

# 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: Абдулгалимов Мурад

Группа: НКНбд-02-21

МОСКВА

2022 г.

## Цель работы:

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

### 1. Создал учетную запись и заполнил основные данные:

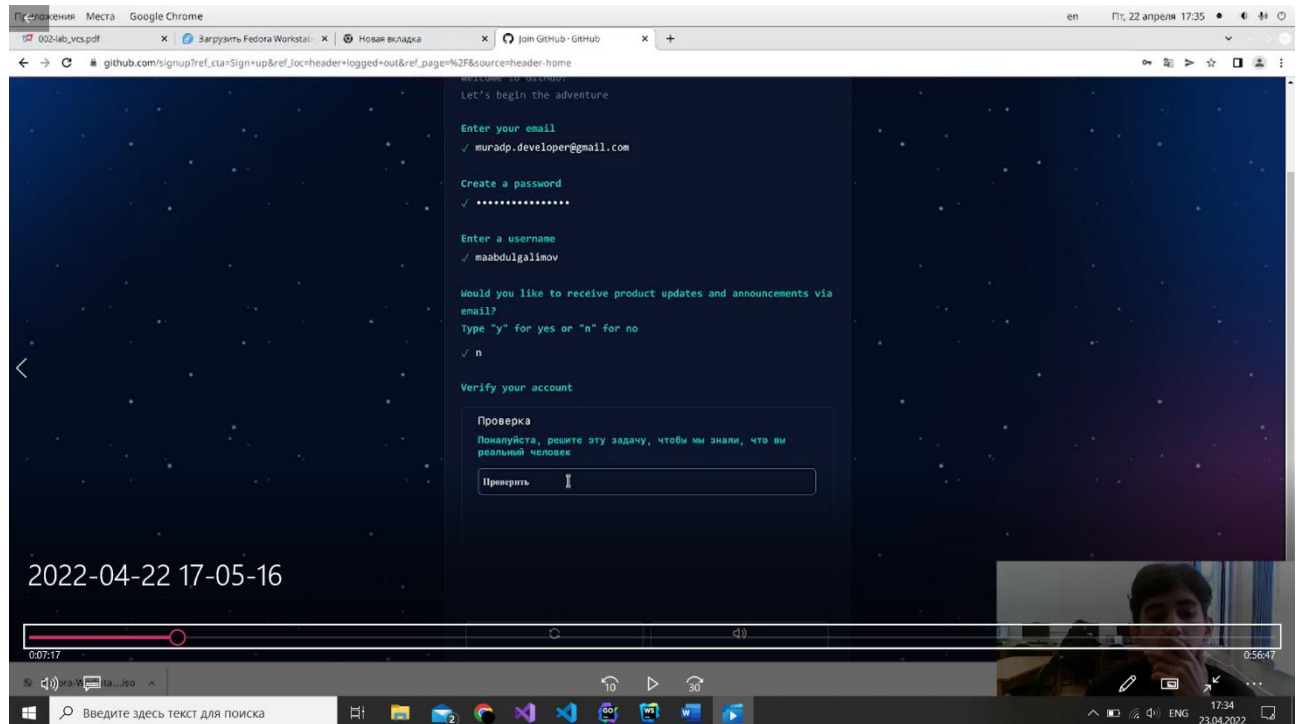


Рис. 1 Создание учетной записи.

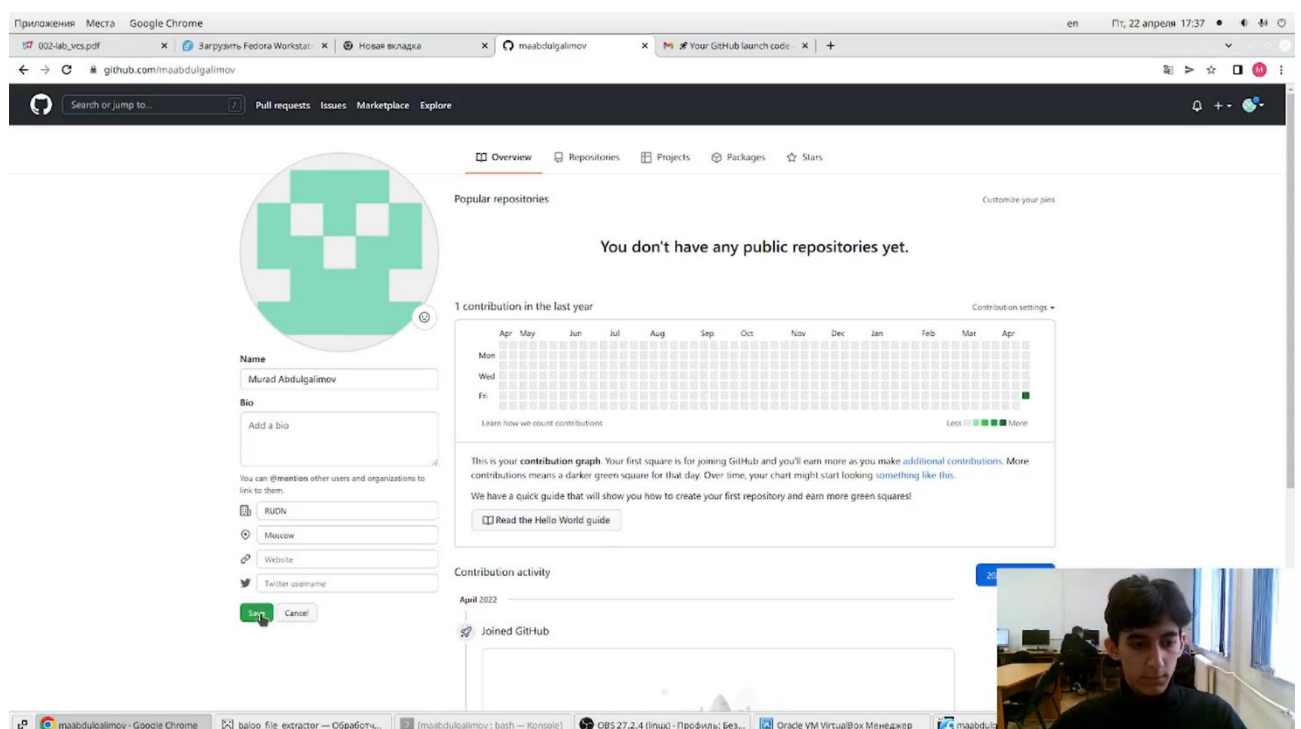


Рис. 2 ввод основной информации

## 2. Установил git-flow в Fedora Linux

Команды:

```
cd /tmp
wget --no-check-certificate -q https://raw.githubusercontent.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh
chmod +x gitflow-installer.sh
sudo ./gitflow-installer.sh install stable
```

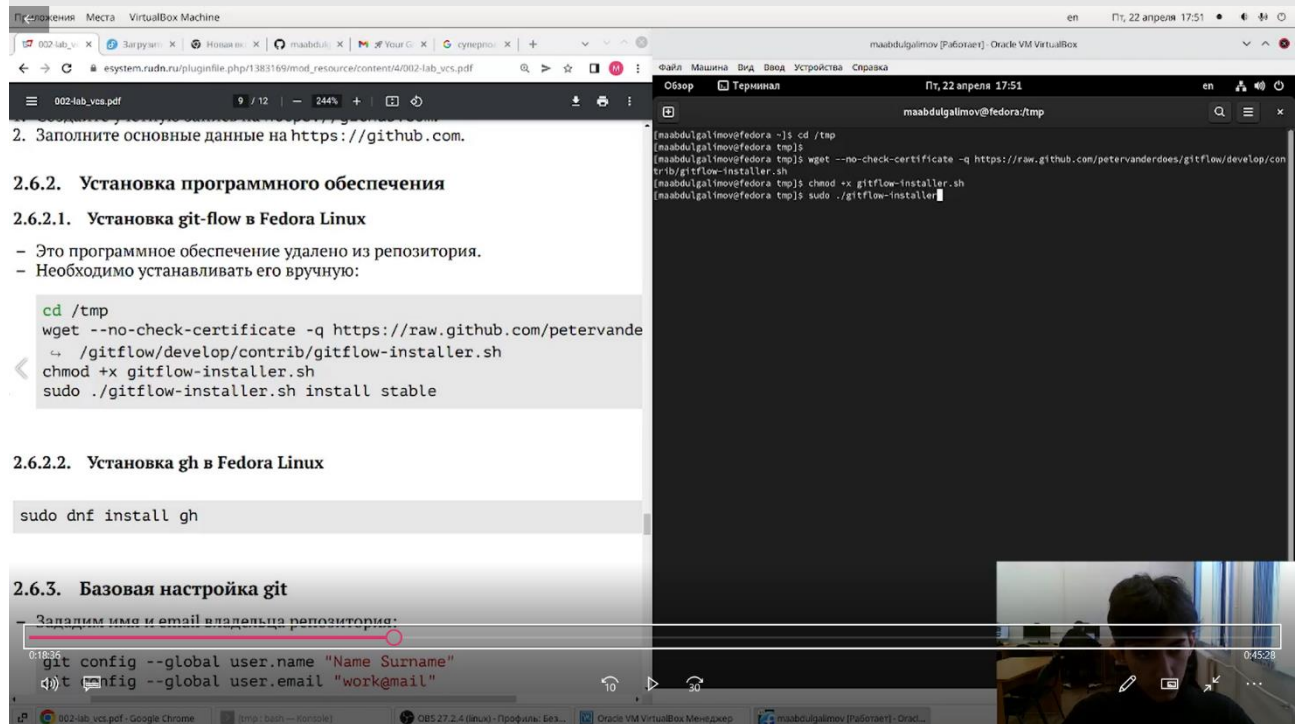


Рис. 3 Установка git-flow в Fedora Linux

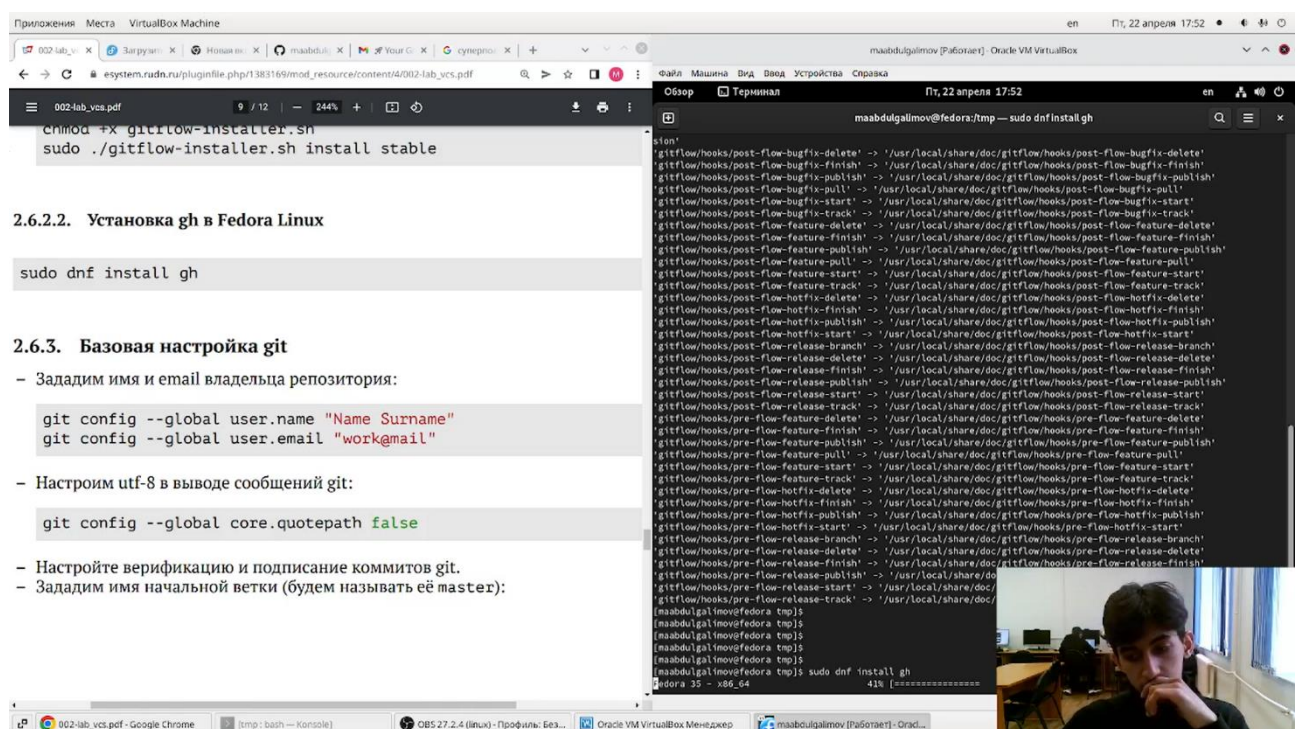


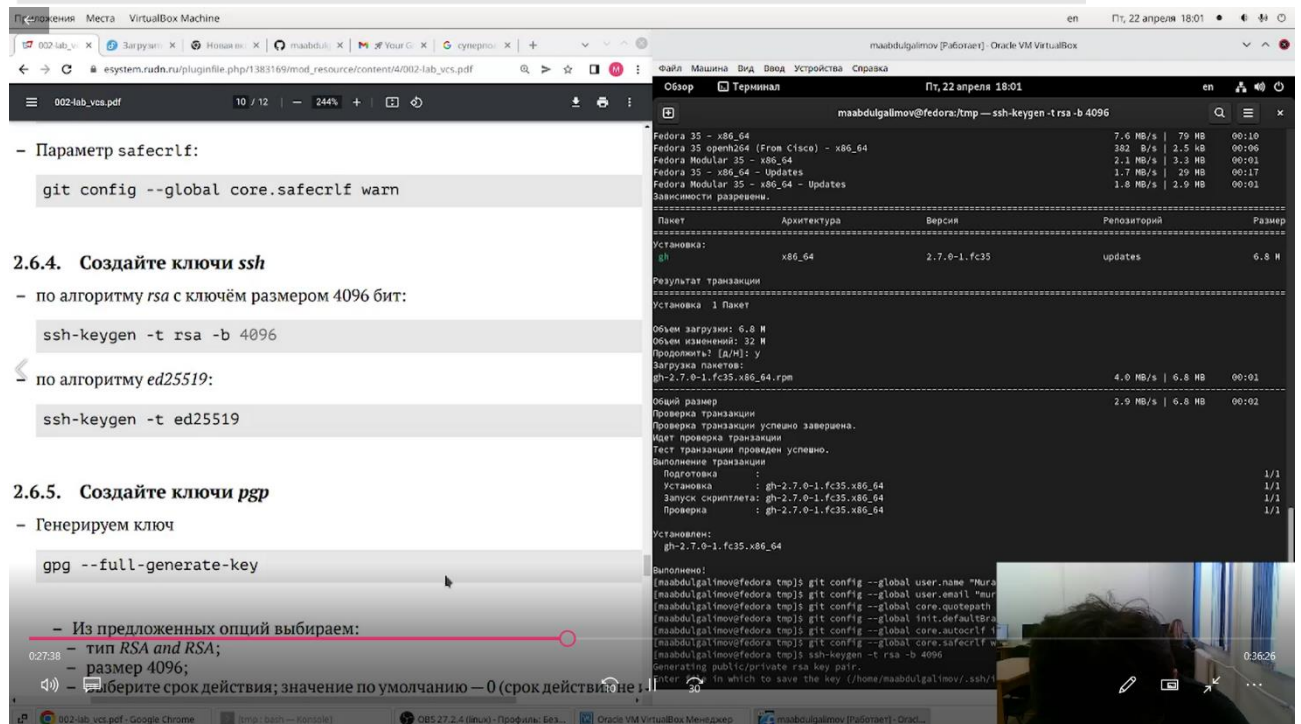
Рис. 4 Установка git-flow в Fedora Linux

### 3. Провел базовую настройку Git

Команды:

```
git config --global user.name "Name Surname"
git config --global user.email "work@mail"
```

```
git config --global core.quotepath false
```





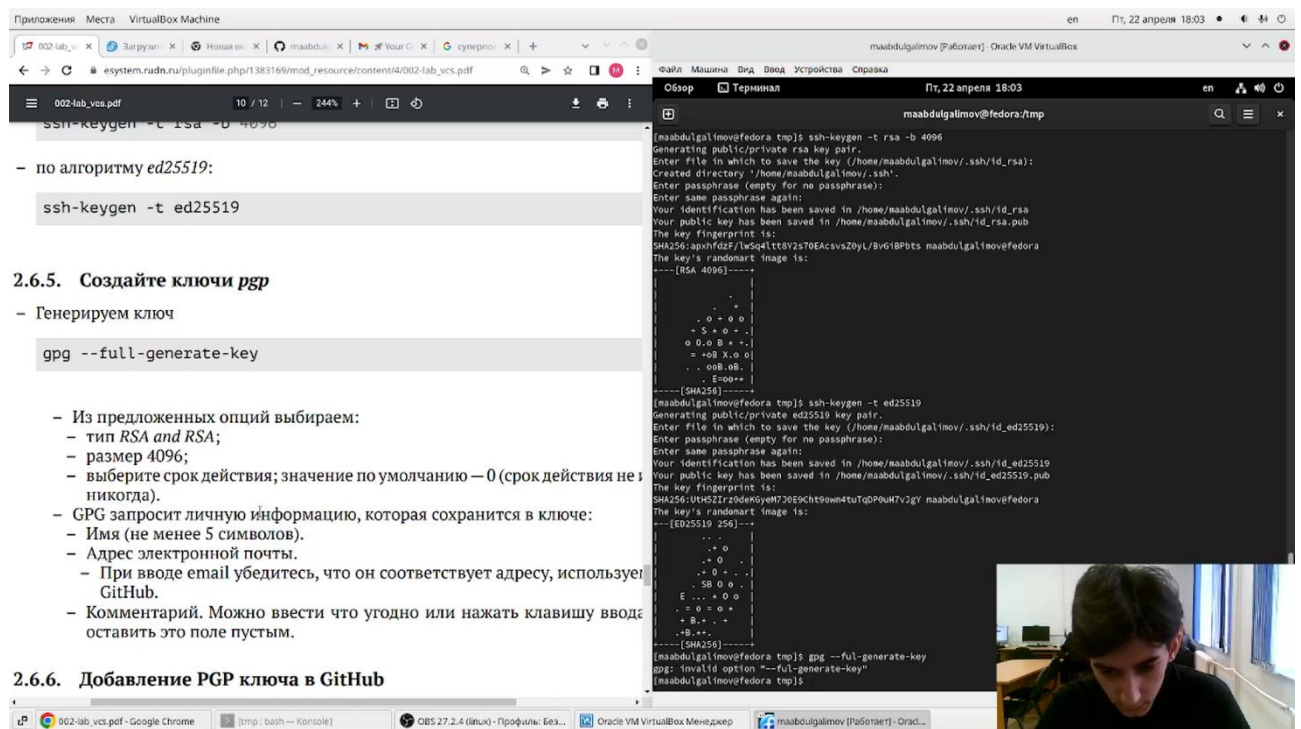


Рис. 6 Генерация SSH

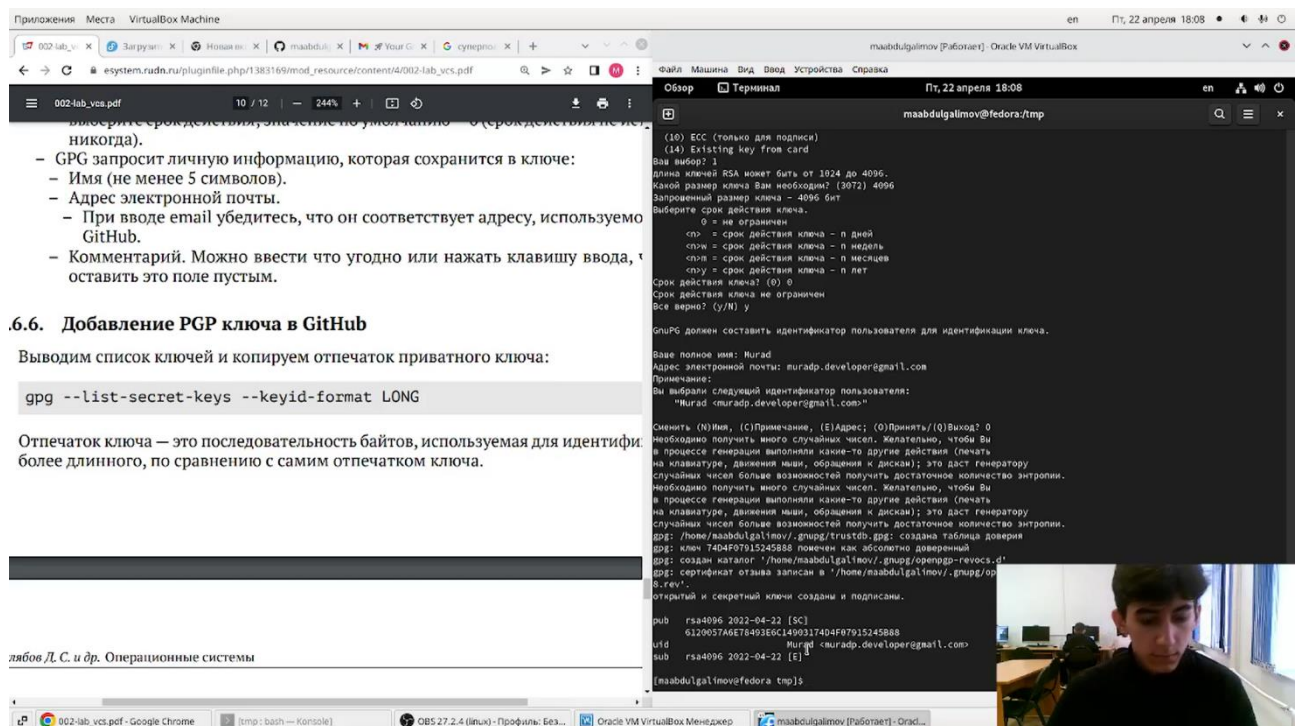


Рис. 7 Генерация PGP

## 5. Добавление PGP в GitHub:

Команды:

```
gpg --list-secret-keys --keyid-format LONG
```

```
gpg --armor --export <PGP Fingerprint> | xclip -sel clip
```

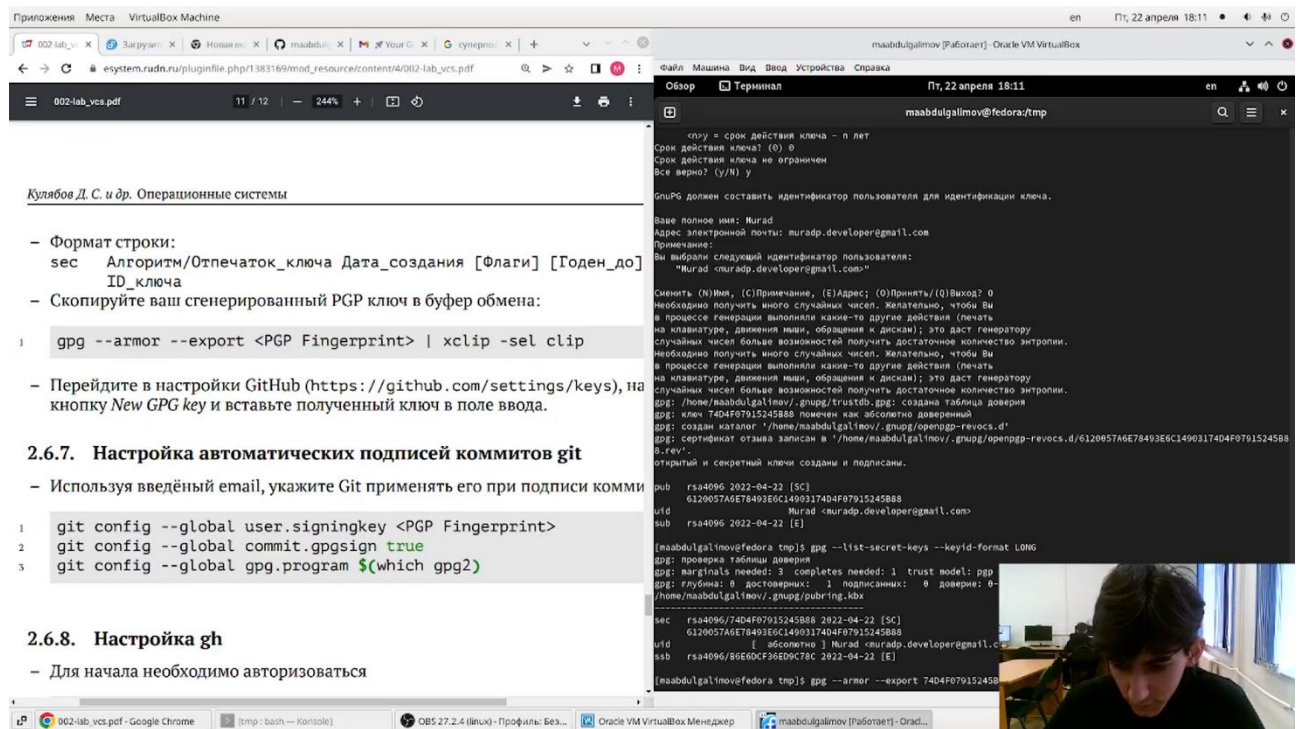


Рис. 8 Добавление PGP в GitHub

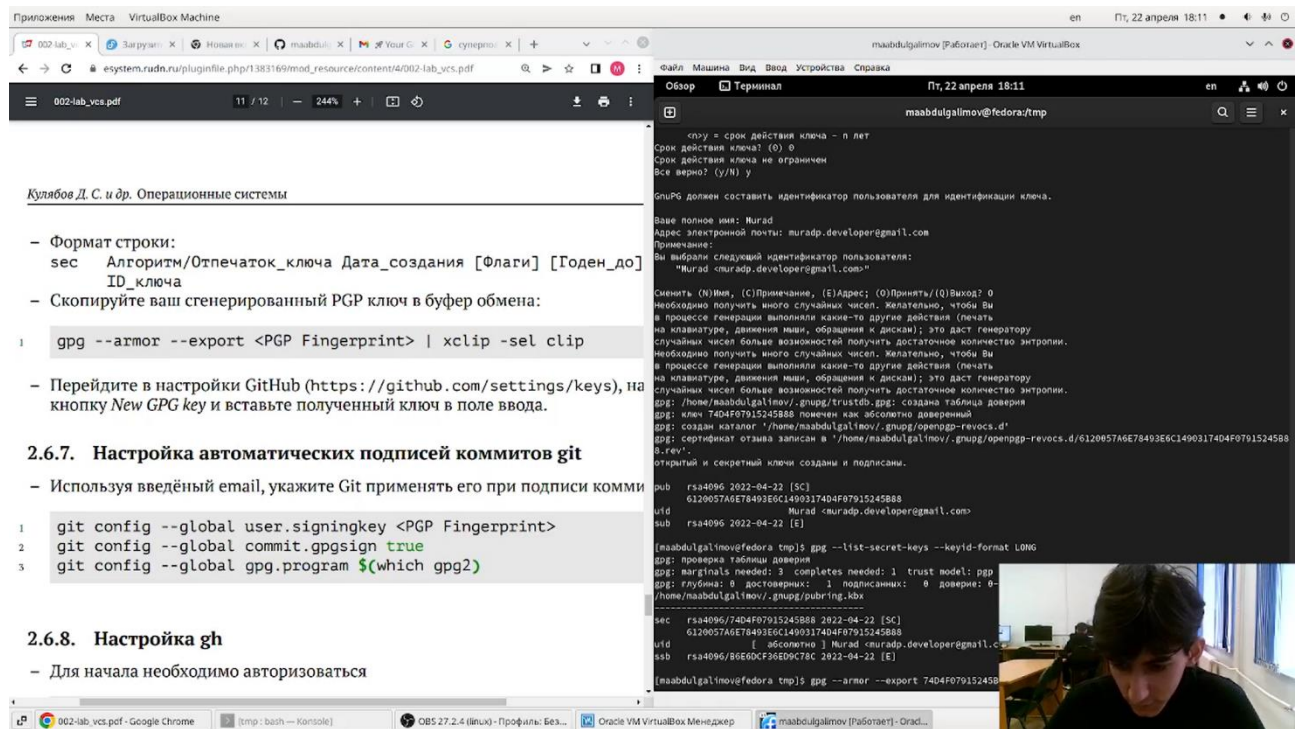


Рис. 9 Добавление PGP в GitHub

## 6. Настроил автоматические подписи для комитов Git:

Команды:

```

git config --global user.signingkey <PGP Fingerprint>
git config --global commit.gpgsign true
git config --global gpg.program $(which gpg2)

```



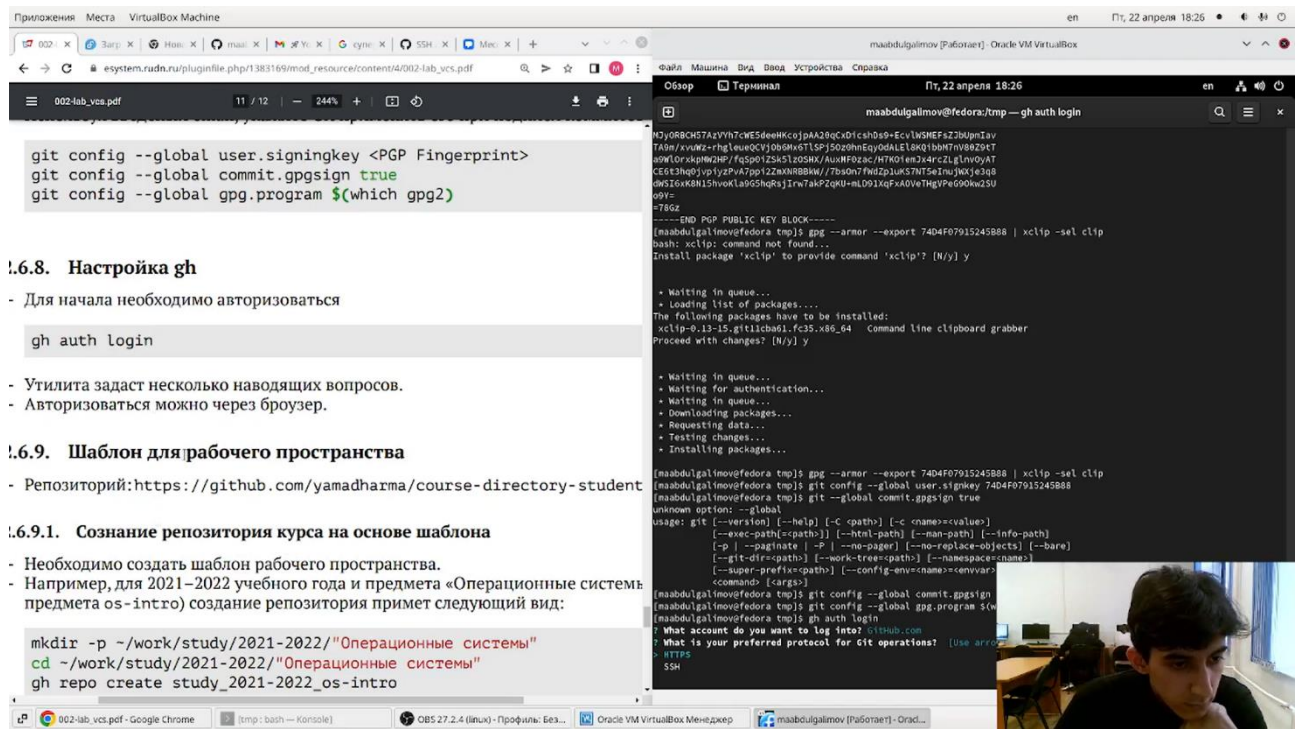


Рис. 10 Настроил автоматические подписи для комитов Git

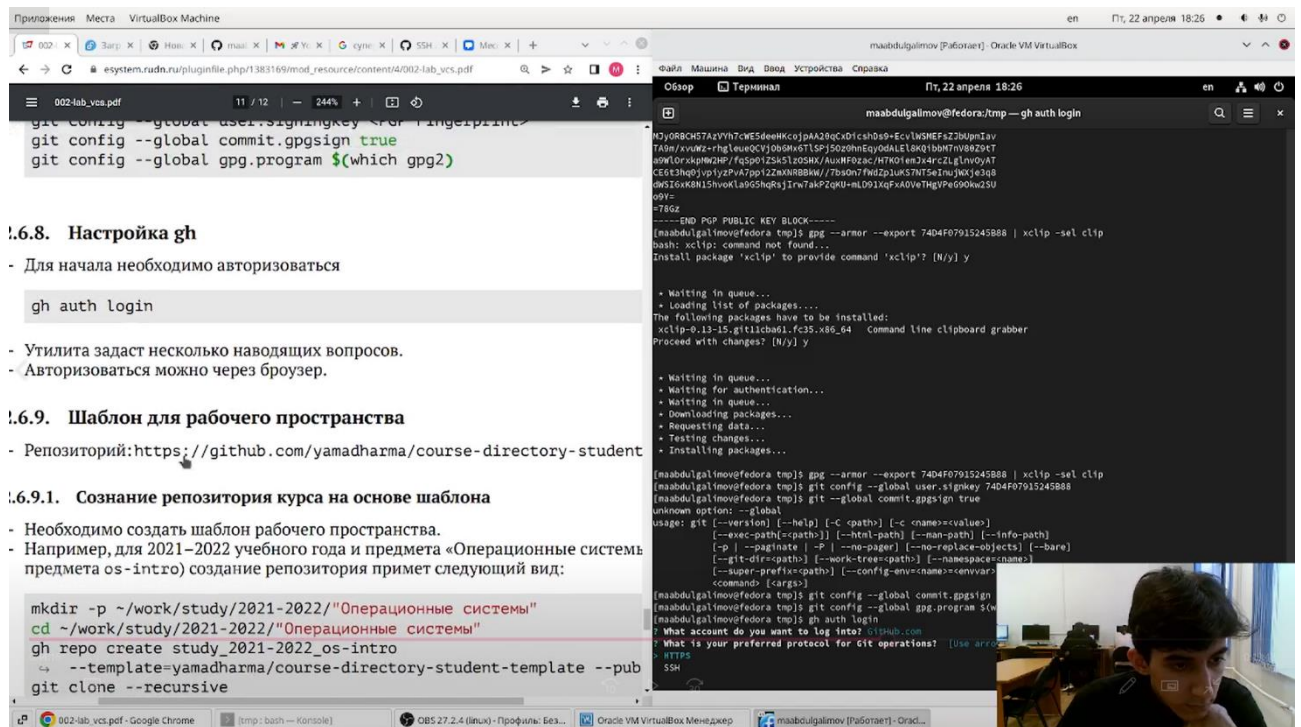


Рис. 11 Настроил gh

## 7. Создание репозитория курса на основе шаблона: Команды:



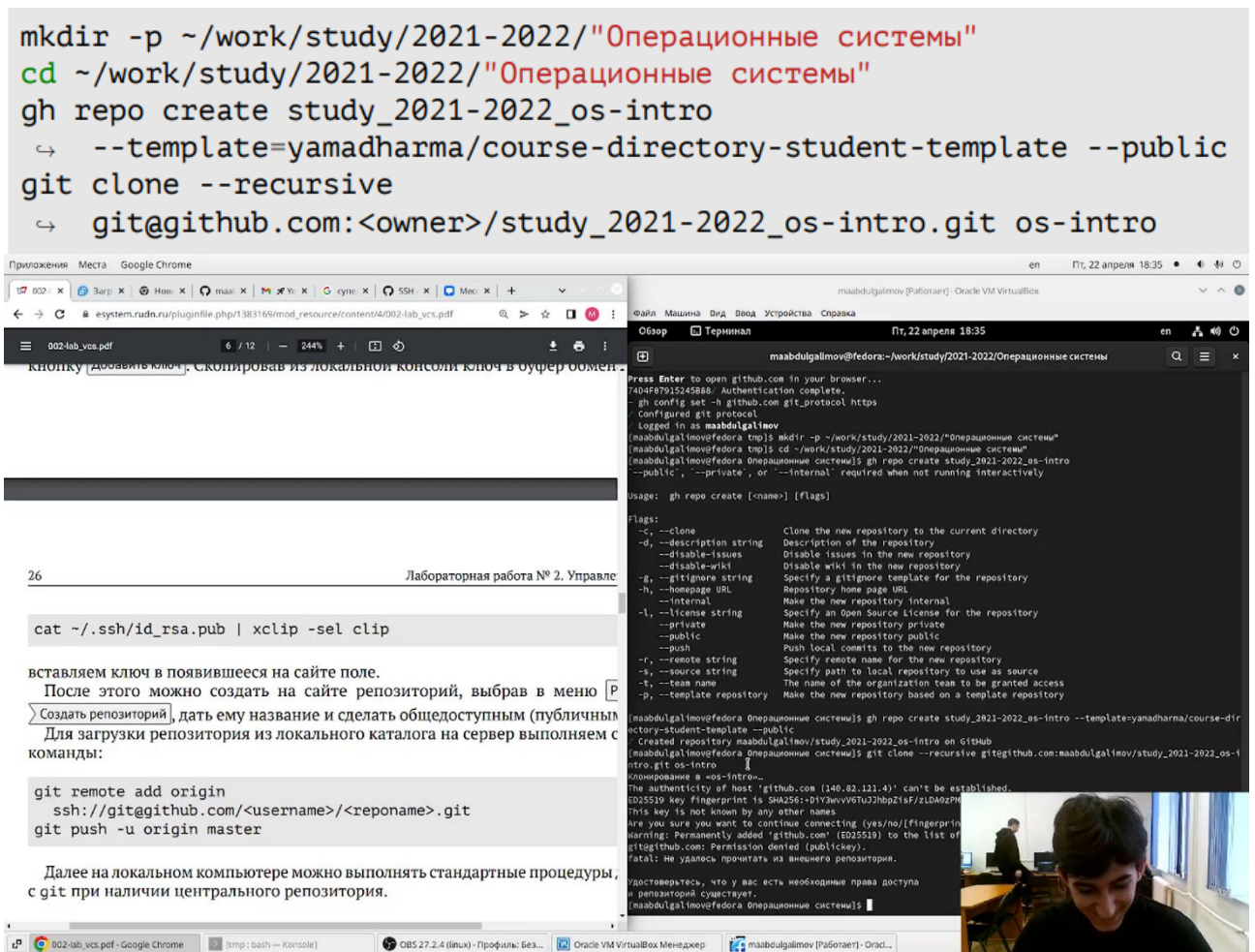


Рис. 12 Создание репозитория на основе шаблона

## 8. Настроил каталог курса и отправил на сервер:

Команды:

```
cd ~/work/study/2021-2022/"Операционные системы"/os-intro
rm package.json
make COURSE=os-intro

git add .
git commit -am 'feat(main): make course structure'
git push
```

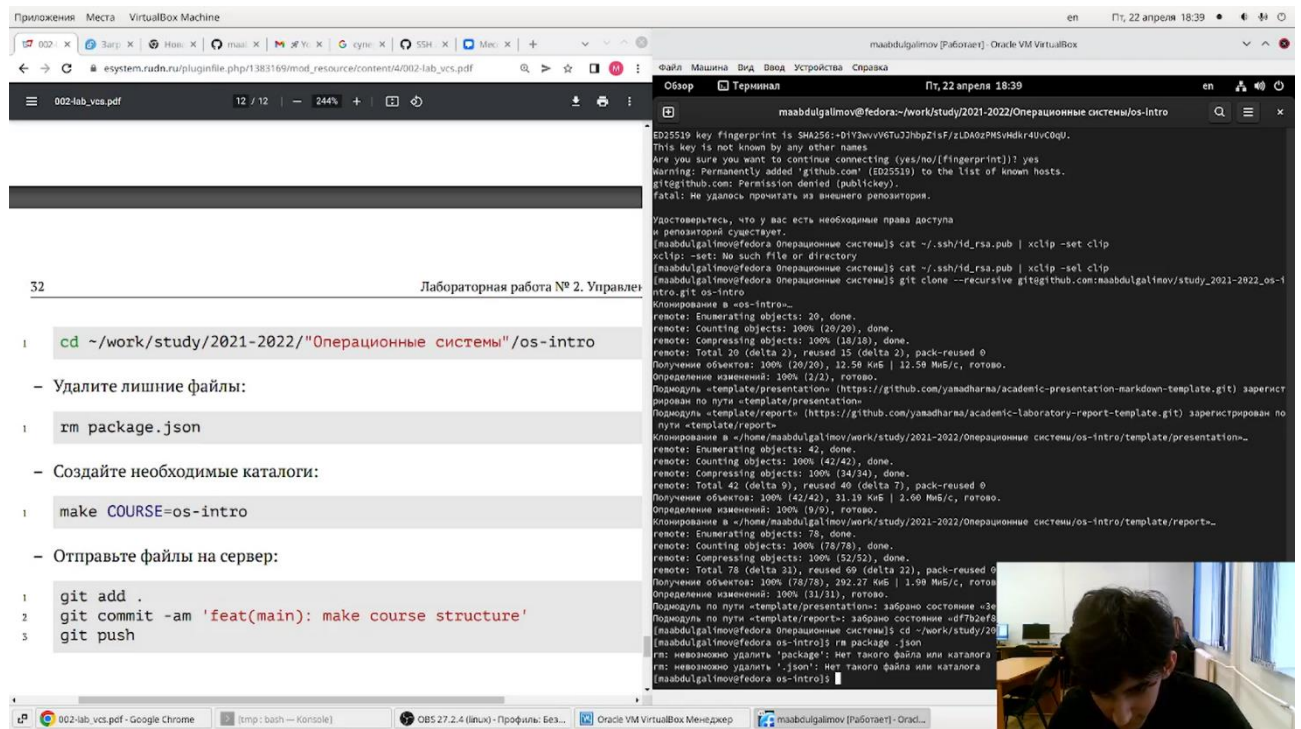


Рис. 13 Настройка каталога курса

## Контрольные вопросы:

1. Системы контроля версий — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом.

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

Хранилище — репозиторий — место хранения всех версий и служебной информации.

Commit — это команда для записи индексированных изменений в репозиторий.

История — место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах.

Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.

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

Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion.

Децентрализованные системы – система, в которой каждый пользователь имеет свой вариант репозитория и есть возможность добавлять и забирать изменения из репозитория. Пример – Git.

4. . Опишите действия с VCS при единоличной работе с хранилищем.

5. Опишите порядок работы с общим хранилищем VCS.

6. Каковы основные задачи, решаемые инструментальным средством git?

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. 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 изменений: `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`

– удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): `git rm` имена\_файлов

– 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 с текущим деревом: `git merge --no-ff` имя\_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. Приведите примеры использования при работе с локальным и удалённым репозиториями.

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

Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.

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

Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл `.gitignore`. Временно игнорировать

изменения в файле можно командой `git update-index-  
assumeunchanged<file>`

Вывод:

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