**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

дисциплина: Операционные системы

Выполнил:

Студент группы НПИбд-02-20

Студенческий билет № 1032201738

Колосова Кристина Александровна

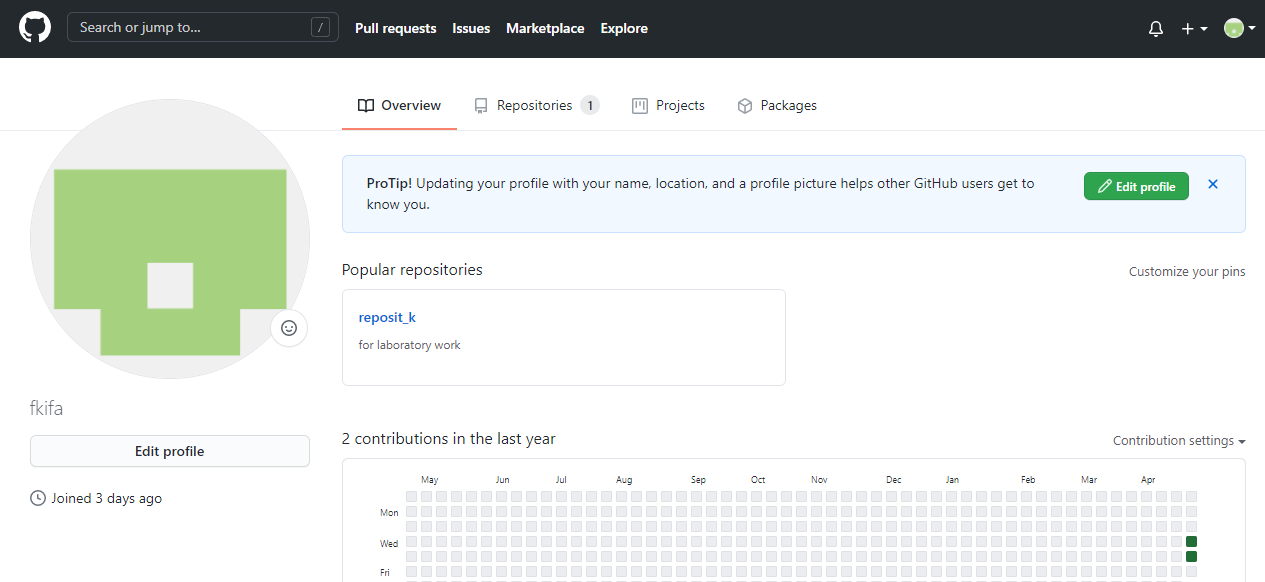
МОСКВА

2020 г.

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

**Ход работы:**

1. Создала учетную запись на github:

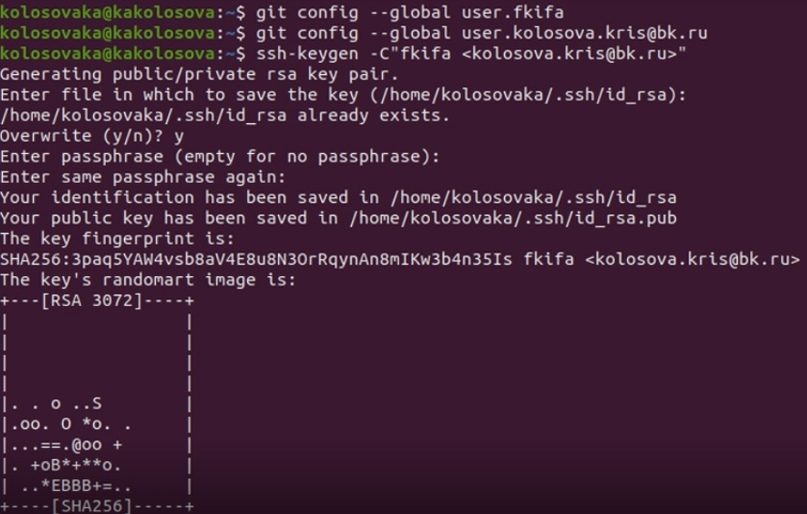


*Рис. 1. Учетная запись*

1. Настроила систему контроля версий git. Нужно синхронизировать учётную запись github с компьютером.

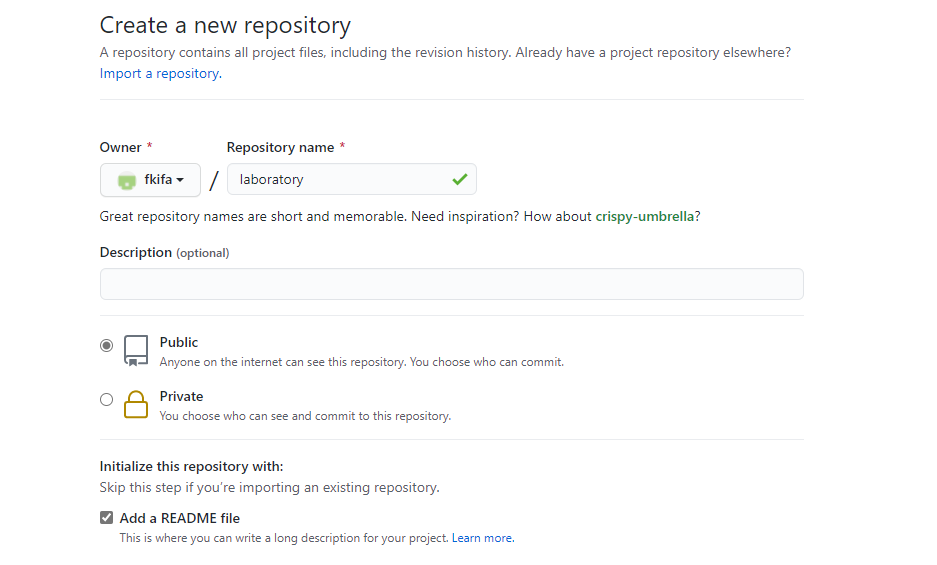
git config --global user.name"Имя Фамилия"

git config --global user.email"work@mail"

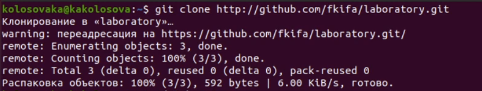


*Рис. 2. Инициализация данных и создание ssh ключа*

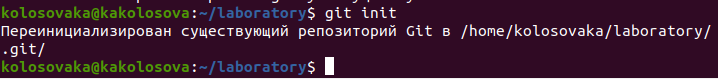
1. Создаю и подключаю репозиторий к github (имя «laboratory», а заголовок для файла README). Копирую в консоль ссылку на репозиторий.



*Рис. 3. Создание репозитория*



*Рис. 4. Клонирование ссылки на репозиторий*

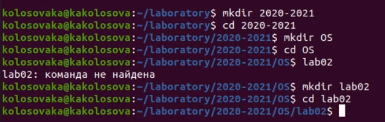
**

*Рис. Инициализируем системы git*

1. Создание структуры каталога:

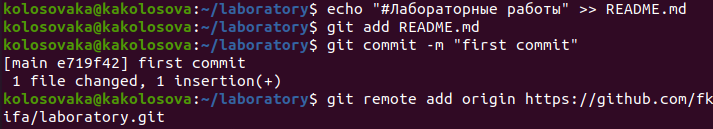


*Рис. 5. Захожу в репозиторий*

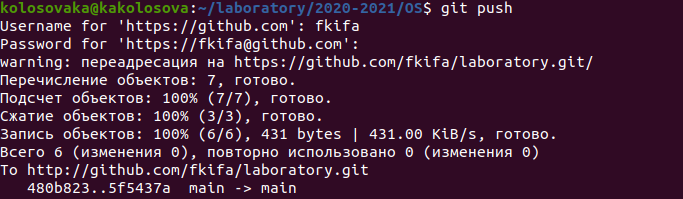
**

*Рис. 6. Создаю файлы*

1. Создаю первый коммит

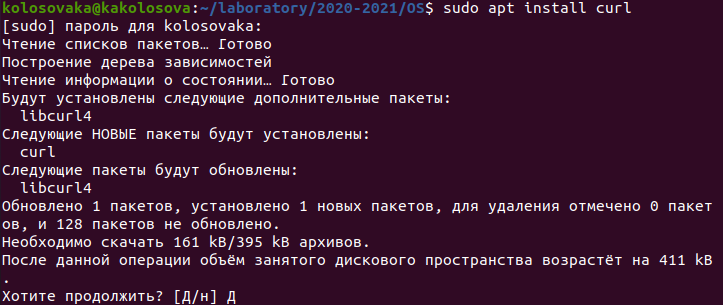


*Рис. 7. Создание коммита*

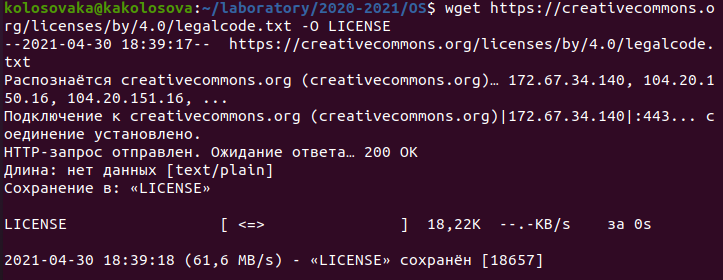
**

*Рис. 8. Сохранение первого коммита*

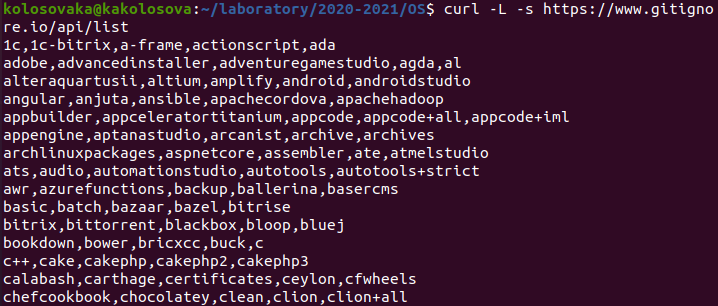
1. Первичная конфигурация



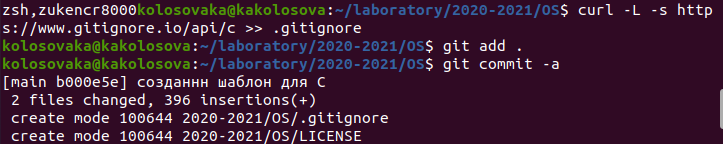
*Рис. 9. Установка команды curl*



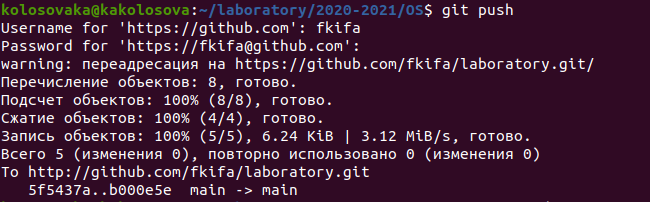
*Рис. 10. Добавление файла лицензии*



*Рис. 11. Добавление шаблона игнорируемых файлов*

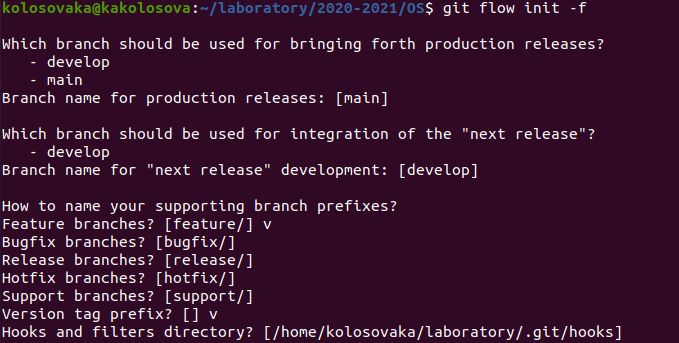
**

*Рис. 12. Добавление шаблона для С*

**

*Рис. 13. Сохранение данных и шаблонов*

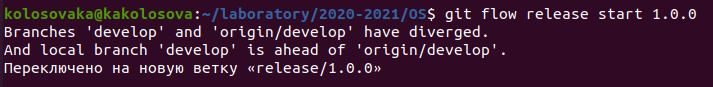
1. Работа с конфигурацией git-flow



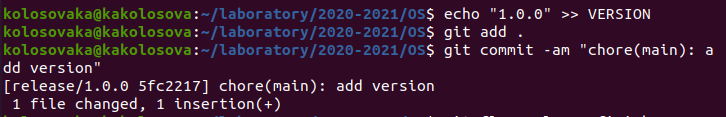
*Рис. 14. Инициализирую git-flow*

**

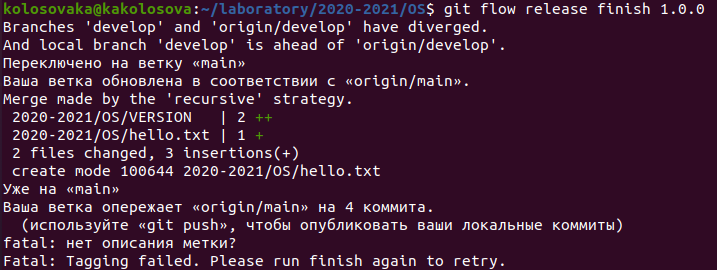
*Рис. 15. Просмотр текущей ветки*

**

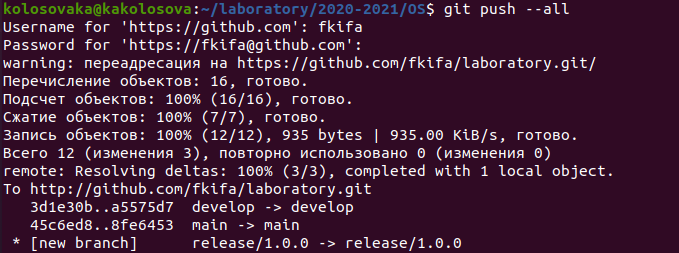
*Рис. 16. Создание релиза с версией 1.0.0*

**

*Рис. 17. Запись версии*

**

*Рис. 18. Слияние релизной ветки с основной*

**

*Рис. 19. Отправка данных на github*

1. Создаем релиз на github

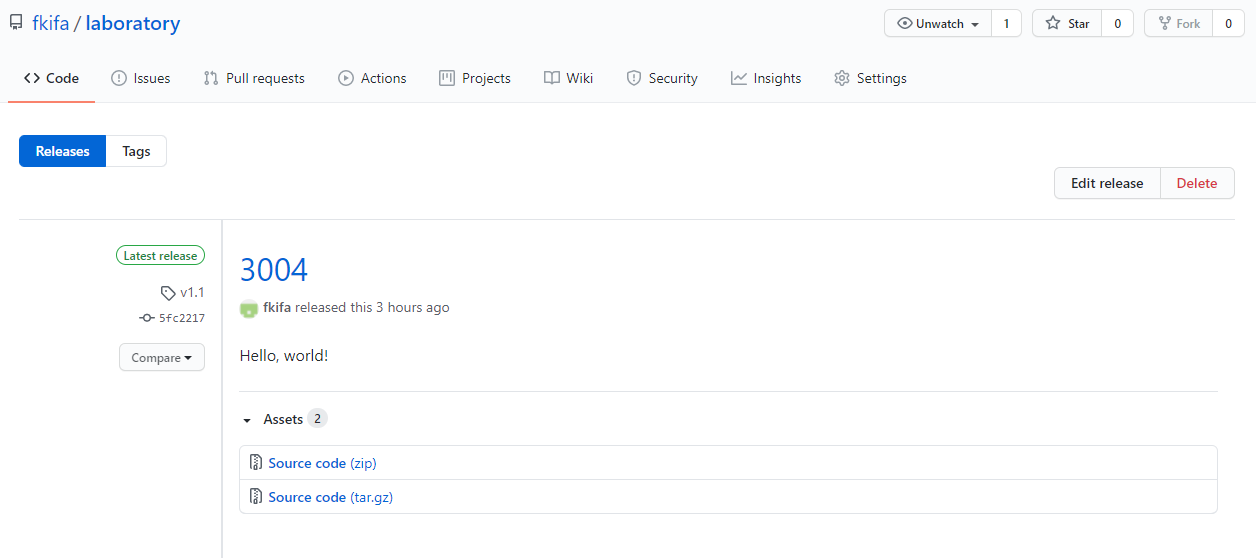


Рис. 20. Релиз

**Вывод:** изучила идеологию и научилась применять средства контроля версий.

**Контрольные вопросы**

1). Система контроля версий Git представляетсобой набор программ командной строки.Доступ к ним можно получить изтерминала посредством ввода командыgitс различ-ными 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 пользователь размещает 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 сообщенийgit:

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 "

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 обо 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 (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). 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

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 списки 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++ >> .gitignorе