Отчёт по лабораторной работе №2

Управление версиями

Калистратова Ксения Евгеньевна

Содержание

[Цель работы 1](#_Toc71397113)

[Задание 1](#_Toc71397114)

[Выполнение лабораторной работы 1](#_Toc71397115)

[Выводы 8](#_Toc71397116)

[Ответы на контрольные вопросы 8](#_Toc71397117)

Цель работы

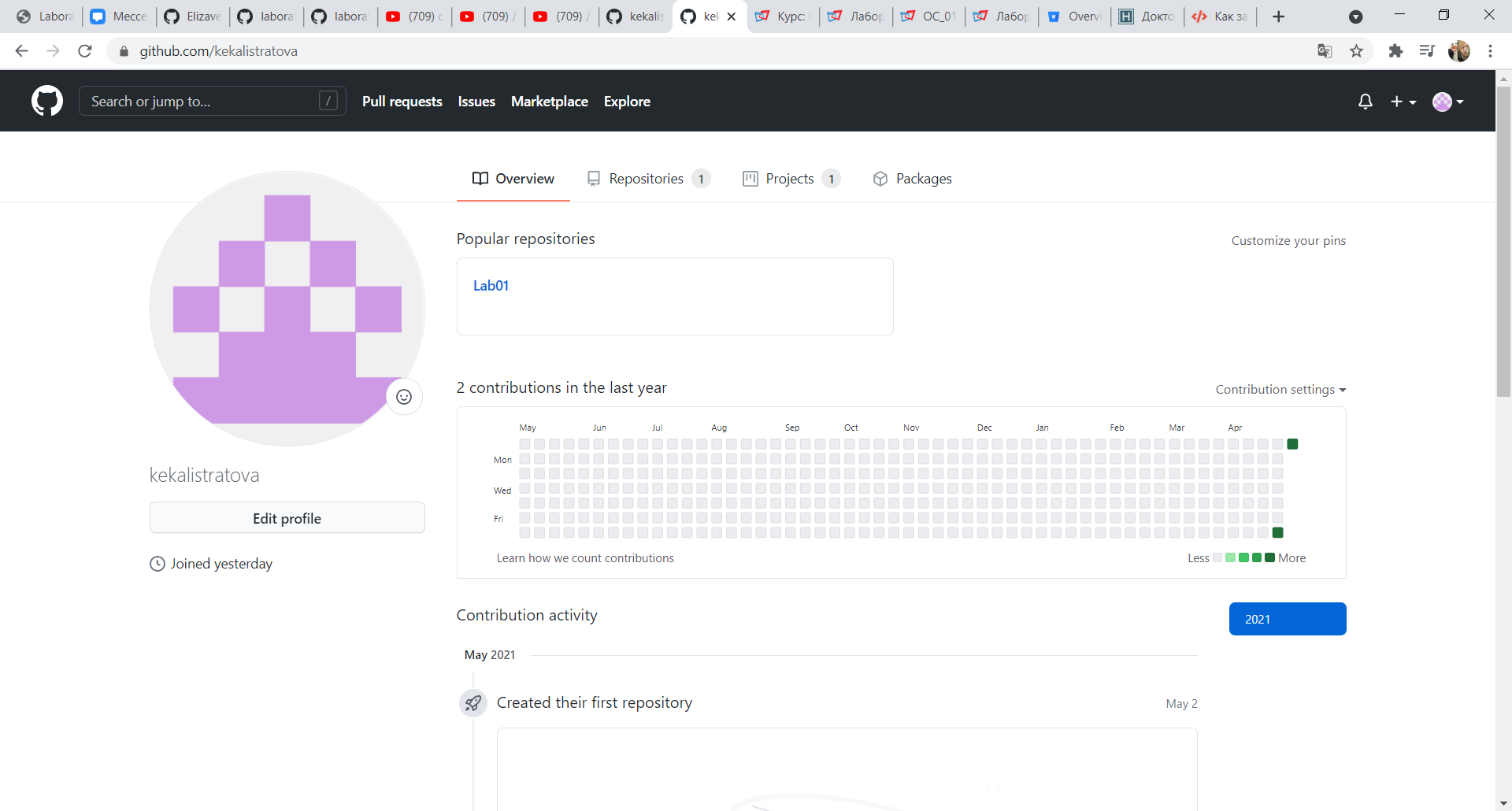
Целью данной работы является изучение идеологии и применения средств контроля версий.

Задание

1. Создать учетную запись на github.com.
2. Настроить репозиторий и организовать доступ по ssh.
3. Изучить механизм управления версиями.

Выполнение лабораторной работы

1. Создаем учетную запись на github.com и репозиторий.



Настраиваю систему контроля версий git. Синхронизирую учётную запись github с компьютером:

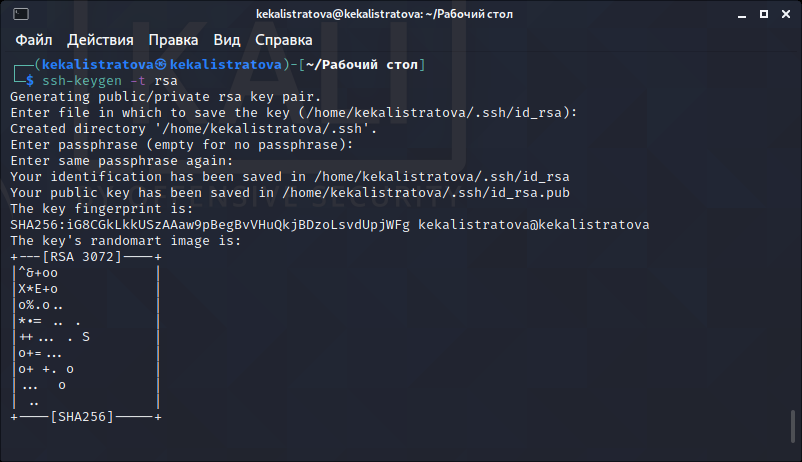
git config --global user.name "kekalistratova"

git config --global user.email "1032201723@pfur.ru"

Далее я открыла терминал и сгенерировала SSH ключ, чтобы связать удаленное файловое хранилище с локальными репозиториями, используя команду

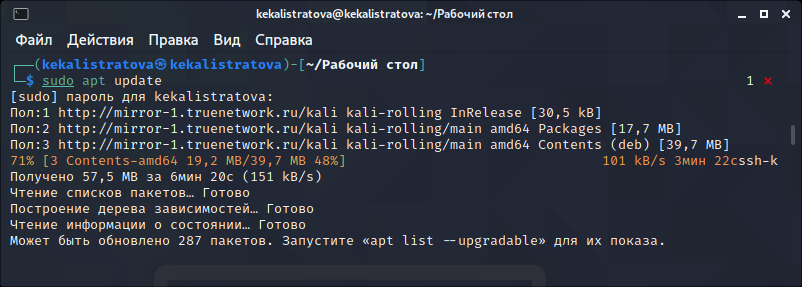
« ssh-keygen -C “kekalistratova <[1032201723@pfur.ru](mailto:1032201723@pfur.ru)>” »

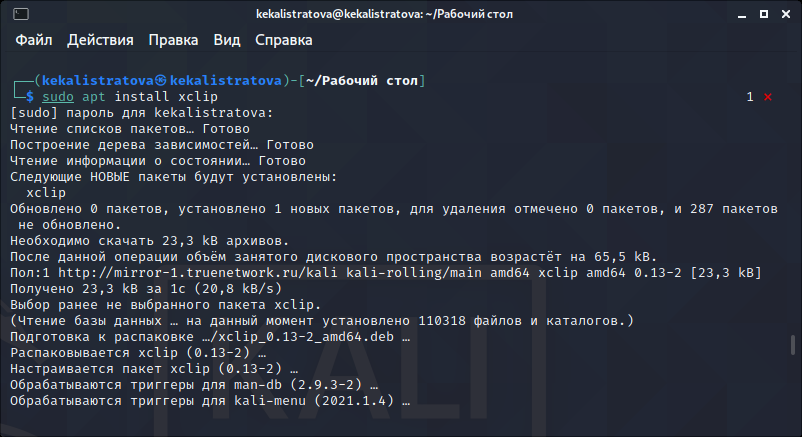
Или ssh-keygen -t rsa

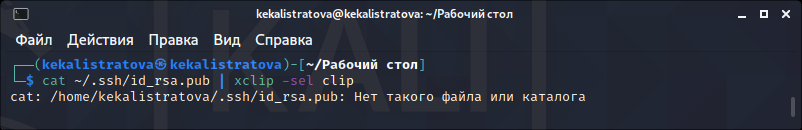


Скопировала данный ключ в буфер обмена (команда

«cat ~/.ssh/id\_rsa.pub | xclip -sel clip»), предварительно установив пакет xclip с помощью команд «sudo apt update» и «sudo apt install xclip»*.*





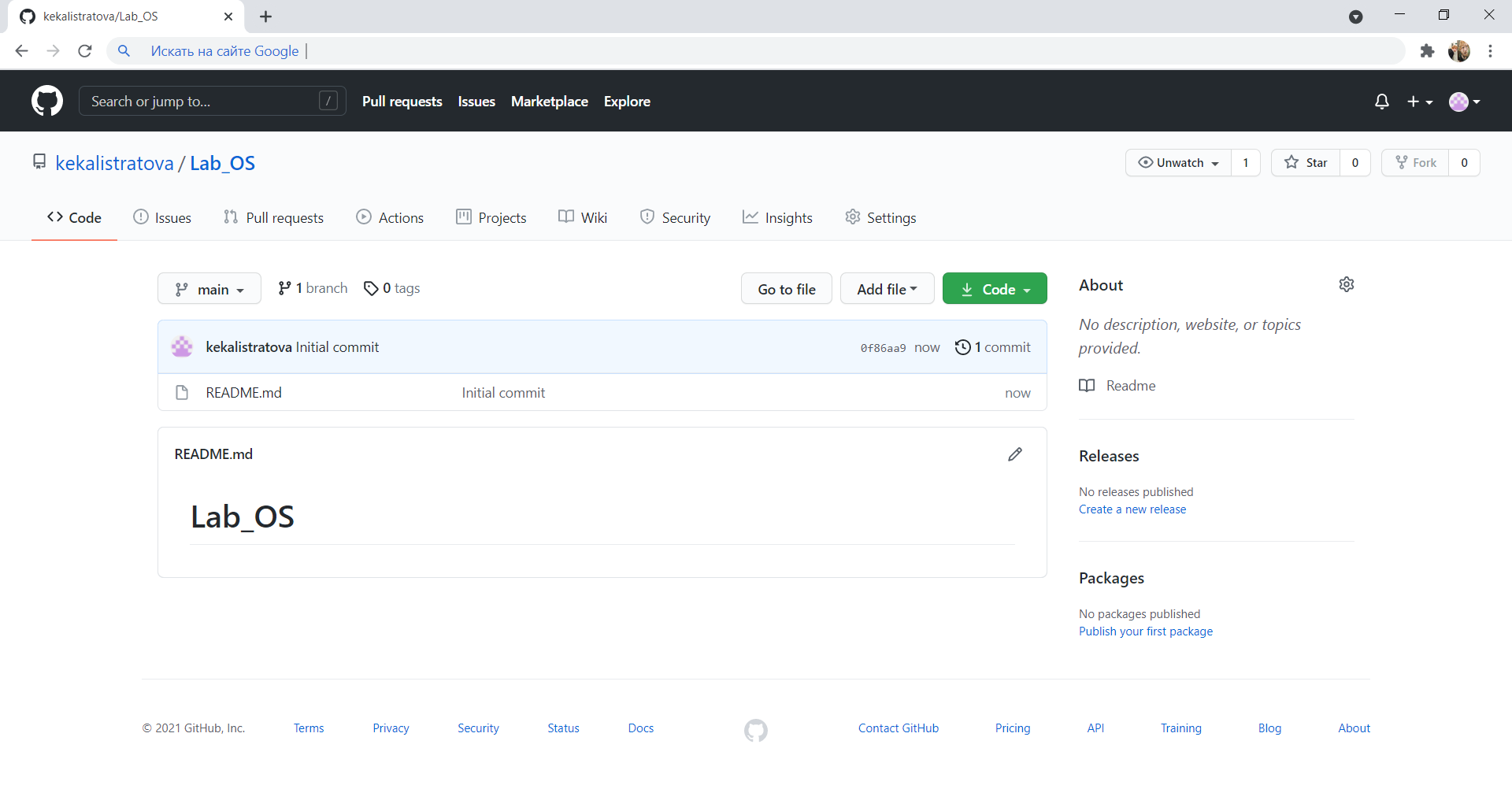


Добавила скопированный SSH ключ в GitHub.

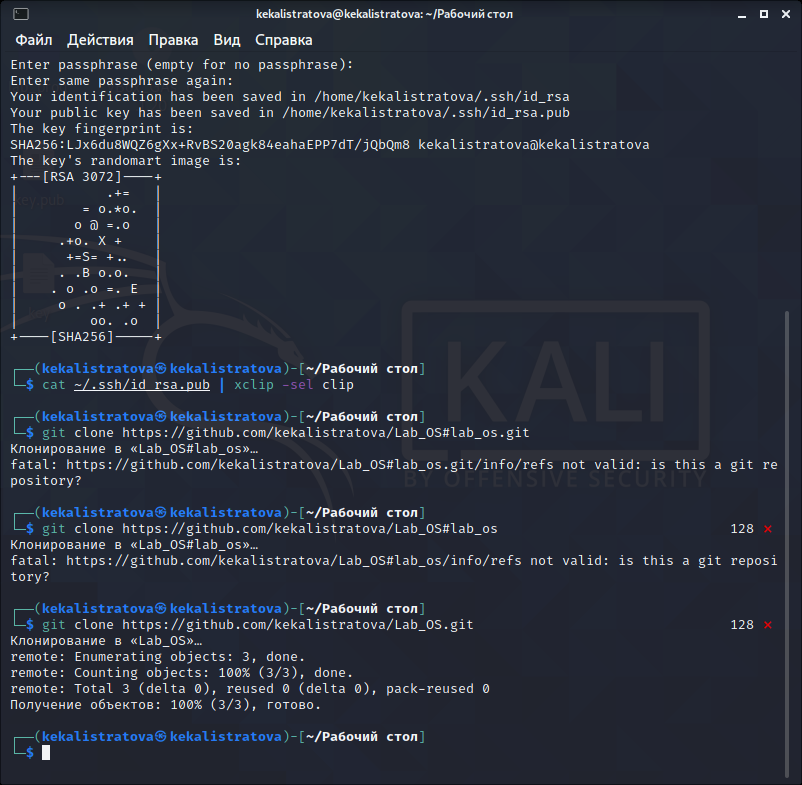
1. **Подключение репозитория к github**

В GitHub захожу в «repositories» и создаю новый репозиторий (имя «lab\_OS», заголовок для файла README). Копируем в консоль ссылку на репозиторий.

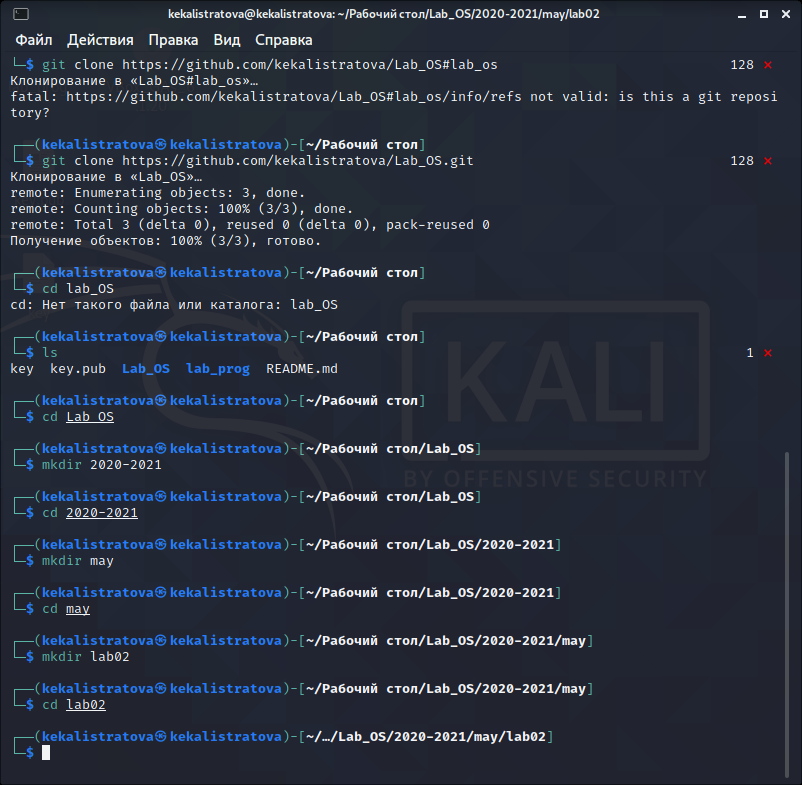
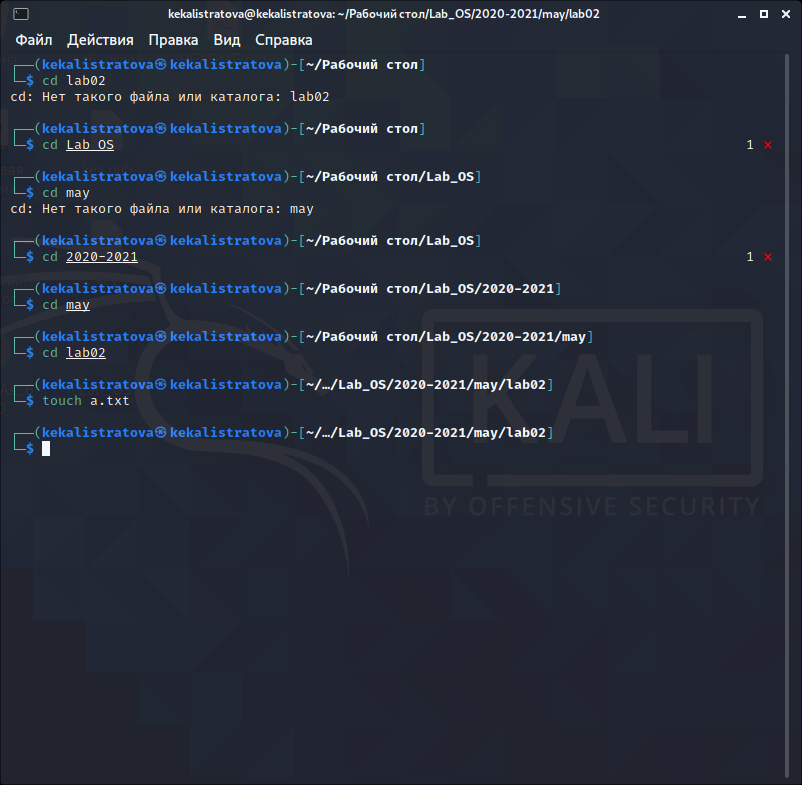
Git clone https://github.com/kekalistratova/Lab\_OS#lab\_os.git



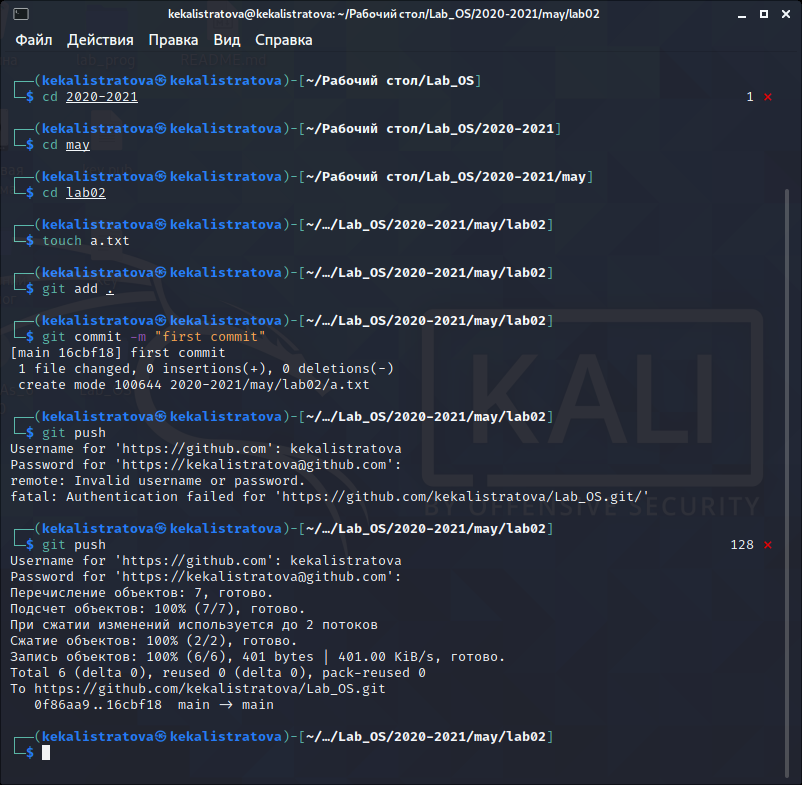
Для создания репозитория я вернулась в терминал и перешла в нужный каталог с помощью команды «cd lab\_OS». Выполнила инициализацию репозитория (создала основное дерево репозитория), используя команду «git init». С помощью команды «ls -a» убедилась в том, что появился скрытый каталог .git



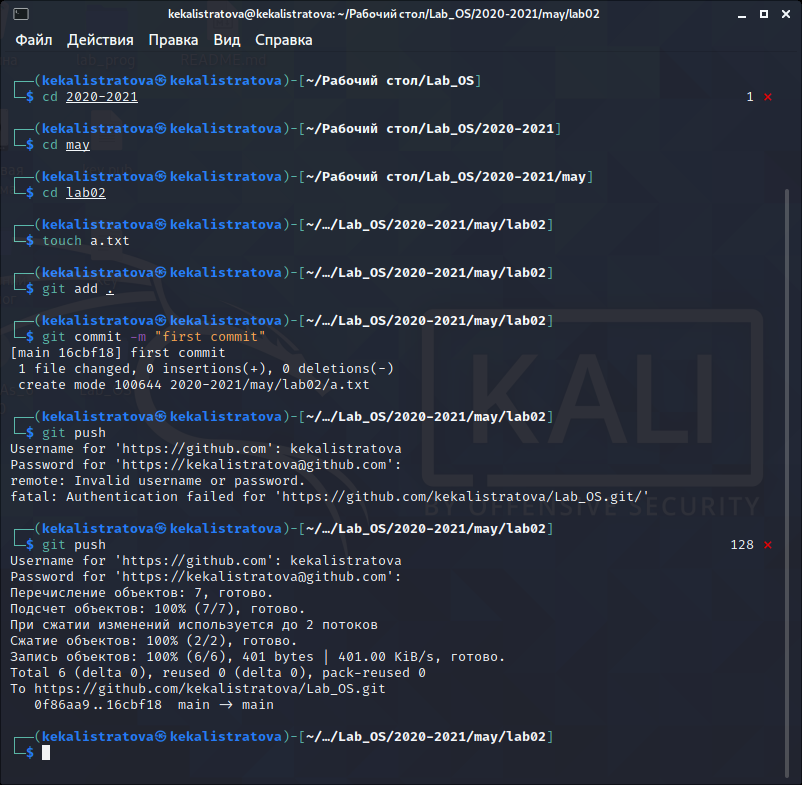
Работаю с каталогом и папками через консоль. Перед тем, как создавать файлы, захожу в репозиторий:

Добавляю первый коммит и выкладываю на githup. Для того, чтобы правильно разместить первый коммит, необходимо добавить команду git add . , далее с помощью команды git commit -m "first commit" выкладываем коммит:

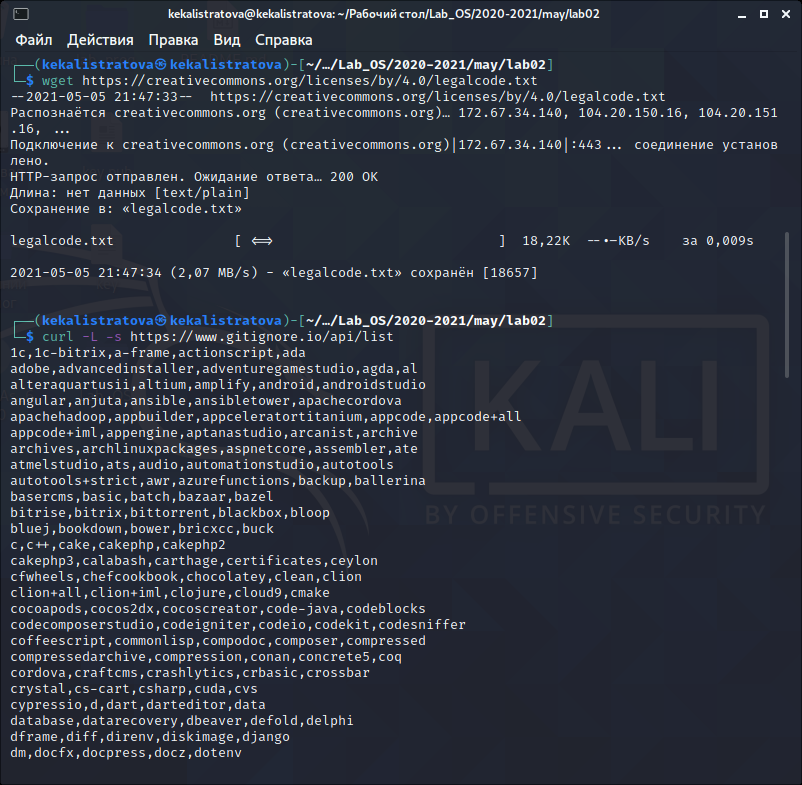


Сохраняю первый коммит (git push):

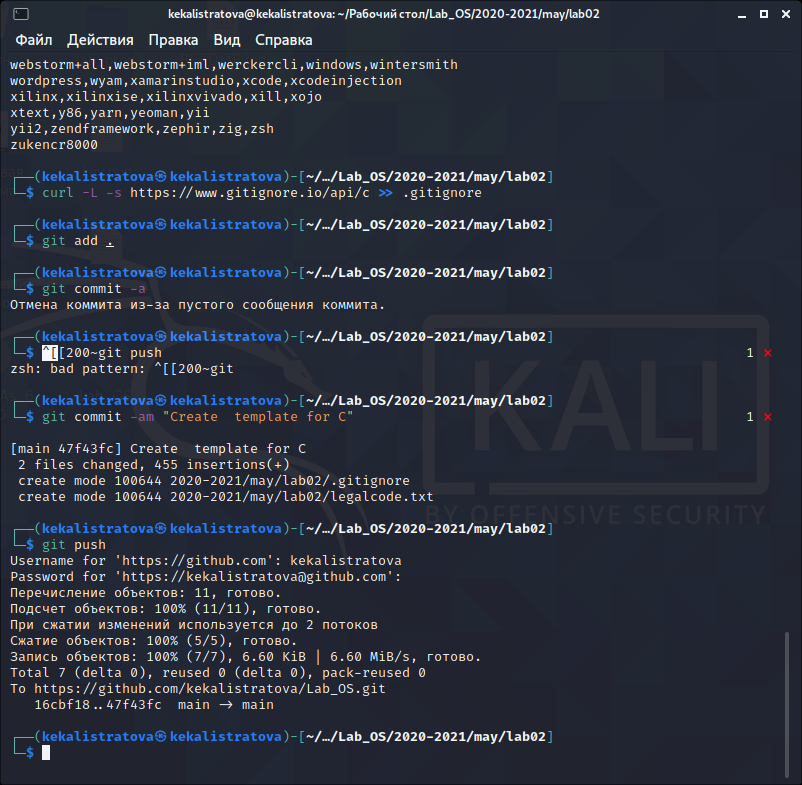


1. **Первичная конфигурация**

Добавляю файл лицензии; Добавляю шаблон игнорируемых файлов. Получаю список имеющихся шаблонов (на скрине представлены не все шаблоны):



Скачиваю шаблон, например, для C. Также добавляю новые файлы и выполняю коммит; Отправляю на github (git push):



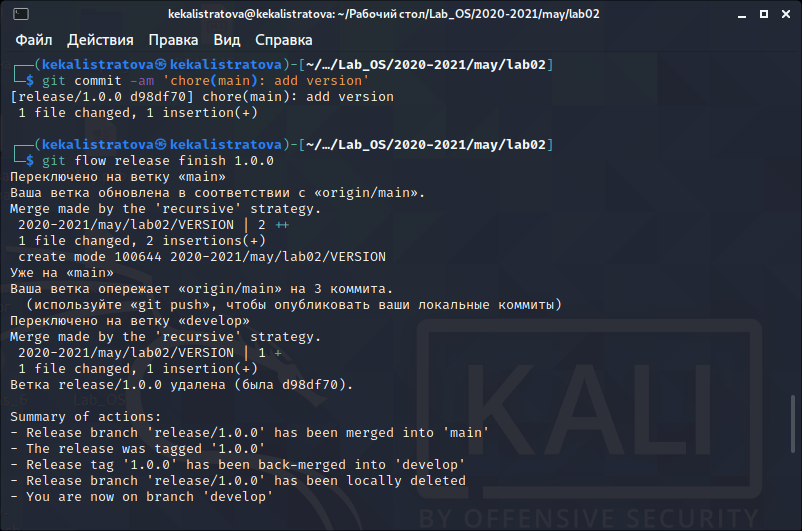
1. **Конфигурация git-flow**

Инициализирую git-flow, используя команду git flow init -f (префикс для ярлыков установлен в v);

Проверяю, что нахожусь на ветке develop (git branch); Создаю релиз с версией 1.0.0;

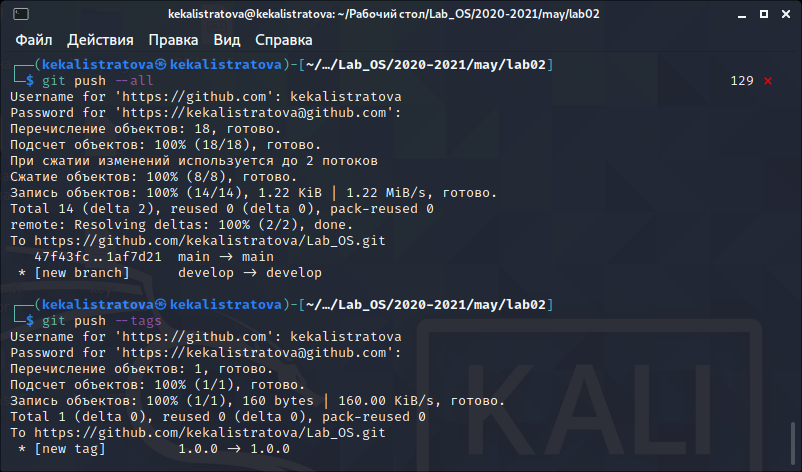
Записываю версию и добавляю в индекс:  
echo "1.0.0" >> VERSION  
git add .  
git commit -am 'chore(main): add version'





Заливаю релизую ветку в основную ветку (команда git flow release finish 1.0.0).

Отправляю данные на github:  
git push --all  
git push –tags

 Создаю релиз на github. Заходим в «Releases», нажимаю «Создать новый релиз». Захожу в теги и заполняю все поля (теги для версии 1.0.0). После создания тега автоматически сформируется релиз.

1. **Контрольные вопросы:**

1) Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом.

2) В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему версию файлов. После внесения изменений

новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять неполную версию изменённых файлов, а производить так называемую дельта-компрессию—сохранять только изменения между последовательными версиями,что позволяет уменьшить объём хранимых данных.

Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например,они могут поддерживать работу с несколькими версиями одного файла,сохраняя общую историю изменений до точки

ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.

3) 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.

В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером.

4) Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

git config --global user.name"Имя Фамилия" git config --global user.email"work@mail"

и настроив utf-8 в выводе сообщений git:

git config --global quotepath false

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:

cd

mkdir tutorial

cd tutorial

git init

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

ssh-keygen -C"Имя Фамилия <work@mail>"

Ключи сохраняться в каталоге~/.ssh/.

Скопировав из локальной консоли ключ в буфер обмена

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

вставляем ключ в появившееся на сайте поле.

6) У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

7) 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 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) Использования git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий):

git add hello.txt

git commit -am' Новый файл

9) Проблемы, которые решают ветки git:

· нужно постоянно создавать архивы с рабочим кодом

· сложно "переключаться" между архивами

· сложно перетаскивать изменения между архивами

· легко что-то напутать или потерять

10) 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. Для этого сначала нужно получить список имеющихся шаблонов: curl -L -s https://www.gitignore.io/api/list

Затем скачать шаблон,например, для C и C++

curl -L -s https://www.gitignore.io/api/c >> .gitignore

curl -L -s https://www.gitignore.io/api/c++ >> .gitignore

**4. Вывод:** В ходе выполнения данной лабораторной работы я изучила идеологию и применение средств контроля версий.