РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

дисциплина: Операционные системы

Студент: Кармацкий Никита Сергеевич

Группа: НФИбд-01-21

Москва

2022 г.

Управление версиями

Цель работы:

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

Теоретическое введение

Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется

Основные команды git

Наиболее часто используемые команды git: – создание основного дерева репозитория: **git init** – получение обновлений (изменений) текущего дерева из центрального репозитория: **git pull** – отправка всех произведённых изменений локального дерева в центральный репозиторий: **git push** – просмотр списка изменённых файлов в текущей директории: **git status** – просмотр текущих изменения: **git diff** – сохранение текущих изменений:

- добавить все изменённые и/или созданные файлы и/или каталоги: **git add** .
- добавить конкретные изменённые и/или созданные файлы и/или каталоги: **git add имена_файлов**
- удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): **git rm имена_файлов**

– сохранение добавленных изменений:

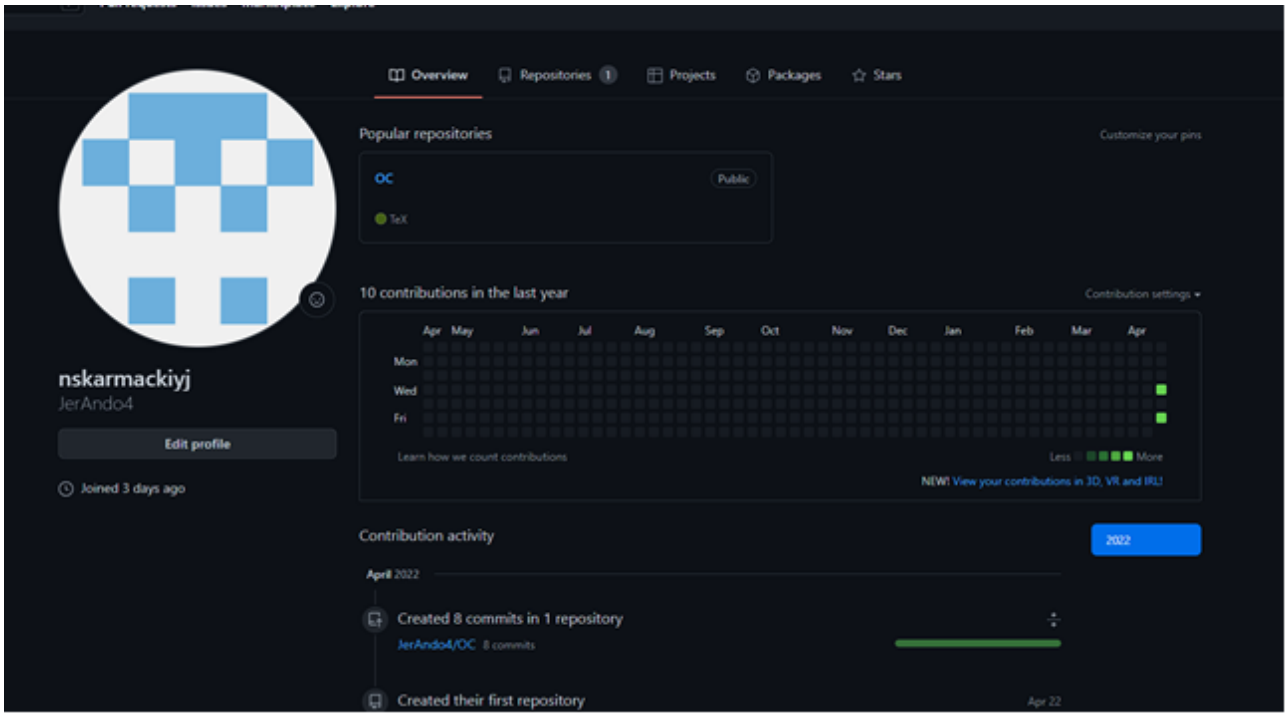
- сохранить все добавленные изменения и все изменённые файлы: **git commit -am 'Описание коммита'**
- сохранить добавленные изменения с внесением комментария через встроенный редактор: **git commit** – создание новой ветки, базирующейся на текущей: **git checkout -b имя_ветки** – переключение на некоторую ветку: **git checkout имя_ветки** (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: **git push origin имя_ветки** – слияние ветки с текущим деревом: **git merge —no-ff имя_ветки**

– удаление ветки:

- удаление локальной уже слитой с основным деревом ветки: **git branch -d имя_ветки**
- принудительное удаление локальной ветки: **git branch -D имя_ветки**
- удаление ветки с центрального репозитория: **git push origin :имя_ветки**

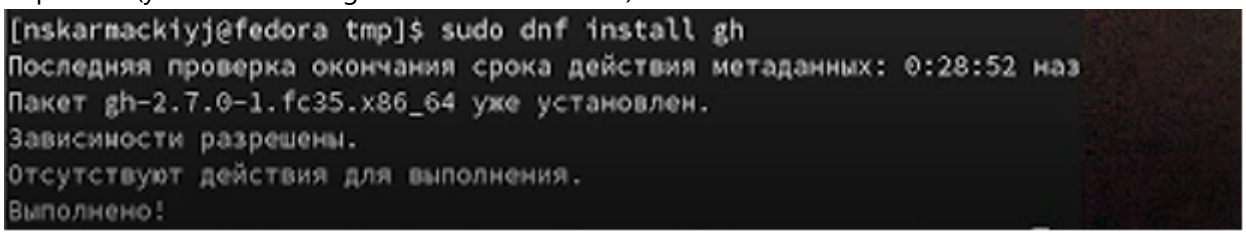
Ход работы

1. Создайте учётную запись на https://github.com (На скриншоте мы уже создали нашу учетную запись)



2. Установка программного обеспечения

Скриншот(устанавливаем git-flow в Fedora Linux)



Устанавливаем gh в Fedora Linux(на скриншоте он уже установлен)

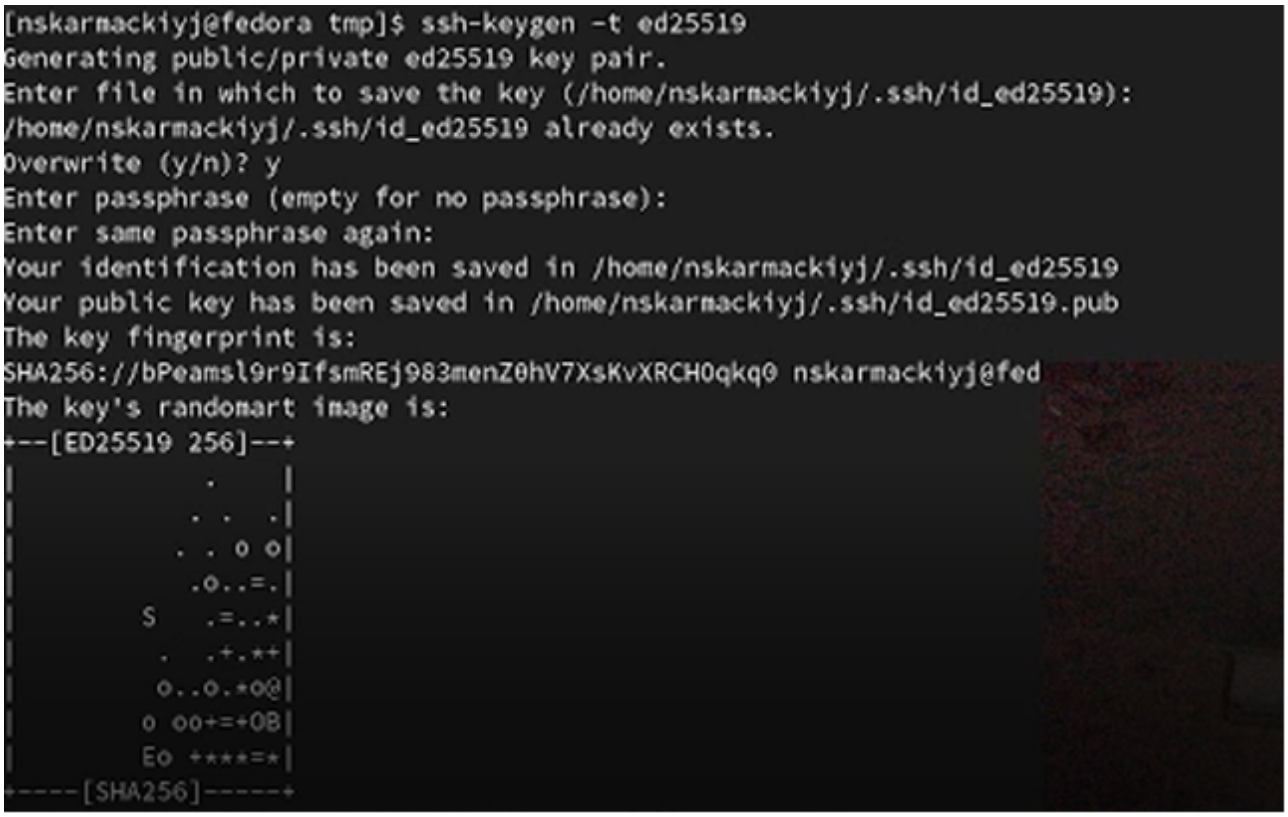
3. Базовая настройка git Пишем наше имя и почту, и другие функции

```
nskarmackiyj@fedora tmp]$ git config --global user.name "nskarmackiyj"
nskarmackiyj@fedora tmp]$ ^[[200~git config --global user.email "work@mail"
[[201~bash: git: command not found...
it config --global user.emagit config --global user.email "work@mail"git config --global user.email "work@mail"
ash: ~git: command not found...
imilar command is: 'git'
nskarmackiyj@fedora tmp]$ git config --global user.email "jernado4@gmail.com"
nskarmackiyj@fedora tmp]$ git config --global core.quotepath false
nskarmackiyj@fedora tmp]$ git config --global init.defaultBranch master
nskarmackiyj@fedora tmp]$ git config --global core.autocrlf input
nskarmackiyj@fedora tmp]$ git config --global core.safecrlf warn
nskarmackiyj@fedora tmp]$ ssh-keygen -t rsa -b 4096
```

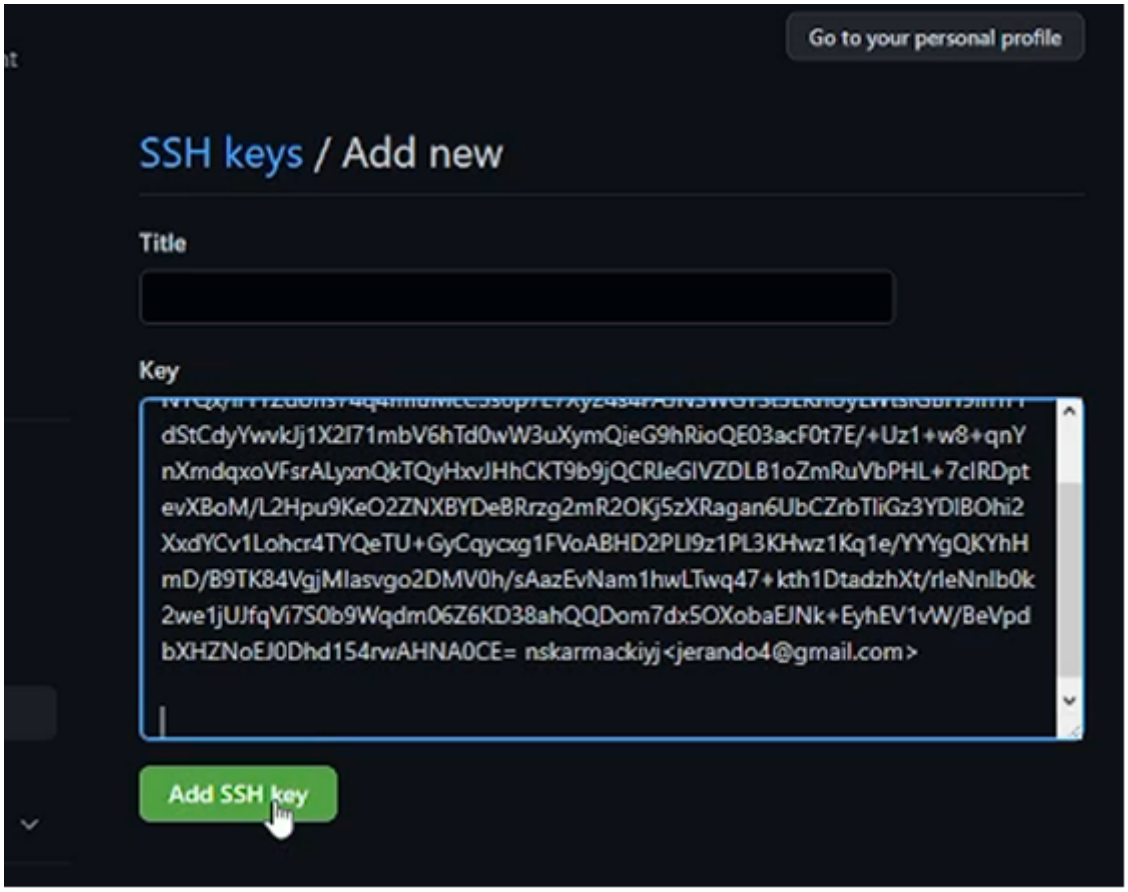
4. Создаем ключи SSH

```
Enter file in which to save the key (/home/nskarmackiyj/.ssh/id_rsa):
/home/nskarmackiyj/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/nskarmackiyj/.ssh/id_rsa
Your public key has been saved in /home/nskarmackiyj/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:RAHPdLh3teIK6qjiXdP2tzQ3YjT+UoDgqXtrTwUfvcQ nskarmackiyj@fedora
The key's randomart image is:
+---[RSA 4096]----+
|       ..+o.     |
|        *..  o . |
|       . *o.. E . |
|        +..+.= o |
|       . S. B.o  |
|      .. . + o.  |
|     o.+ o B.o   |
| . . .o+oo +o= . |
| .o.o..+.oo..o.  |
+----[SHA256]-----+
```

Скриншот: Устанавливаем ключ по алгоритму rsa с ключем размером 4096 бит На скриншоте выбрали все основные параметры ключа



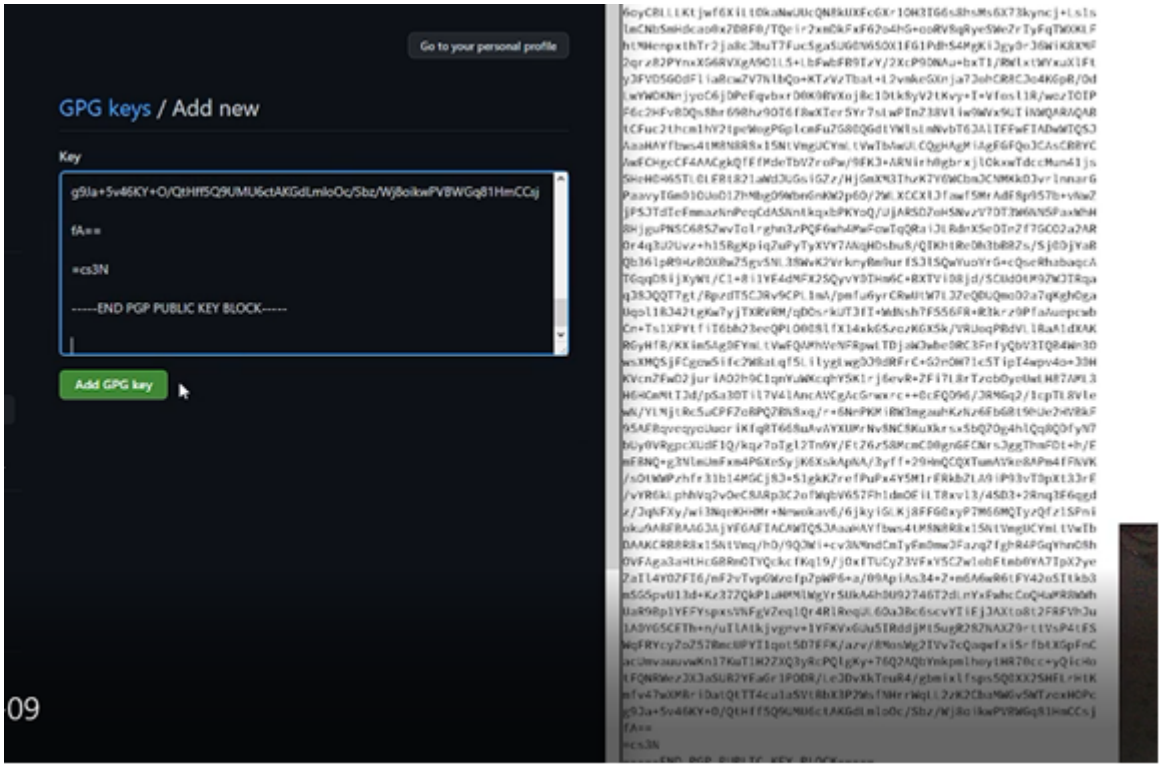
Скриншот: Ключ алгоритма ed25519 И добавляем ключ в гит хаб



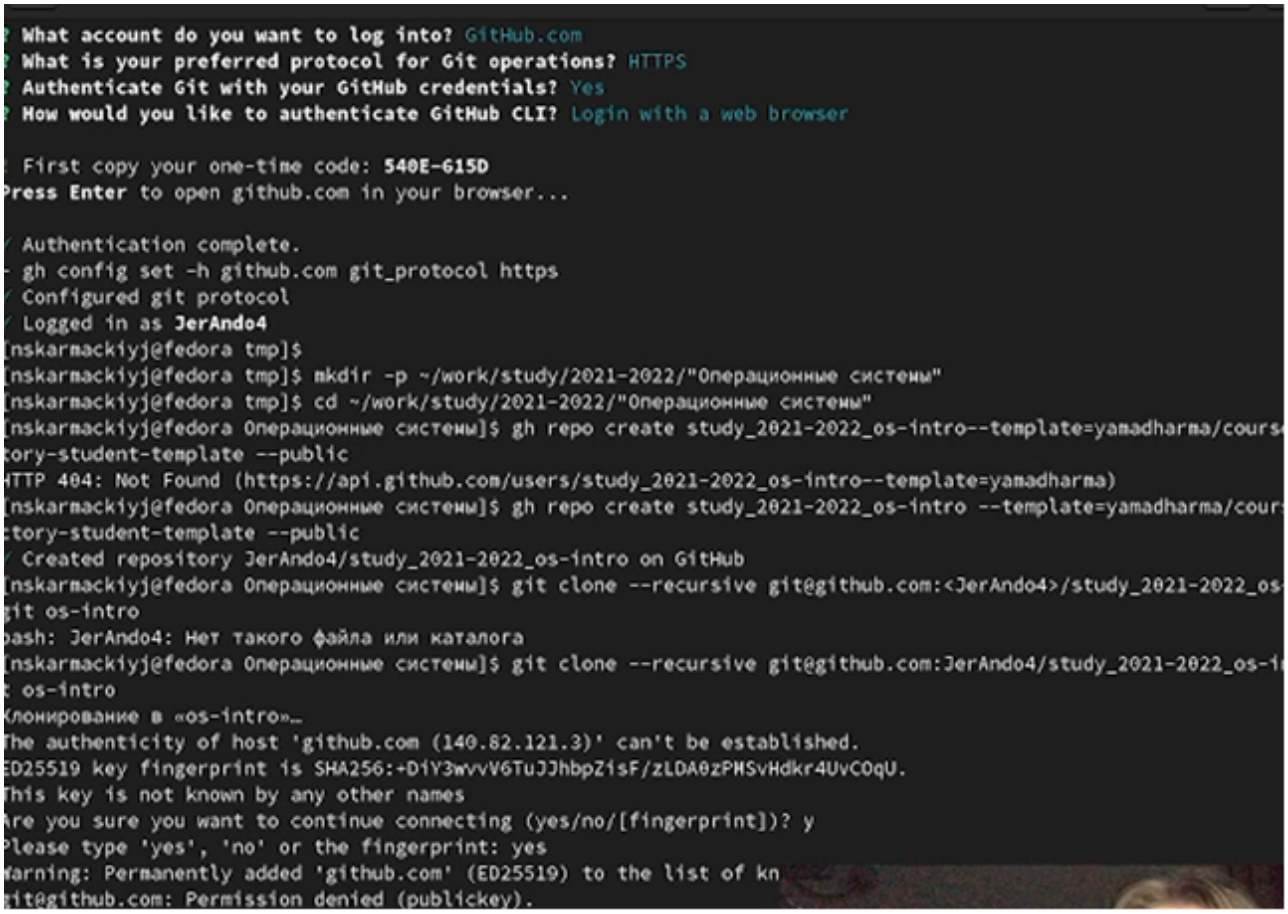
5. Создаем ключ PGP

```
nskarmackiyj@fedora tmp]$ ^[[200~gpg --full-generate-key
^[[201~bash: gpg: command not found...
gpg --full-generate-key[nskarmackiyj@fedora tmp]$ gpg --full-generate-key
gpg (GnuPG) 2.3.2; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Выберите тип ключа:
   (1) RSA and RSA
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
   (9) ECC (sign and encrypt) *default*
  (10) ECC (только для подписи)
  (14) Existing key from card
Ваш выбор? 1
Длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
         0 = не ограничен
      <n>  = срок действия ключа - n дней
      <n>w = срок действия ключа - n недель
      <n>m = срок действия ключа - n месяцев
      <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: nskarmackiyj
Адрес электронной почты: jerando4@gmail.com
Примечание:
Вы выбрали следующий идентификатор пользователя:
    "nskarmackiyj <jerando4@gmail.com>"
```

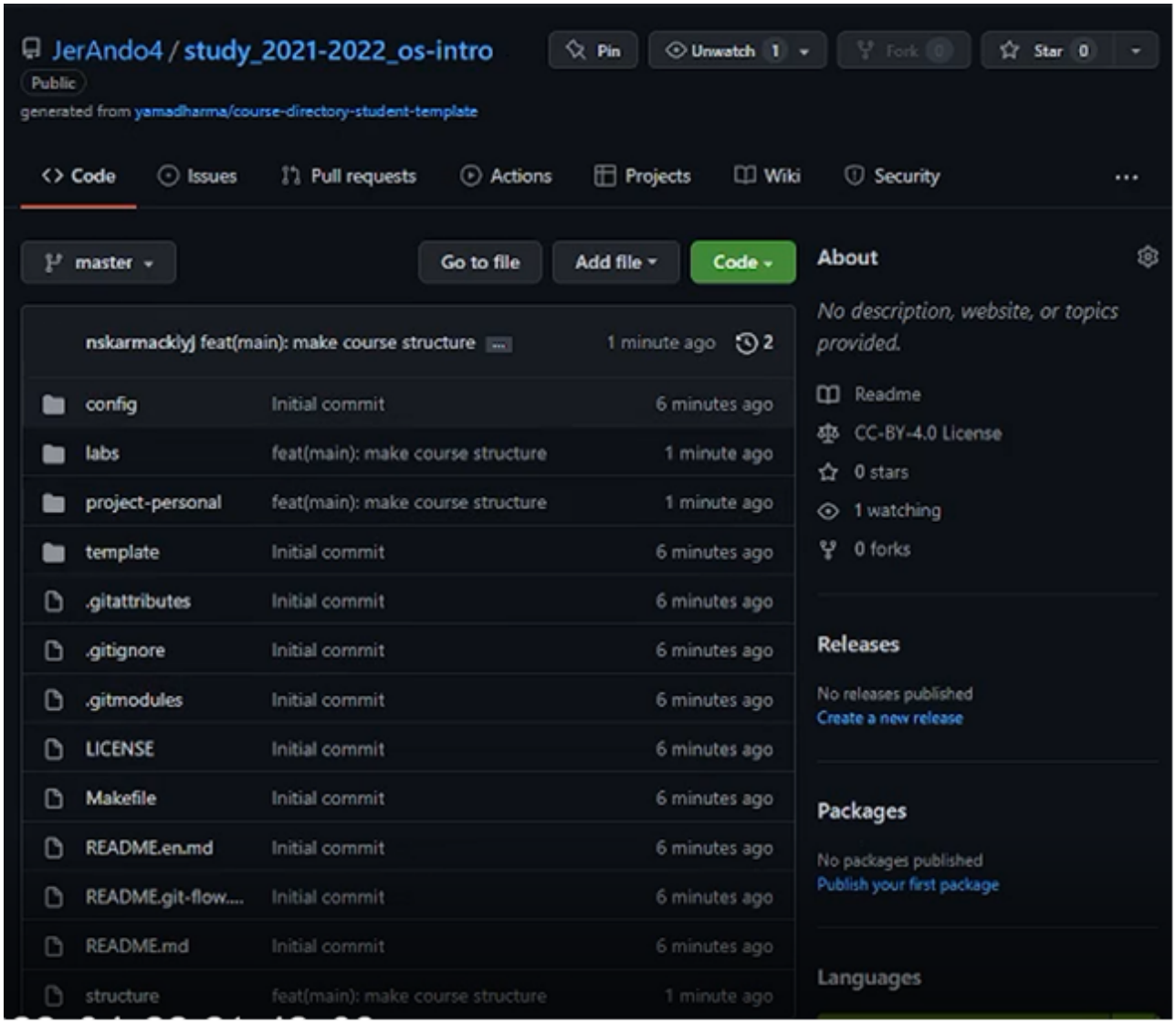
Скриншот:Выбрали все основные параметры ключа Добавляем ключ в Гит Хаб



6. Создаем репозиторию курса на основе шаблона



И смотри как получилось на GitHub



Вывод:

Мы изучили идеологию и применение средств контроля версий.

Контрольные вопросы:

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

Система контроля версий (VCS) — это место хранения кода. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое. Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы.

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

Репозиторий - хранилище версий - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией Commit («[трудовой] вклад», не переводится) — процесс создания новой версии Рабочая копия (working copy) — текущее состояние файлов проекта, основанное на версии, загруженной из хранилища (обычно на последней). Версия (revision), или ревизия, — состояние

всех файлов на определенный момент времени, сохраненное в репозитарии, с дополнительной информацией

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

Централизованные системы — это системы, которые используют архитектуру клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. (Пример — Wikipedia.) В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов. (Пример — Bitcoin)

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

При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, производить откат к любой более ранней версии проекта, если это требуется.

5. Опишите порядок работы с общим хранилищем VCS.

Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения.

6. Каковы основные задачи, решаемые инструментальным средством git?

У Git есть две основные задачи: хранить информацию обо всех изменениях в коде, начиная с самой первой строчки, и обеспечить удобства командной работы над кодом.

7. Назовите и дайте краткую характеристику командам git.

Создание основного дерева репозитория: git init – получение обновлений (изменений) текущего дерева из центрального репозитория: git pull – отправка всех произведённых изменений локального дерева в центральный репозиторий: git push – просмотр списка изменённых файлов в текущей директории: git status – просмотр текущих изменения: git diff – сохранение текущих изменений: – добавить все изменённые и/или созданные файлы и/или каталоги: git add . – добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена_файлов – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена_файлов – сохранение добавленных изменений: – сохранить все добавленные изменения и все изменённые файлы: git commit -am 'Описание коммита' – сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit – создание новой ветки, базирующейся на текущей: git checkout -b имя_ветки – переключение на некоторую ветку: git checkout имя_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_ветки – сл ияние ветки с текущим деревом: git merge —no-ff имя_ветки – удаление ветки: – удаление локальной уже слитой с основным деревом ветки: git branch -d имя_ветки – принудительное удаление локальной ветки: git branch -D имя_ветки – удаление ветки с центрального репозитория: git push origin :имя_ветки

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

‘Git branch’ – это команда для управления ветками в репозитории Git. Ветка – это просто «скользящий» указатель на один из коммитов. Когда мы создаём новые коммиты, указатель ветки автоматически сдвигается вперёд, к вновь созданному коммиту. Ветки используются для разработки одной части функционала изолированно от других. Каждая ветка представляет собой отдельную копию кода проекта. Ветки позволяют одновременно работать над разными версиями проекта. Ветвление («ветка», branch) — один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления). Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом.

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

Игнорируемые файлы обычно представляют собой файлы, специфичные для платформы, или автоматически созданные из сборочных систем. Временно игнорировать изменения в файле можно командой: git update-index —assume-unchanged **< file>**