**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: Балханова Алтана Юрьевна

Группа: НПМбд-03-21

**МОСКВА**

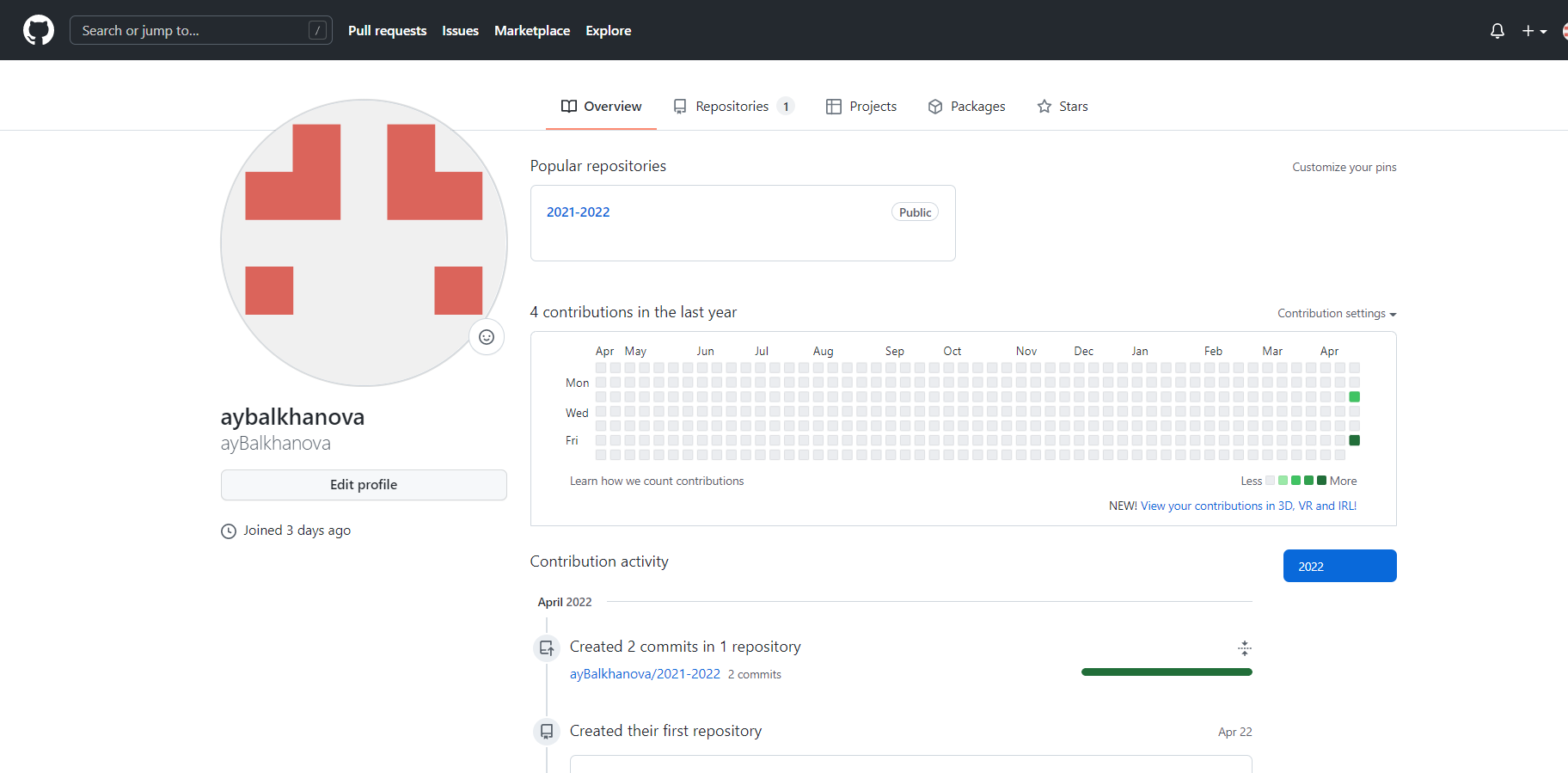
2022 г.

**Цель работы:**

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

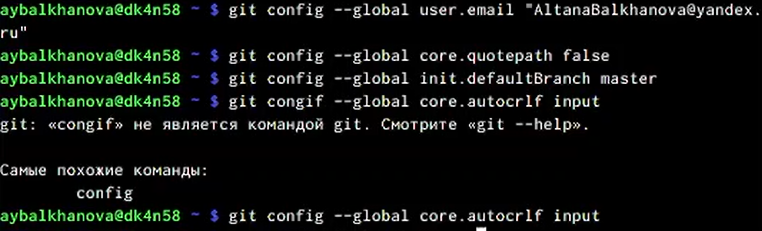
**Ход работы:**

1. Создала учётную запись на github:

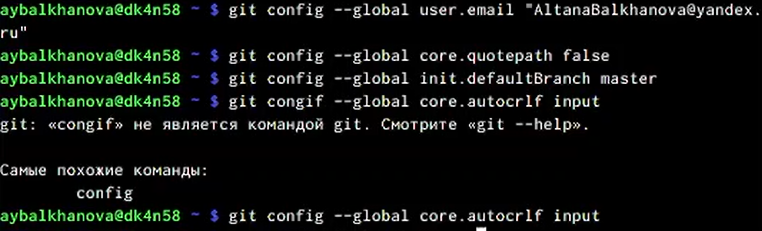


1. Задала имя и email своего репозитория:

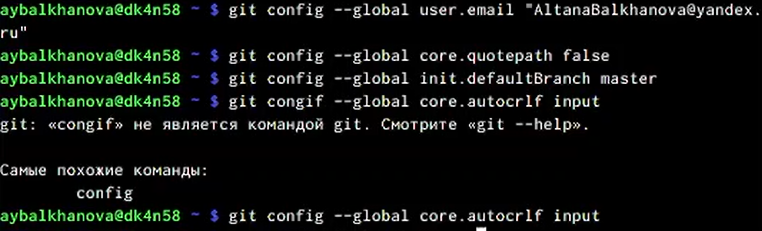




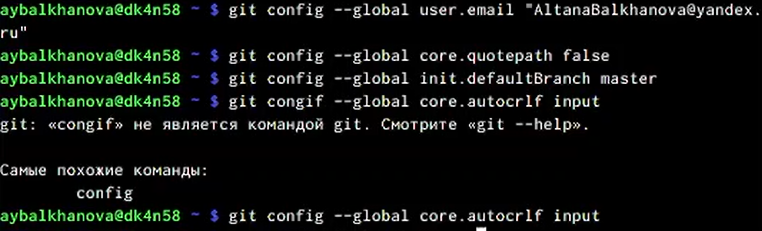
1. Настроила utf-8 в выводе сообщений git:



1. Настроила верификацию и подписание коммитов git, задала имя начальной ветки (будем называть её master):



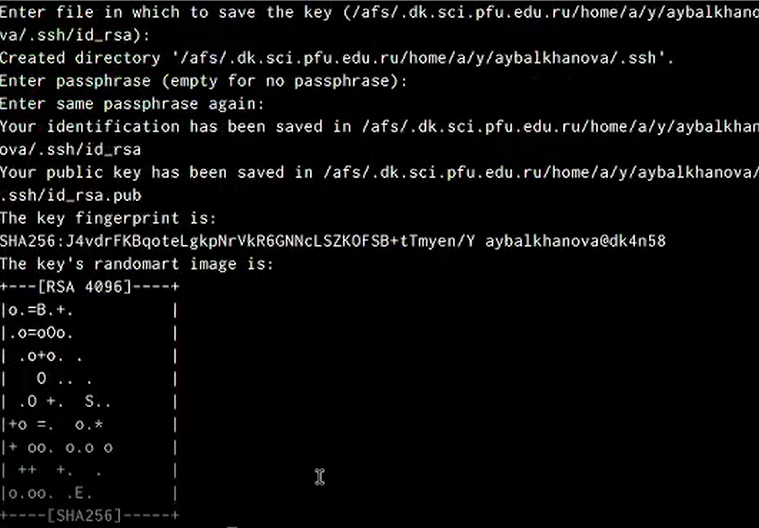
1. Задала параметры autocrlf и safecrlf:





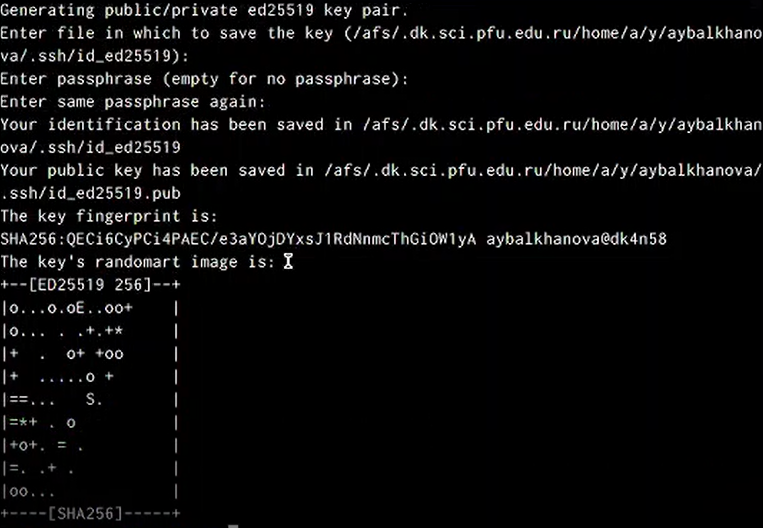
1. Создала ключ ssh – по алгоритму rsa с ключём размером 4096 бит:



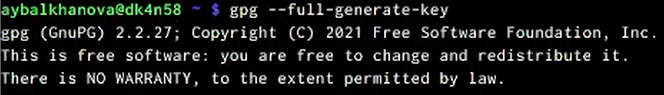


1. Создала ключ ssh – по алгоритму ed25519:

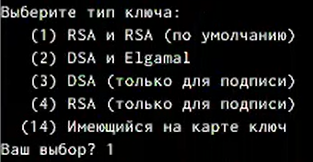




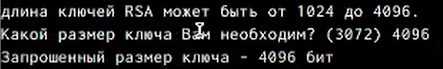
1. Создала ключи pgp:

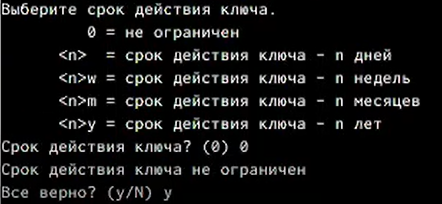


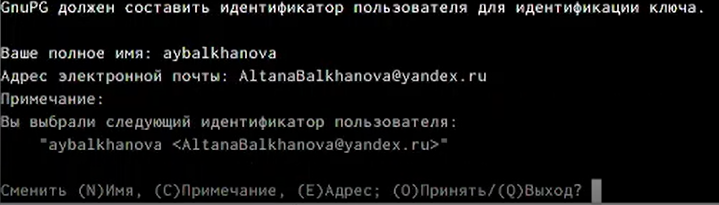
-Выбрала тип ключа RSA и RSA:



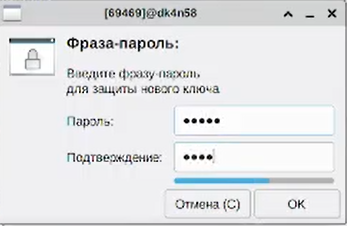
-Выбрала размер ключа 4096:



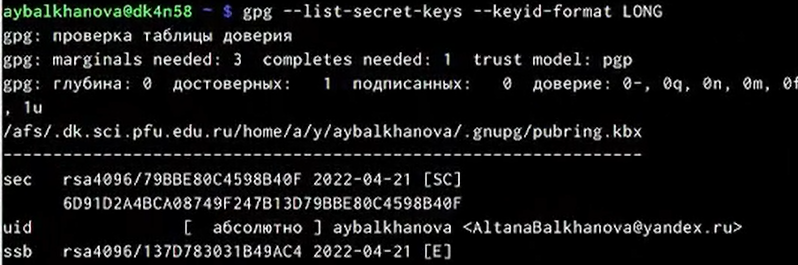
-Выбрала срок действия - неограничен

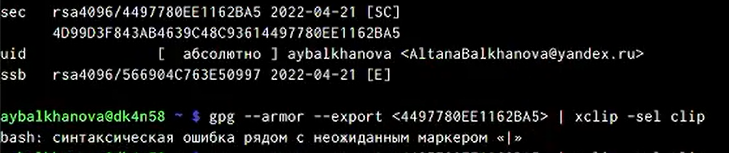
-Для составления идентификатора пользователя я ввела своё имя и адрес электронной почты:

-Ввела фразу-пароль для защиты ключа:

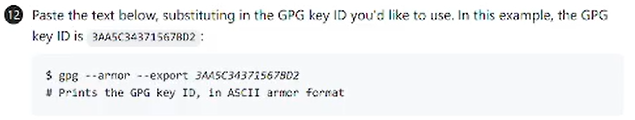


1. Вывела список ключей и пыталась скопировать отпечаток приватного ключа:

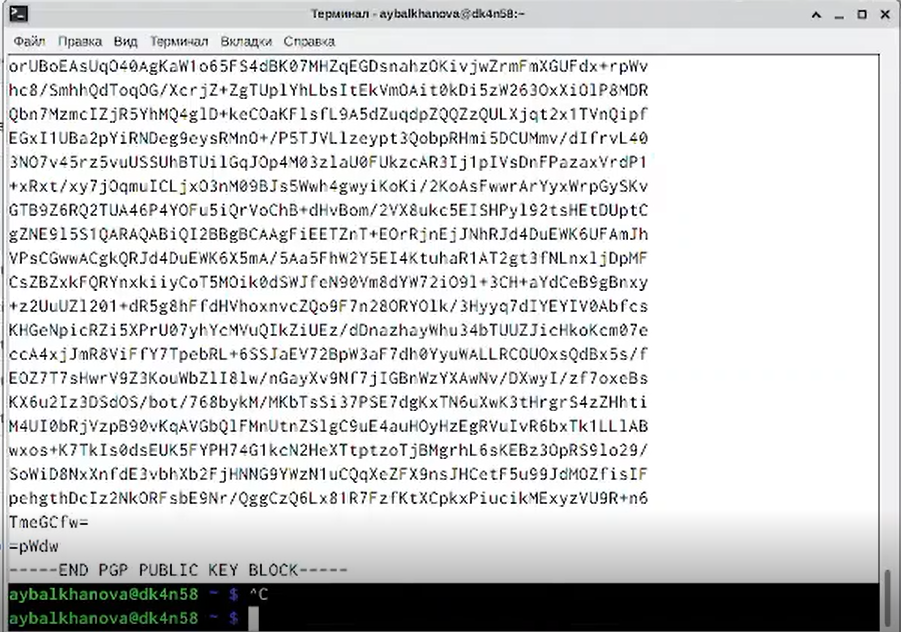


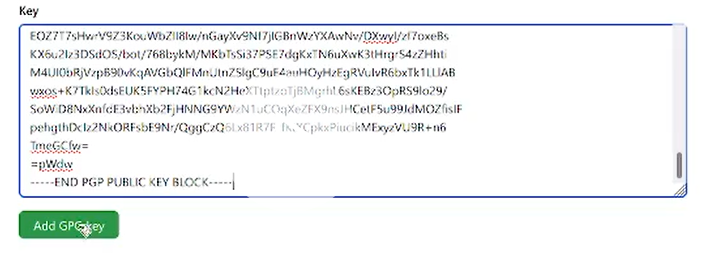


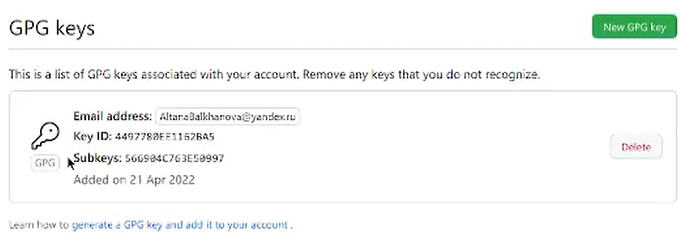
-Но у меня это не получилось. Во время выполнения отчёта я поняла, что проблема была в том, что я не убрала угловые скобки <>, когда писала свой PGP Fingerprint. Но во время выполнения лабораторной работы, я зашла на github, где нашла, как вывести отпечаток ключа:



-Затем, я ввела эту команду, меняя данный пример отпечатка ключа на свой и скопировала выведенный ключ:



-После этого я перешла в настройки GitHub, нажала на кнопку New GPG key и вставила скопированный ключ в поле ввода

-Так он выглядит:

1. Затем я попыталась настроить автоматические подписи коммитов git, но не убрала угловые скобки <>, из-за чего операция не удалась. Это я поняла только во время выполнения отчёта:



**Вывод:**

Я изучила идеологию и применение средств контроля версий, освоила умения по работе с git.

**Контрольный вопросы:**

1. Системы контроля версий (VCS) представляет собойпрограммное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления.
2. Хранилище — это содержимое скрытой папки .git. В этой папке хранятся все версии рабочей области и служебная информация. Этим версиям система автоматически даёт название, состоящее из букв и цифр.

Коммиты — основные конструктивные элементы временной шкалы проекта Git.

Рабочая копия является снимком одной версии проекта.

1. Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Примером является CVS, Subversion.

Децентрализованная система означает, что у каждого разработчика есть личный репозиторий проекта с полным набором всех версий. Примером является Git.

1. Создадим тестовый текстовый файл hello.txt и добавим его в локальный репозиторий:

echo 'hello world' > hello.txt

git add hello.txt

git commit -am 'Новый файл'

1. Для идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

ssh-keygen -C "Имя Фамилия "

Ключи сохраняться в каталоге ~/.ssh/

1. Хранение информации об изменениях в коде. Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.
2. 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

– удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm

– 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

1. Создаём тестовый текстовый файл hello.txt и добавляем его в локальный репозиторий:

echo 'hello world' > hello.txt

git add hello.txt

git commit -am 'Новый файл'

1. Ветка в Git — это простой перемещаемый указатель на один из таких коммитов. Они нужны для того, чтобы разделять код. Например одна ветка у нас может быть основная для разработки. Если мы делаем новый функционал, то мы создаем новую ветку под него, а после окончания работы сливаем то, что мы сделали в основную ветку.

Это дает нам возможность легко откатывать код, если вдруг мы передумаем его сливать в основную ветку, либо делать несколько различных изменений в разных ветках.

1. Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов.