**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:

Ездаков Егор Андреевич

Группа:

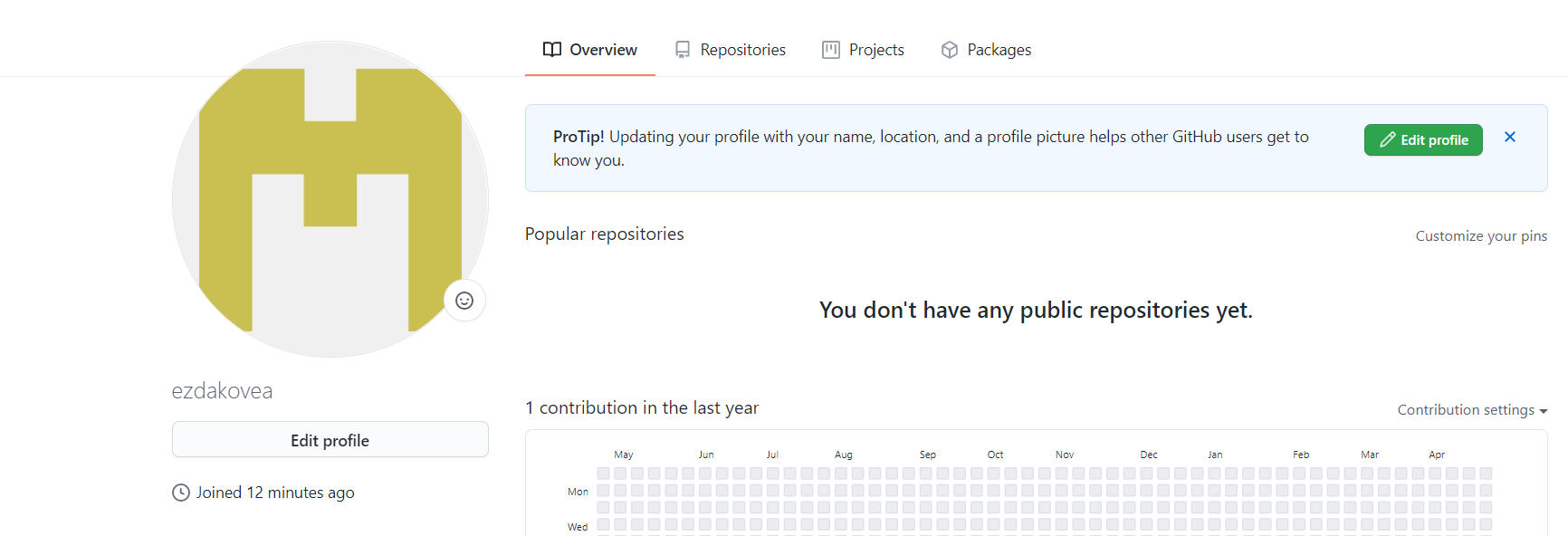
НПМбд-01-20

**МОСКВА**

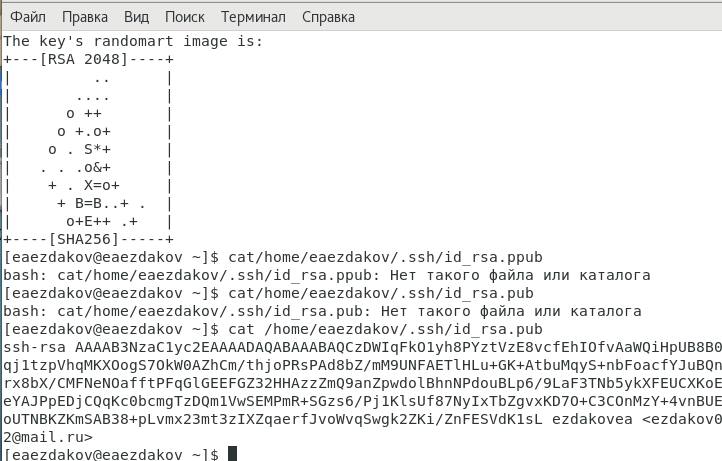
2021 г.

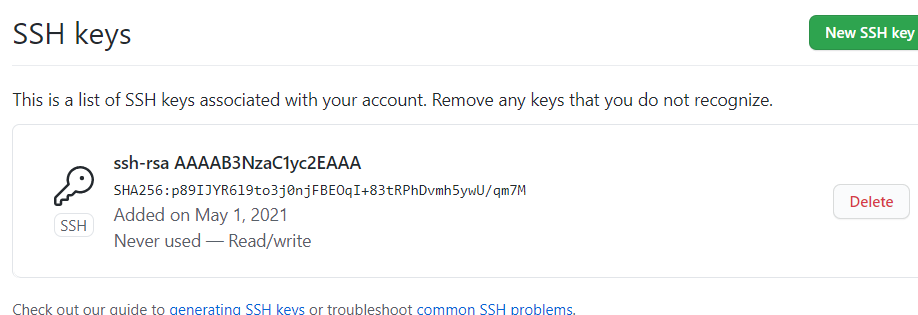
**Цель работы:** изучить идеологию и применение средств контроля версий.

**Ход работы:**

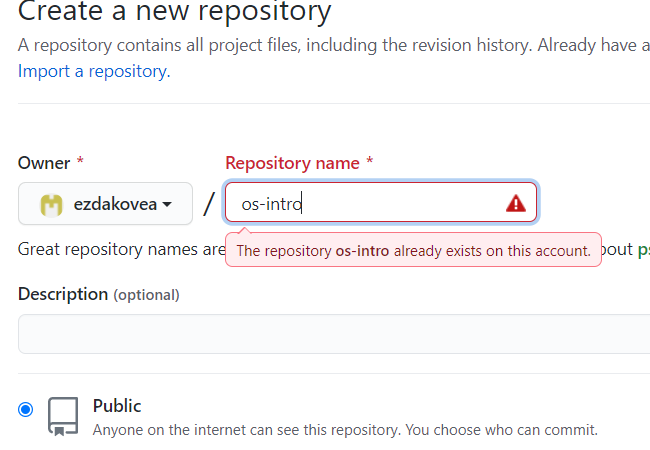
1. Создал учетную запись на GitHub:

2. Настроил систему контроля версий git:

1) Сгенерировал ssh-ключ:

2) Добавил его в настройках учетной записи на GitHub:

3. Создал структуру каталога лабораторных работ с помощью mkdir:

4. Создал репозиторий на GitHub. Назвал его os-intro:

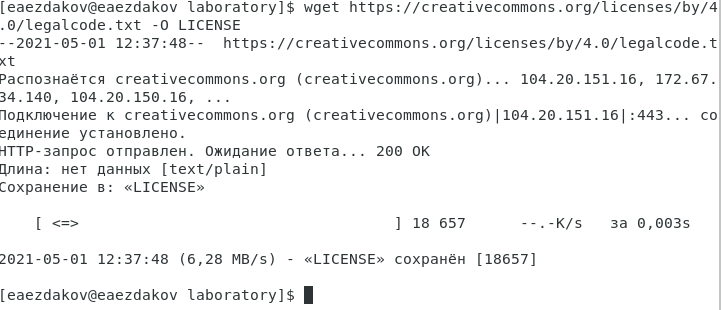
5. Перешёл в каталог laboratory: cd

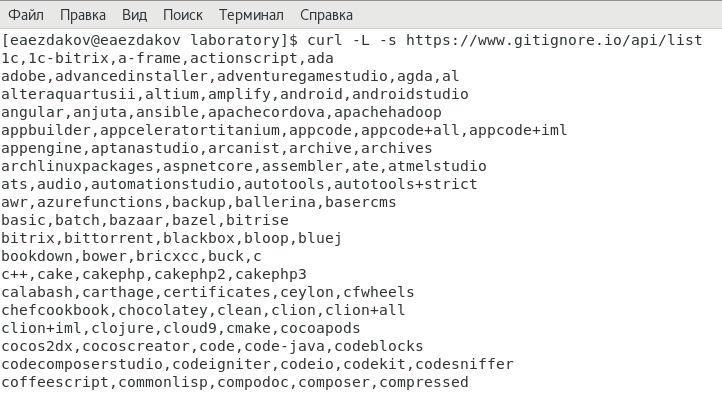
6. Инициализировал системы git:

7. Создал заготовку для файла README.md:

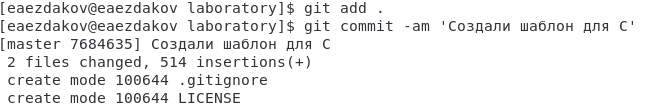


8. Создал первый коммит и отправил его на GitHub:

9. Добавил файл лицензии:

10. Просмотрел список имеющихся шаблонов игнорируемых файлов:

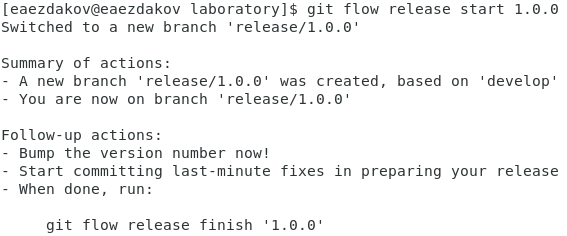
11. Загрузил шаблон для С:

12. Добавил новые файлы, отправил на GutHub:

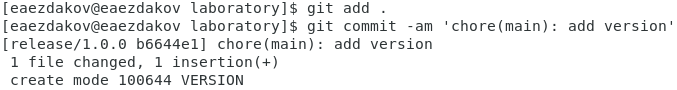
13. Инициализировал git-flow:

14. Префикс для ярлыков установил в v:

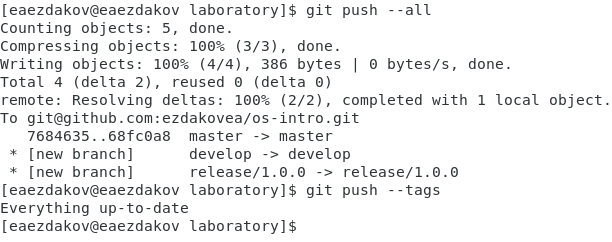


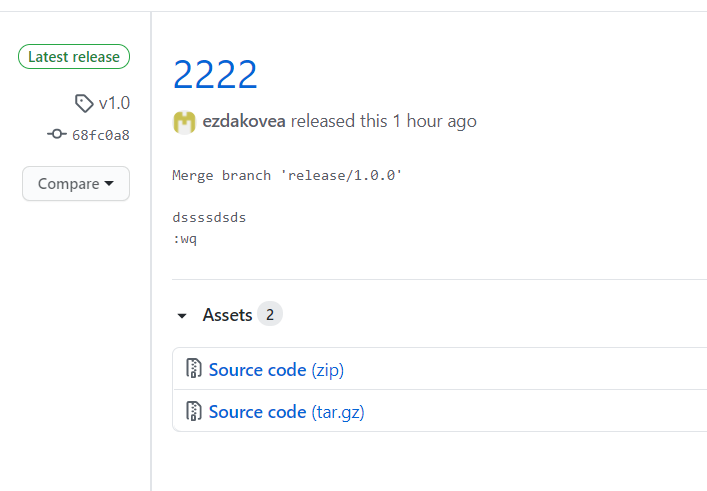
15. Находясь на ветке develop, создал релиз с версией 1.0.0:

16. Записал версию:

17. Добавил в индекс:

18. Залил релизную ветку в основную ветку:

19.Отправил данные на GitHub:

20. Создаем релиз на github. Для этого заходим в «Releases», нажимаем «Создать новый релиз». Заходим в теги и заполняем все поля (создаём теги для версии 1.0.0). После создания тега, автоматически сформируется релиз.

**Вывод:** я изучил идеологию и применение средств контроля версий.

**Ответы на контрольные вопросы:**

1. Система контроля версий (VCS) – ПО, облегчающее реализацию удалённой работы нескольких пользователей над одним проектом.

2. Хранилище – это буквально хранилище для всей информации, которая используется в проекте .Commit – команда, позволяющая сохранить все добавленные изменения. История – сортированный по времени список всех когда-либо внесённых изменений. Рабочая копия – копия удалённого репозитория, расположенная на устройстве пользователя.

3. Централизованная (классическая) VCS представляет собой один репозиторий для хранения файлов. Самые известные примеры: CVS, Subversion. Децентрализованная (распределённая) VCS не предполагает обязательного наличия центрального репозитория, её суть в том, что файлы хранятся не на одном сервере, а на устройствах всех участников проекта. Примеры: Git, Bazaar, Mercurial.

4. При единоличной работе с хранилищем на устройстве создаётся локальный репозиторий, в файлы вносятся необходимые изменения, которые через commit отправляются уже в удалённый репозиторий.

5. При работе с общим хранилищем обычно файлы скачиваются из удалённого репозитория, создаётся новая ветка, на которой в файлы так же вносятся и сохраняются с помощью commit нужные изменения. После этого ветка отправляется в удалённый репозиторий, где сливается с основной веткой.

6. Основные задачи git – обеспечение удобства работы с версиями и хранения информации

7. – Наиболее часто используемые команды git: – создание основного дерева репозитория:git init–получение обновлений текущего дерева из центрального репозитория:git pull–отправка всех произведённых изменений локального дерева в центральный репози-торий:git push–просмотр списка изменённых файлов втекущей директории:git status–просмотртекущих изменения:git diff–сохранениетекущих 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 (приэтомфайл и/илик аталог 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 натекущей: 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 стекущим 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

8. С локальным репозиторием git можно использовать для работы с личными проектами, - система контроля версий позволяет удобно систематизировать файлы и экспериментировать, ведь в случае неудачи можно откатиться на более раннюю версию. С удалённым репозиторием удобно осуществлять групповые работы, - помимо возможного отката на ранние версии здесь можно следить за работой каждого из участников, а значит, эффективнее распределять задачи и корректировать проект.

9. Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом.

10. Некоторые файлы не нужно добавлять в проект, - например, файлы редакторов, остаточные файлы и прочий мусор. С помощью gitignore можно удобно выбрать шаблон игнорирования и автоматически избавить себя от необходимости «чистить» репозиторий от ненужный файлов.