Лабораторная работа №2

Markdown

Рытов Алексей Константинович

# 1 **Цель работы**

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

# 2 Задание

– Сделайте отчёт по предыдущей лабораторной работе в формате Markdown.

– В качестве отчёта просьба предоставить отчёты в 3 форматах: pdf, docx и md (в архиве, поскольку он должен содержать скриншоты, Makefile и т.д.)

# 3 Выполнение лабораторной работы

1. Создали учётную запись на https://github.com. (я же просто вошел в существующий аккаунт) (рис. 1)

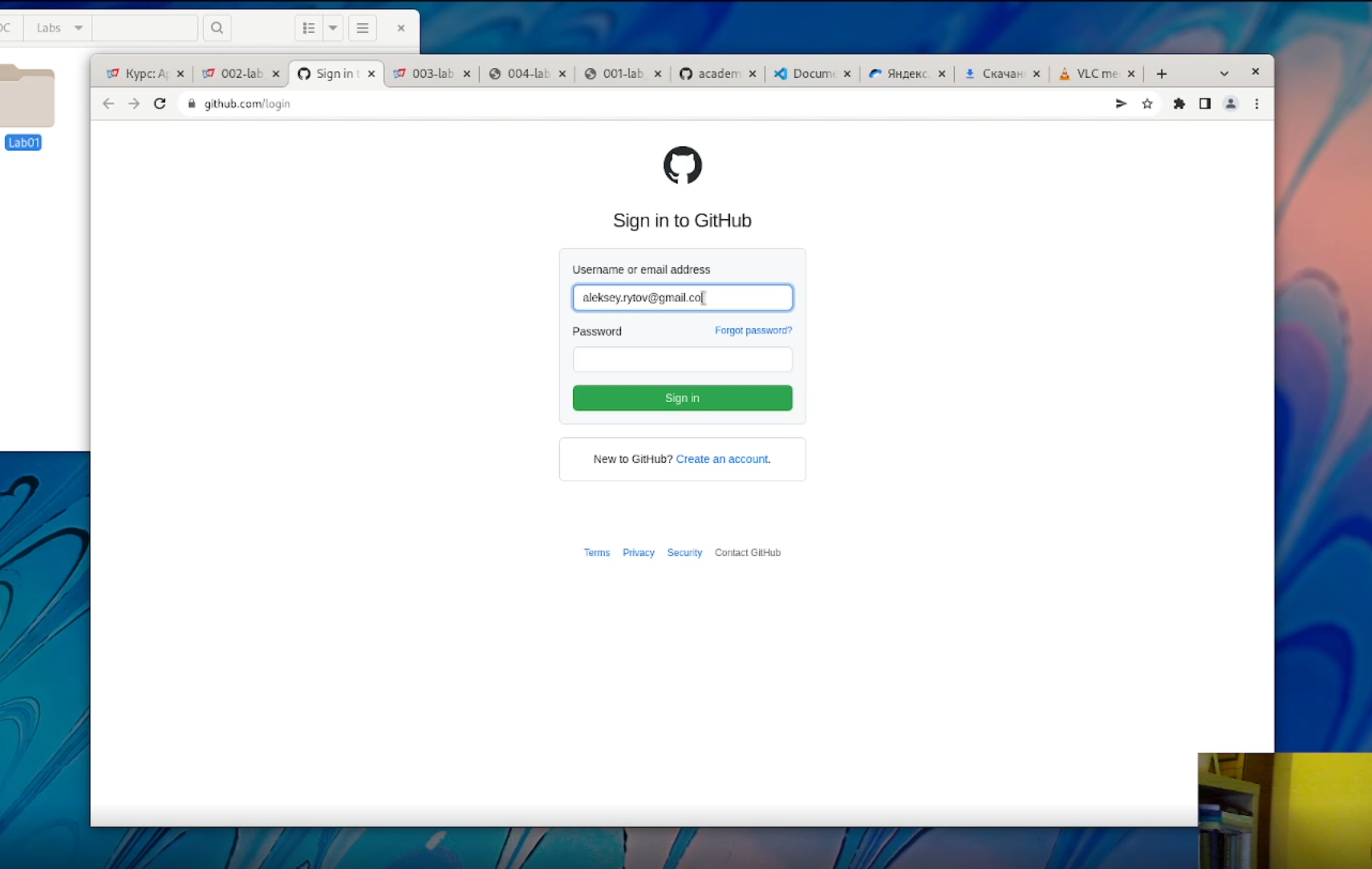


Рис. 1: Вход в учётную запись github

1. Заполнили основные данные на https://github.com. (рис. 2)

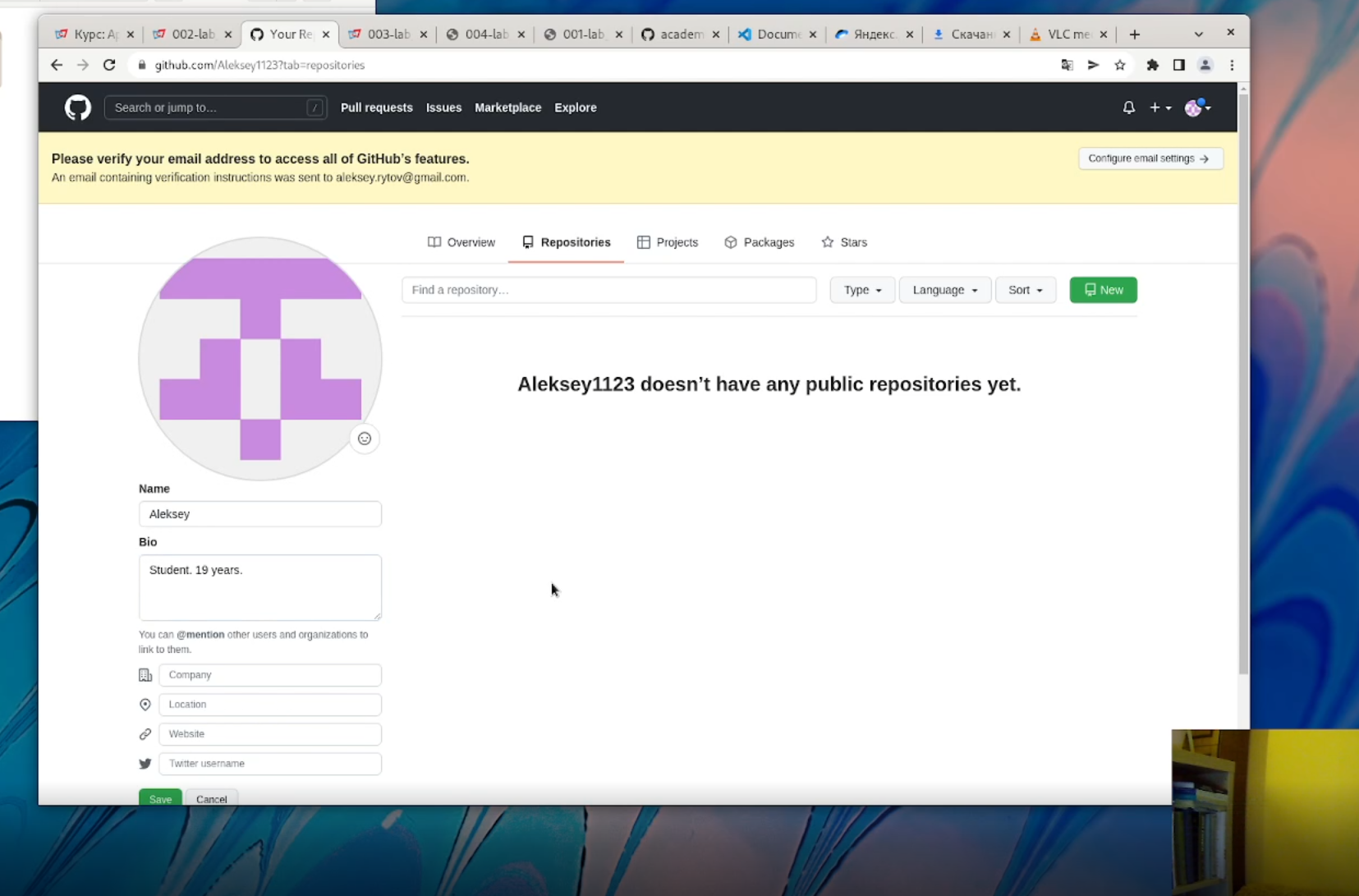


Рис. 2: Заполнение основных данных на github

1. Установили git-flow в Fedora Linux. (рис. 3)



Рис. 3: Установка git-flow

1. Установили gh в Fedora Linux. (рис. 4)

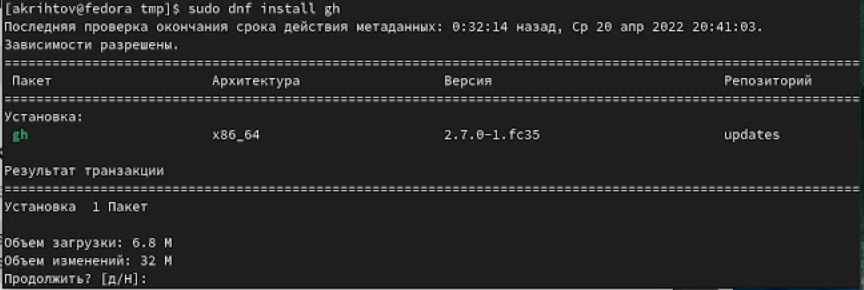


Рис. 4: Установка git-flow

1. Произвели базовую настройку git. (рис. 5)

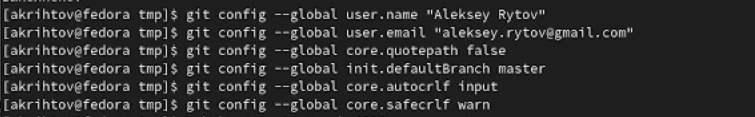
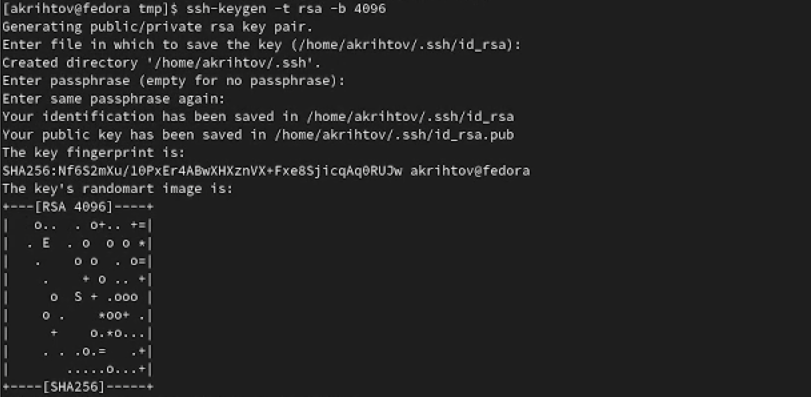


Рис. 5: Настройка git

1. Создали ключи ssh. (рис. 6.1 – 6.2)



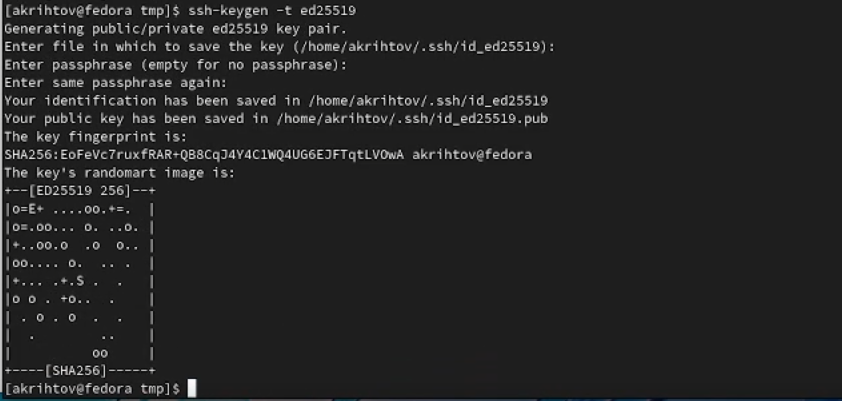
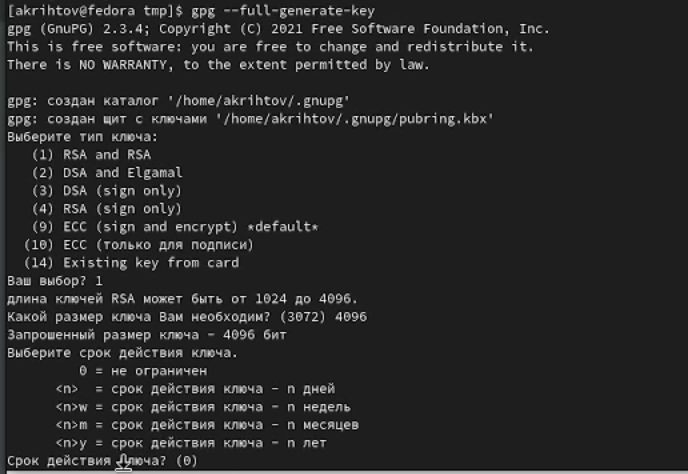


Рис. 6.2: Создание ssh ключа по алгоритму rsa размером 4096 бит

1. Создали ключ pgp. (рис. 7.1 – 7.2)



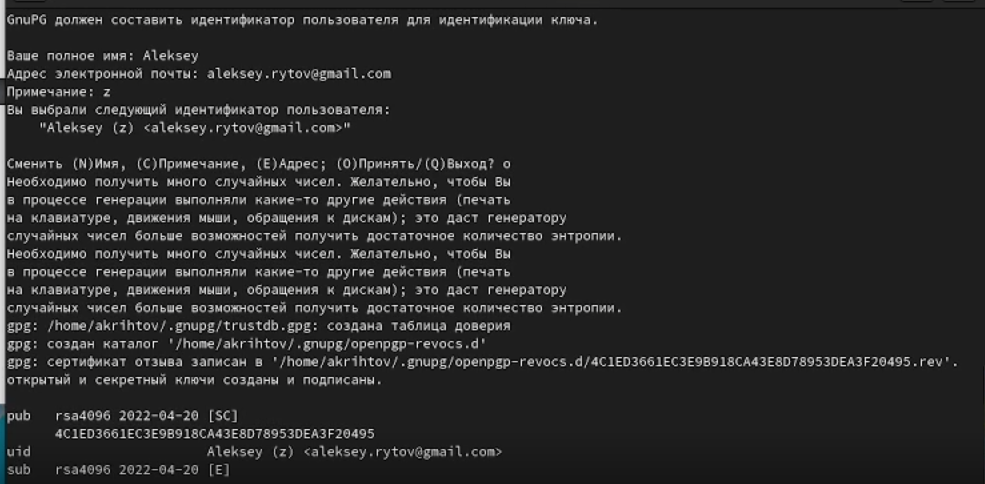
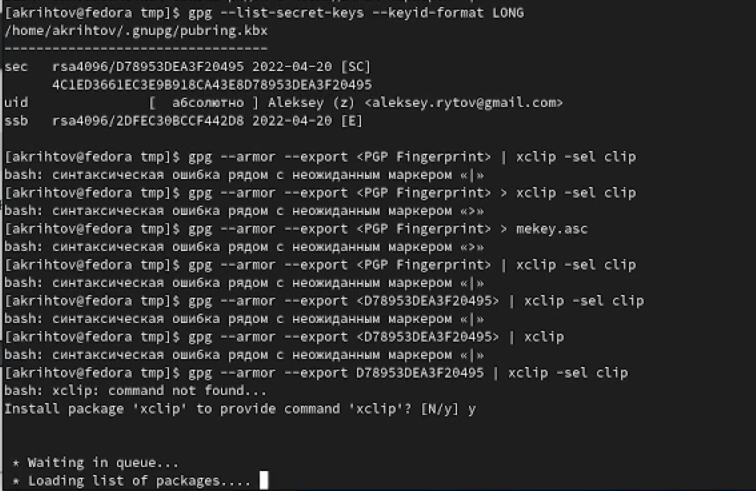


Рис. 7.2: Создание ключа gpg

1. Добавили pgp ключ в github. (рис. 7.1 – 7.2)



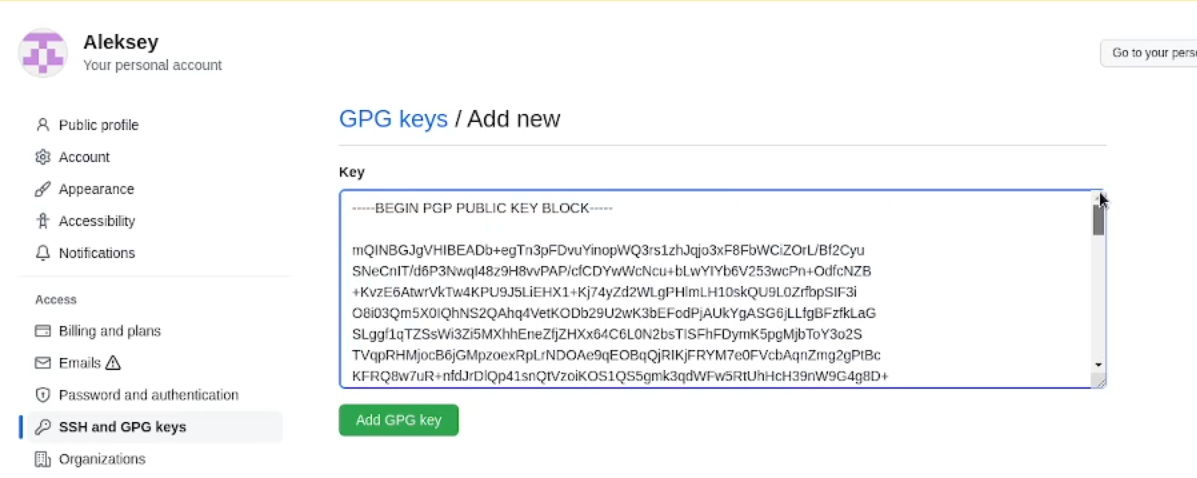


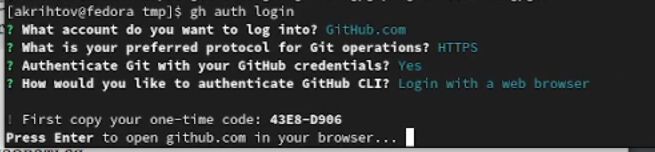
Рис. 8.2: Добавление gpg ключа на guthub

1. Настроили автоматические подписи. (рис. 9)

Настроили автоматические подписи. (рис. 9)

Настроили автоматические подписи. (рис. 9)

1. Настроили gh. (рис. 10.1 - 10.2)



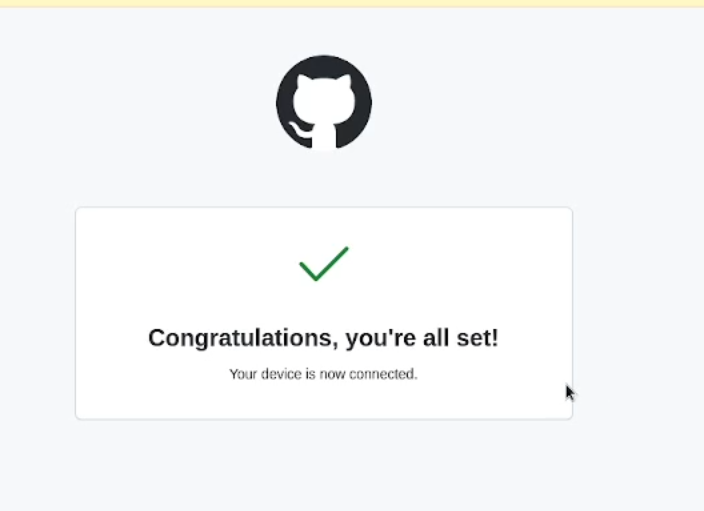
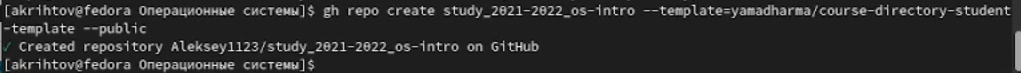


Рис. 10.2: Подтверждение регистрации

1. Создали репозиторий курса по шаблону. (рис. 11.1 – 11.3)

Рис. 11.1: Создание директории “Операционные системы” и переход в неё



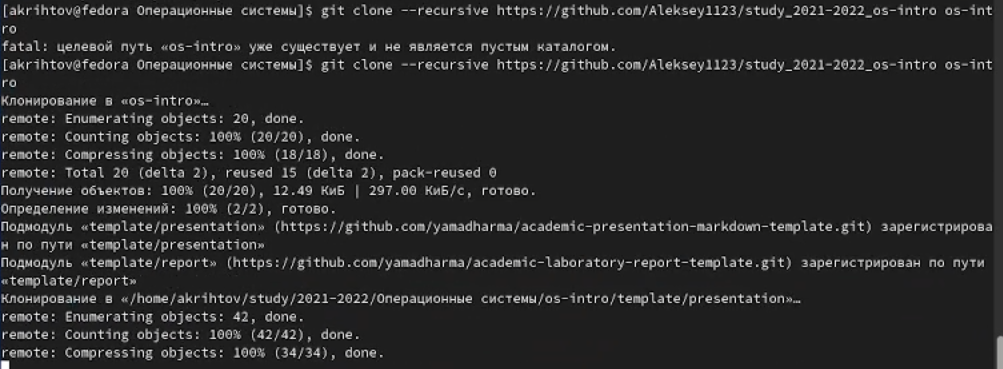
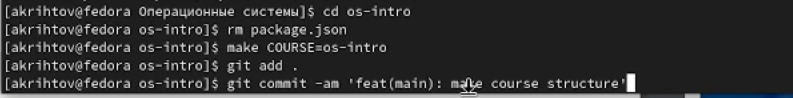


Рис. 11.3: Создание репозитория по шаблону на github

1. Настроили каталога курса. (рис. 12.1 – 12.)



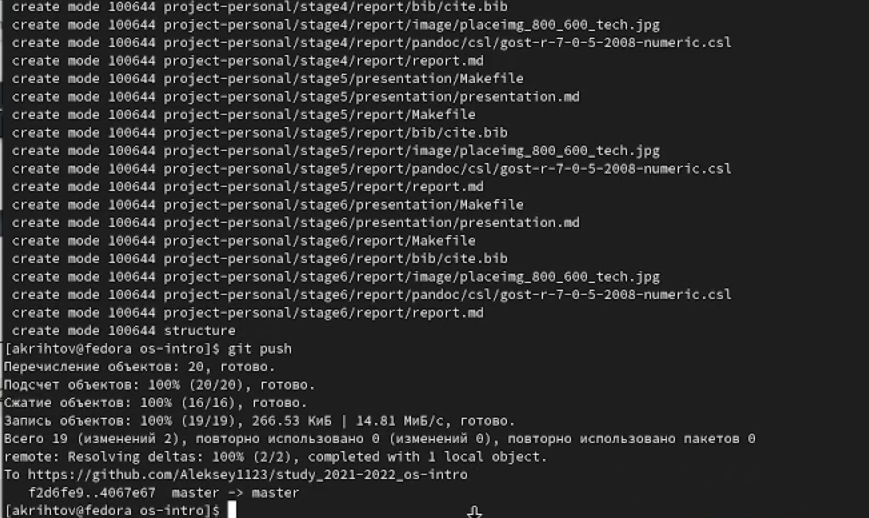


Рис. 12.2: Вывод команд git commit и git push

# 4 Выводы

Мы изучили идеологию и применение средств контроля версий, а также освоили умения по работе с git.

# 5 Ответы на контрольные вопросы:

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

Система контроля версий(VCS) – это хранилище, где находится код. Она предназначена для хранения нескольких версий файлов, позволяет совмещать изменения, произведённые разными участниками проекта, а также производить откат к любой более ранней версии проекта.

1. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

Хранилище – место, где хранится вся служебная информация, история изменения документов. Commit – процесс создания новой версии. Рабочая копия – последняя рабочая версия проекта, загруженная из хранилища.

1. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществлялся через специальное клиентское приложение. В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов.

1. Опишите действия с VCS при единоличной работе с хранилищем.
2. Произвожу блокировку(Lock) последней версии проекта.
3. Выгружаю проект с VCS. (Check-out)
4. Произвожу работу над проектом.
5. Сохраняю и выгружаю проект на VCS. (Check-in)
6. Произвожу разблокировку(Unlock) проекта.
7. Опишите порядок работы с общим хранилищем VCS.
8. Произвожу блокировку(Lock) последней версии проекта.
9. Выгружаю проект с VCS. (Check-out)
10. Произвожу работу над проектом.
11. Сохраняю и выгружаю проект на VCS. (Check-in)
12. Произвожу разблокировку(Unlock) проекта.
13. Каковы основные задачи, решаемые инструментальным средством git?

Git предназначен для хранения нескольких версий проекта, позволяет совмещать изменения, произведённые разными участниками проекта.

1. Назовите и дайте краткую характеристику командам 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: 1 git commit -am ‘Описание коммита’

– сохранить добавленные изменения с внесением комментария через встроенный редактор: Кулябов Д. С. и др. Операционные системы 23 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. Приведите примеры использования при работе с локальным и удалённым репозиториями.

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

1 cd

2 mkdir tutorial

3 cd tutorial

4 git init

команда status для просмотра изменений в рабочем каталоге, сделанных с момента последней ревизии: git status

1. Что такое и зачем могут быть нужны ветви (branches)?

Ветка – это просто «скользящий» указатель на один из коммитов. Когда мы создаём новые коммиты, указатель ветки автоматически сдвигается вперёд, к вновь созданному коммиту.

Ветки используются для разработки одной части функционала изолированно от других. Каждая ветка представляет собой отдельную копию кода проекта. Ветки позволяют одновременно работать над разными версиями проекта.

Ветвление («ветка», branch) — один из параллельных участков истории в одном хранилище, исходящих из одной версии (точки ветвления). Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом.

1. Как и зачем можно игнорировать некоторые файлы при commit?

Игнорируемые файлы обычно представляют собой файлы, специфичные для платформы, или автоматически созданные из сборочных систем. Временно игнорировать изменения в файле можно командой: git update-index —assume-unchanged