Отправка локальных коммитов на удалённый репозиторий
* git pull — Получение и слияние изменений из удалённого репозитория с локальной веткой
* git branch — Управление ветками (создание, удаление, просмотр списка)
* git checkout — Переключение между ветками или восстановление файлов
* git merge — Слияние одной ветки с текущей

### **8. Приведите примеры использования при работе с локальным и удалённым репозиториями.**

Ответ:

* Локальный репозиторий :

git init

git add .

git commit -m "Initial commit"

Удалённый репозиторий :

git remote add origin git@github.com:user/repo.git

git push -u origin master

### **9. Что такое и зачем могут быть нужны ветви (branches)?**

Ответ:  
Ветви позволяют работать над разными частями проекта независимо друг от друга. Например, одна ветка может использоваться для разработки, другая — для тестирования, третья — для исправления ошибок.

### **10. Как и зачем можно игнорировать некоторые файлы при commit?**

Ответ:  
Файлы, не относящиеся к исходному коду (временные, объектные, системные), можно игнорировать, создав файл .gitignore. Это позволяет избежать загрузки лишних данных в репозиторий.

Пример содержания .gitignore:

\*.log

\_\_pycache\_\_

env/

.DS\_Store

## **Выводы по работе:**

Выполнив данную лабораторную работу, я:

* Настроил Git под свои нужды.
* Создал и зарегистрировал SSH- и PGP-ключи.
* Настроил автоматическую подпись коммитов.
* Зарегистрировался на GitHub и создал репозиторий курса.
* Изучил и применил базовые команды Git для работы с локальным и удалённым репозиторием.
* Разобрался с принципами управления версиями и безопасностью.