Отчёта по лабораторной работе №3

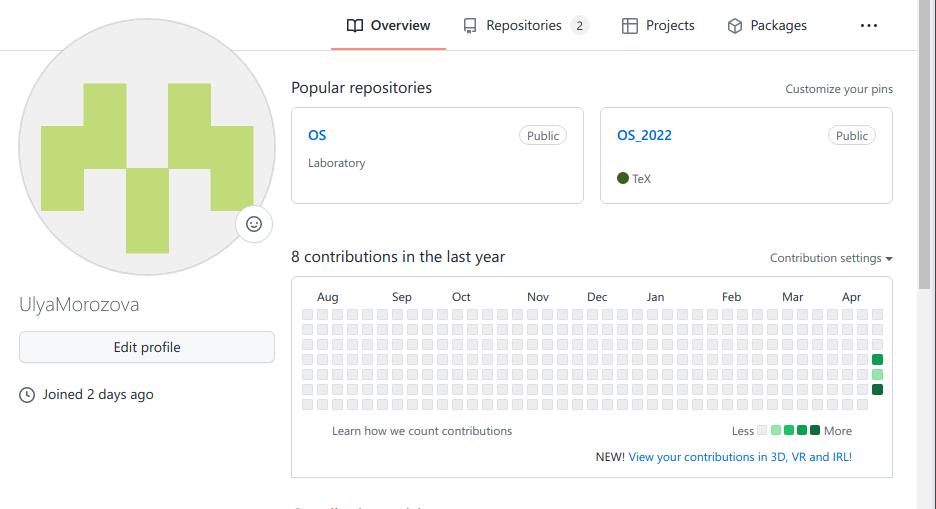
*дисциплина: Операционные системы*

Морозова Ульяна Константиновна

#### **Цель работы**

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

#### **Ход работы**

1. Создаем учетную запись на GitHub и заполним основные данные (рис.1). 
2. Настроим базовую конфигурацию git. Для этого зададим имя и email владельца репозитория (рис.2). Настроим utf-8 в выводе сообщений git. Настроим верификацию и подписание коммитов git: зададим имя начальной ветки (будем называть её master), параметр autocrlf, параметр safecrlf (рис.3). Рис.2

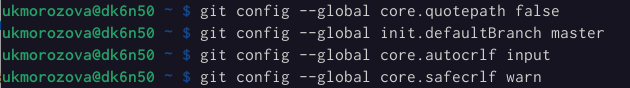


Рис.3

1. Теперь нам надо сгенерировать для ключа ssh и gpg и вставить их в учетную запись github для того, чтобы привязать наш компьютор с github. Создадим ключ ssh с помощью команды (рис.4)

ssh-keygen -t rsa -b 4096

и скопируем его в буфер обмена командой (рис.5)

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

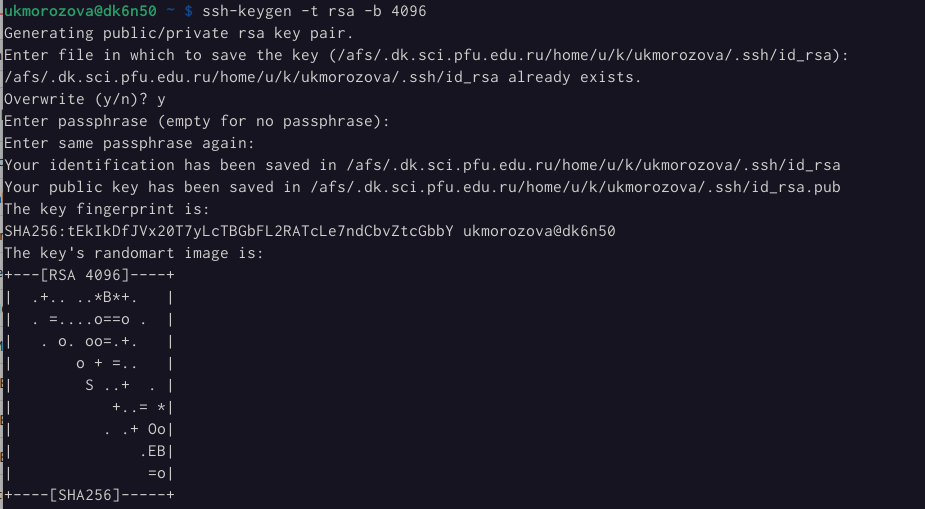


Рис.4

Рис.5

Рис.5

Теперь вставим ключ в аккаунт на GitHub (рис.9). Создадим ключ gpg командой

gpg –full-generate-key

и выбираем из предложенных опций варианты, которые указаны в лабораторной работе (рис.6). Выведем список ключей, чтобы скопировать отпечаток приватного ключа (рис.7).

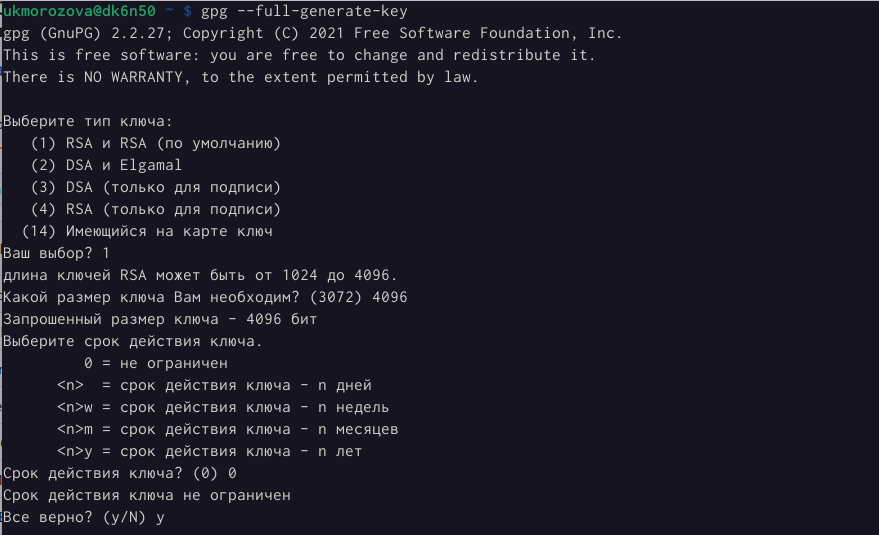


Рис.6

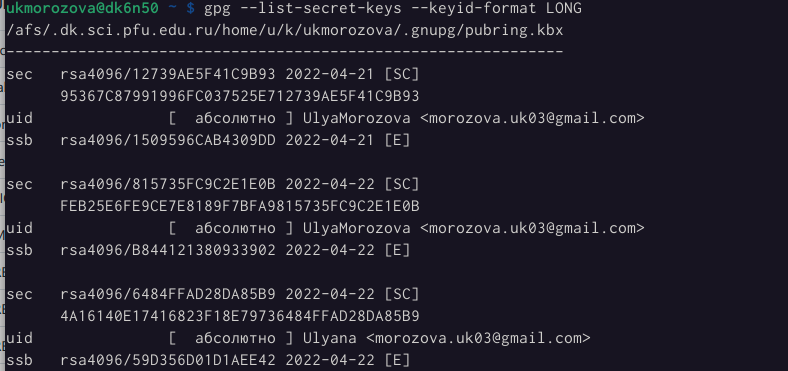


Рис.7

Скопируем сгенерированный PGP ключ в буфер обмена (рис.8):

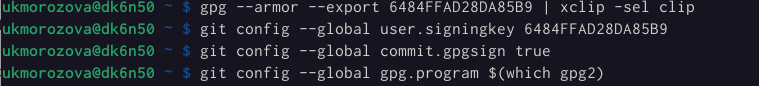


Рис.8

И вставим его на GitHub (рис.9)

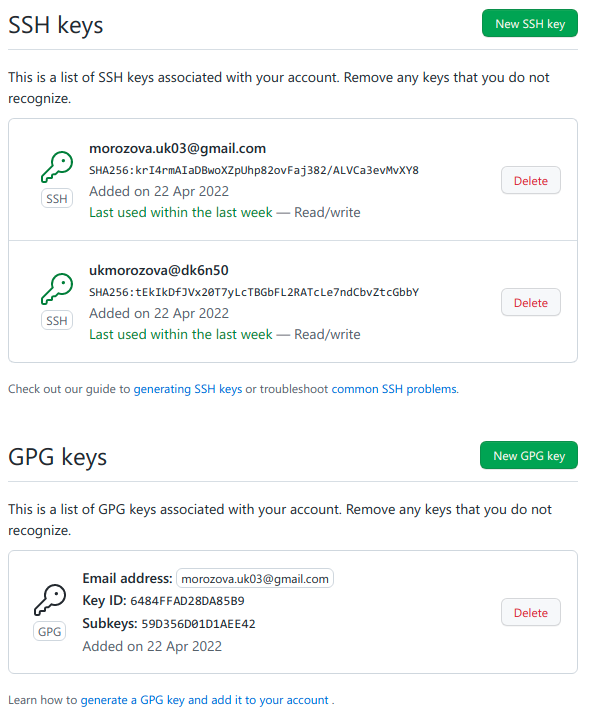


Рис.9

1. Используя введенный email, укажем Git применять его при подписи коммитов (рис.10)

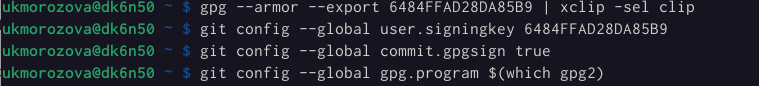


Рис.10

1. Создадим путь, гдe будут храниться материалы к лабораторным работам и перейдем в последнюю папку (рис.11) и скачаем шаблон репозитория (рис.11) в папку.

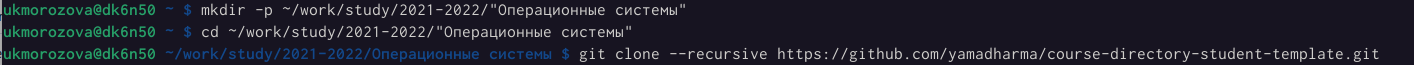


Рис.11

Теперь создадим репозиторий на GitHub, где будут храниться только что созданные папки.

1. Перейдем в каталог курса (команда cd os-intro), удалим лишние файлы (команда rm package.json) и создадим необходимые каталоги для дальнейшей работы (make COURSE=os-intro) (рис.12).

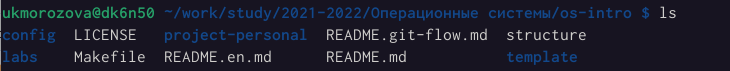


Рис.12

Теперь скопируем наш собственный репозиторий OS\_2022 в папку “Операционные системы”, перенесем в него все файлы из папки os-intro и отправим файлы на сервер, чтобы они также появились в репозитории на GitHub (рис.13-16).

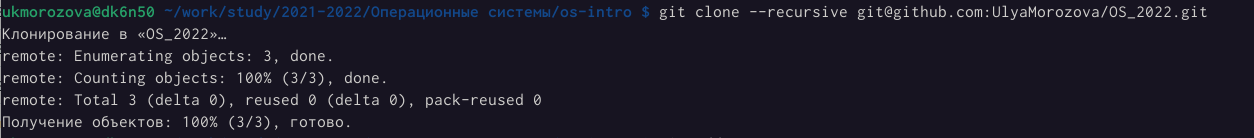


Рис.13

Рис.14

Рис.14

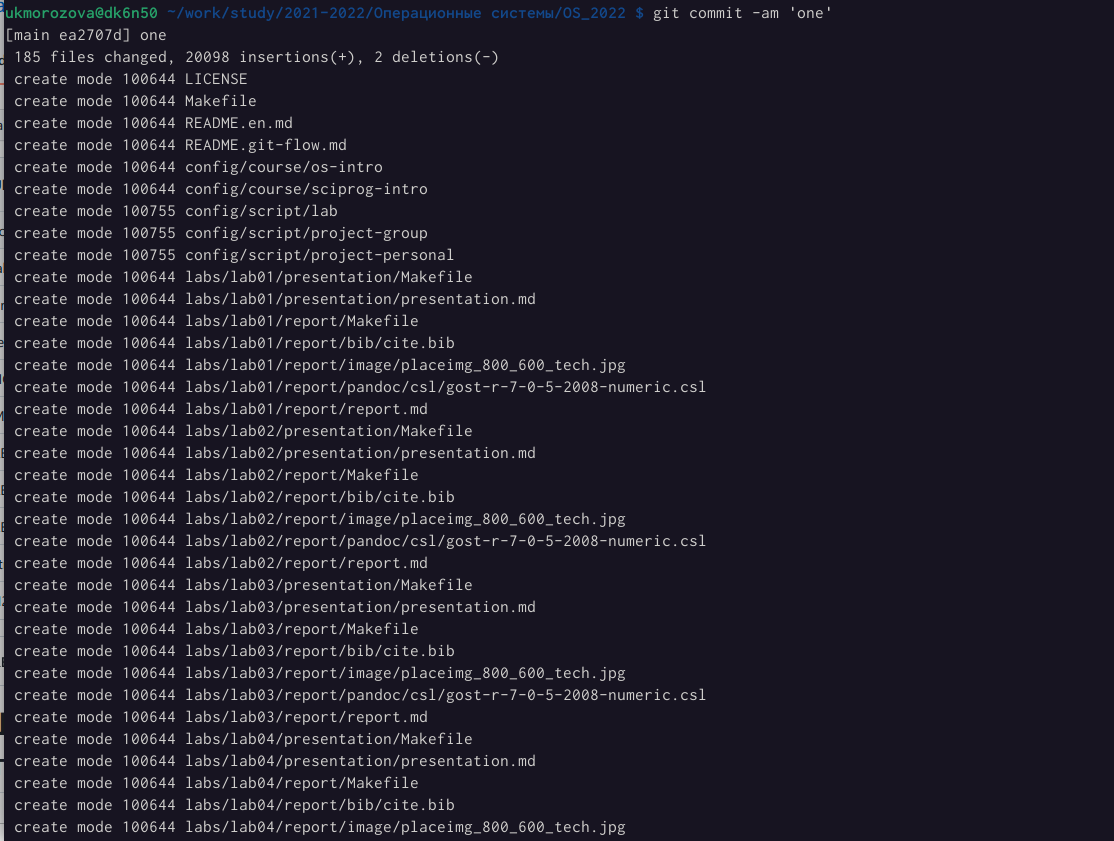


Рис.15

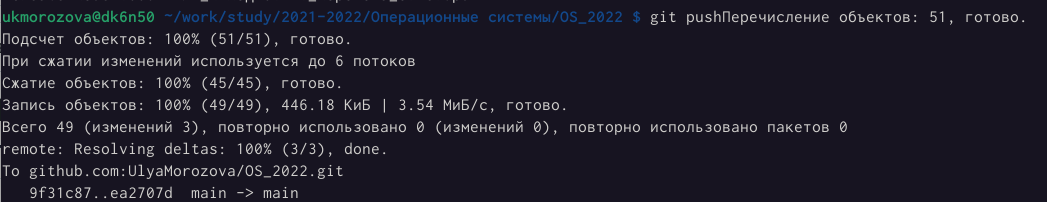


Рис.16

Смотрим репозиторий на сайте GitHub и убеждаемся, что мы сделали все правильнo(рис.17).

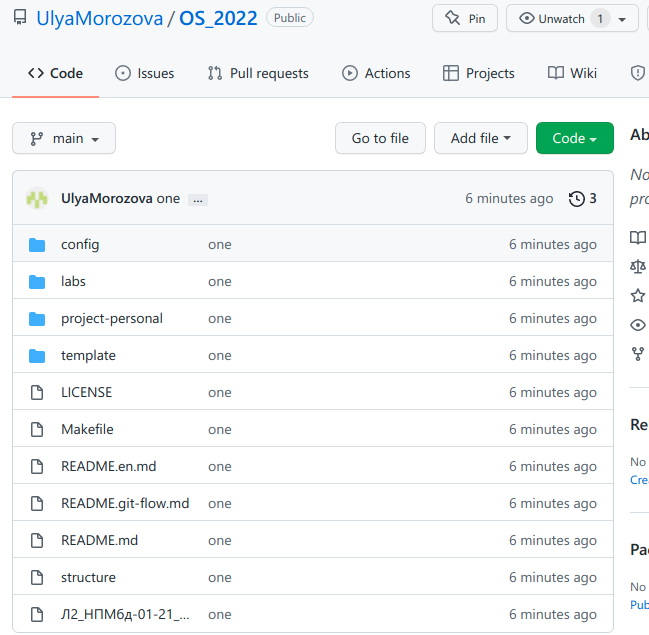


Рис.17

#### Выводы

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

#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. 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.
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 определённыхкоманд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляютсяиз 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,чтопозволяет уменьшить объём хранимых данных. Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например,они могут поддерживать работу с нескольки-ми версиями одного файла,сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Крометого, 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.
4. 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 не полную версию изменённых файлов, а производить так называемую дельтакомпрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.
5. 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.
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 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 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’

1. Проблемы, которые решают ветки git:
   1. нужно постоянно создавать архивы с рабочим кодом
   2. сложно “переключаться” между архивами
   3. сложно перетаскивать изменения между архивами
   4. легко что-то напутать или потерять
2. 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.