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: КОНЕ СИРИКИ

Группа: НФИБД -01-20

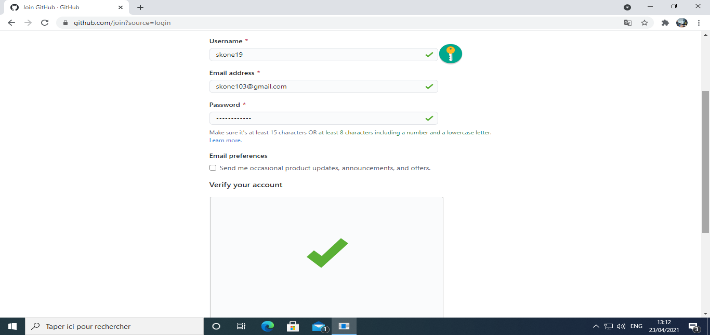
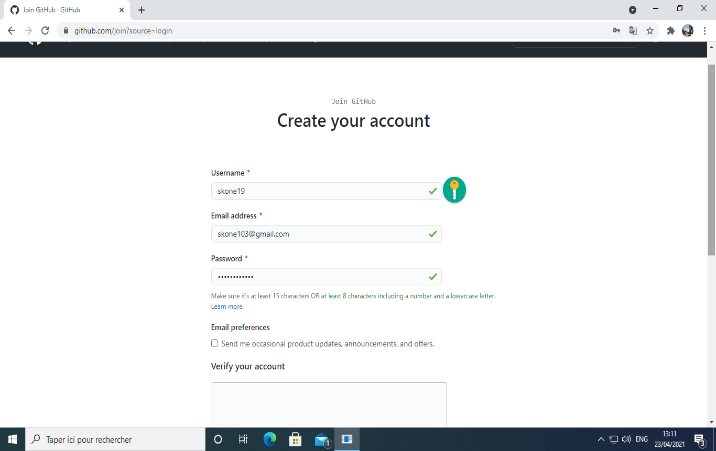
**МОСКВА**

2021 г.

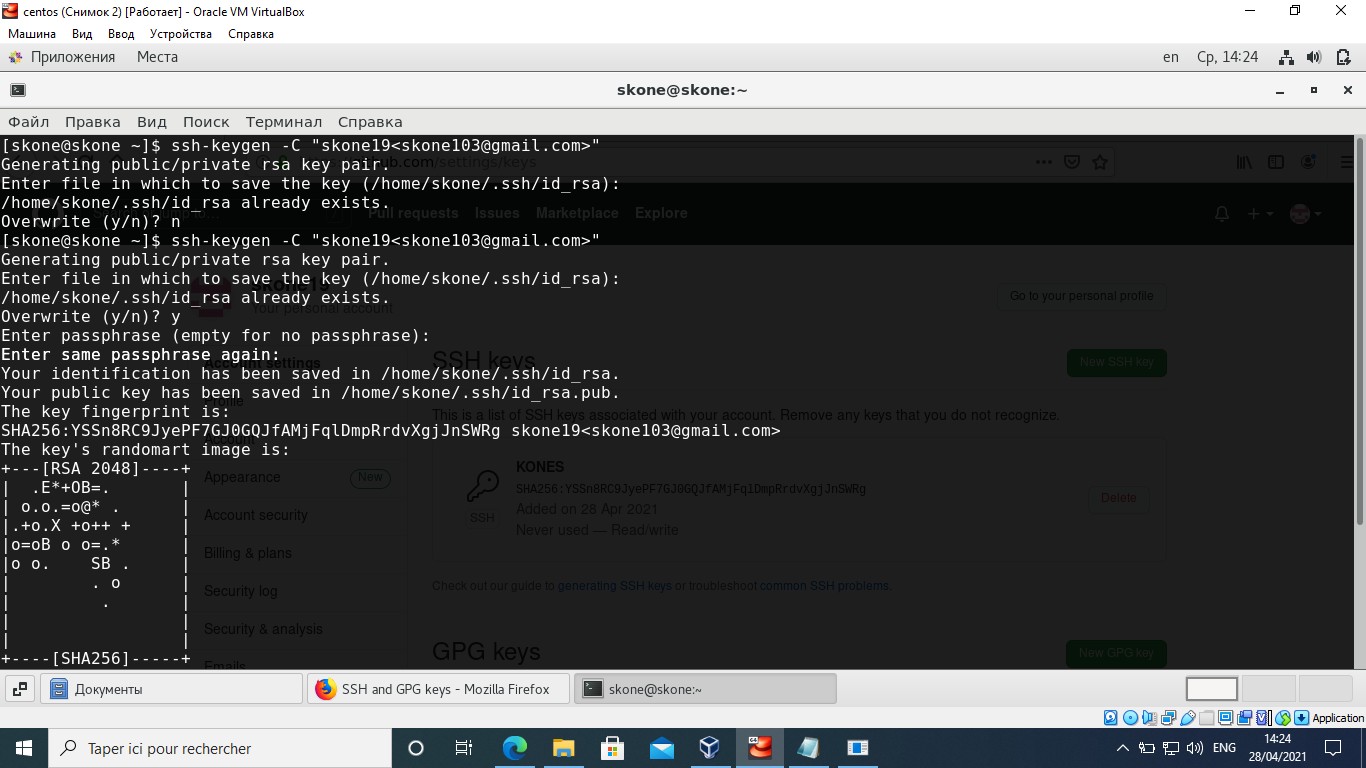
**Цель работы: изучить идеологию и применение средств контроля версий.**

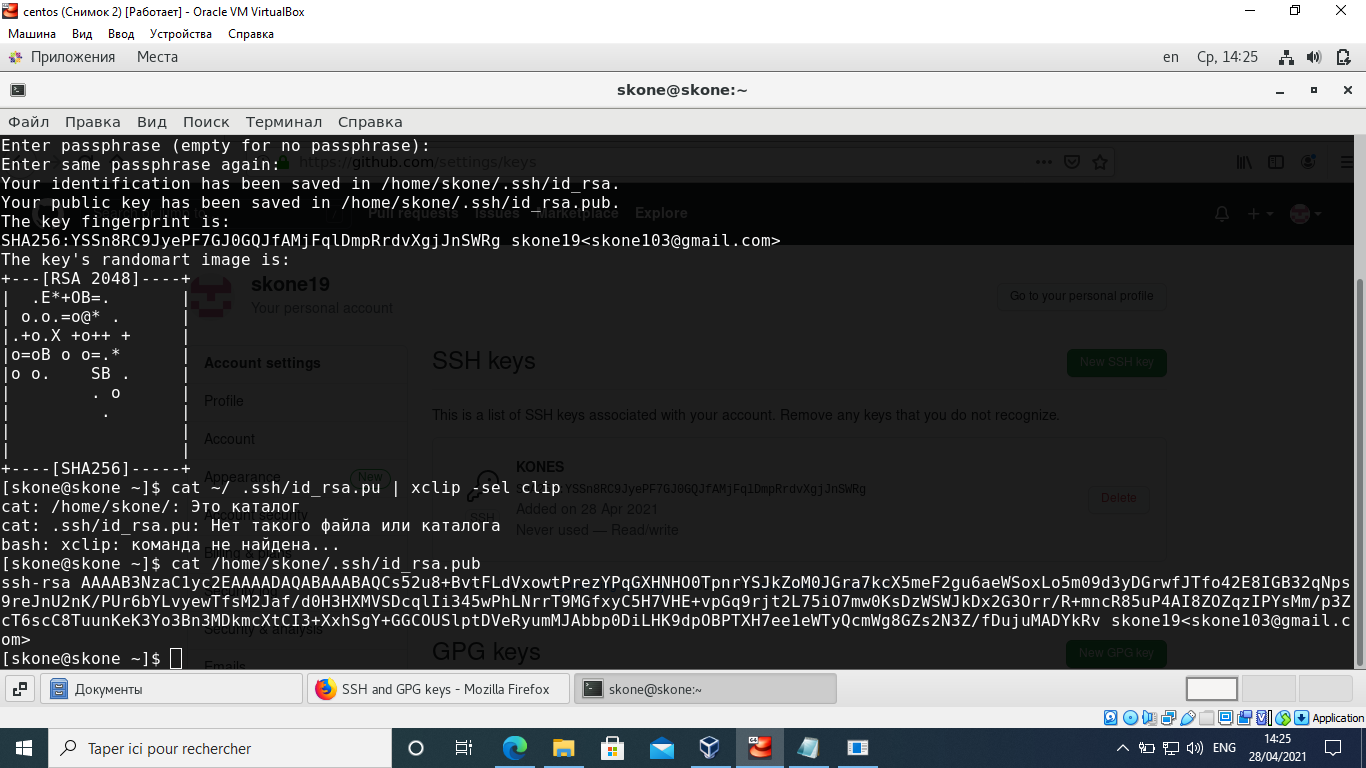
**Ход работы:**

1. Создайте учётную запись на <https://github.com>

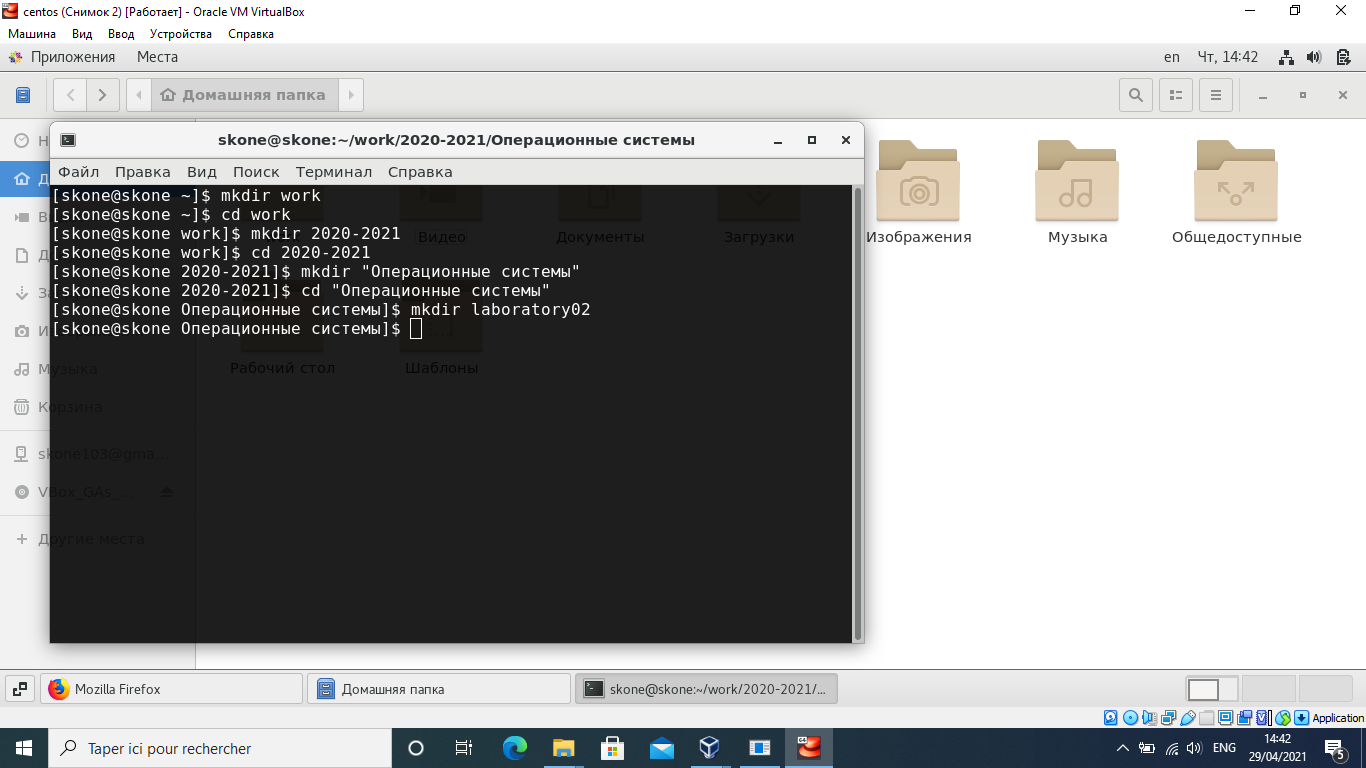


1. Настройте систему контроля версий git, как это описано выше c использованием сервера репозиториев <https://github.com/>.



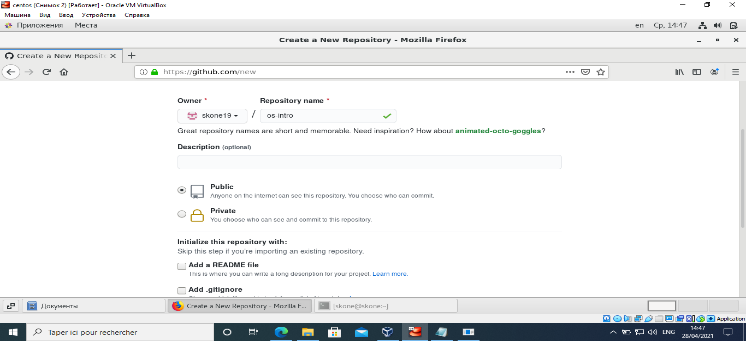


1. Создал структуру каталога лабораторных работ согласно пункту М.2.

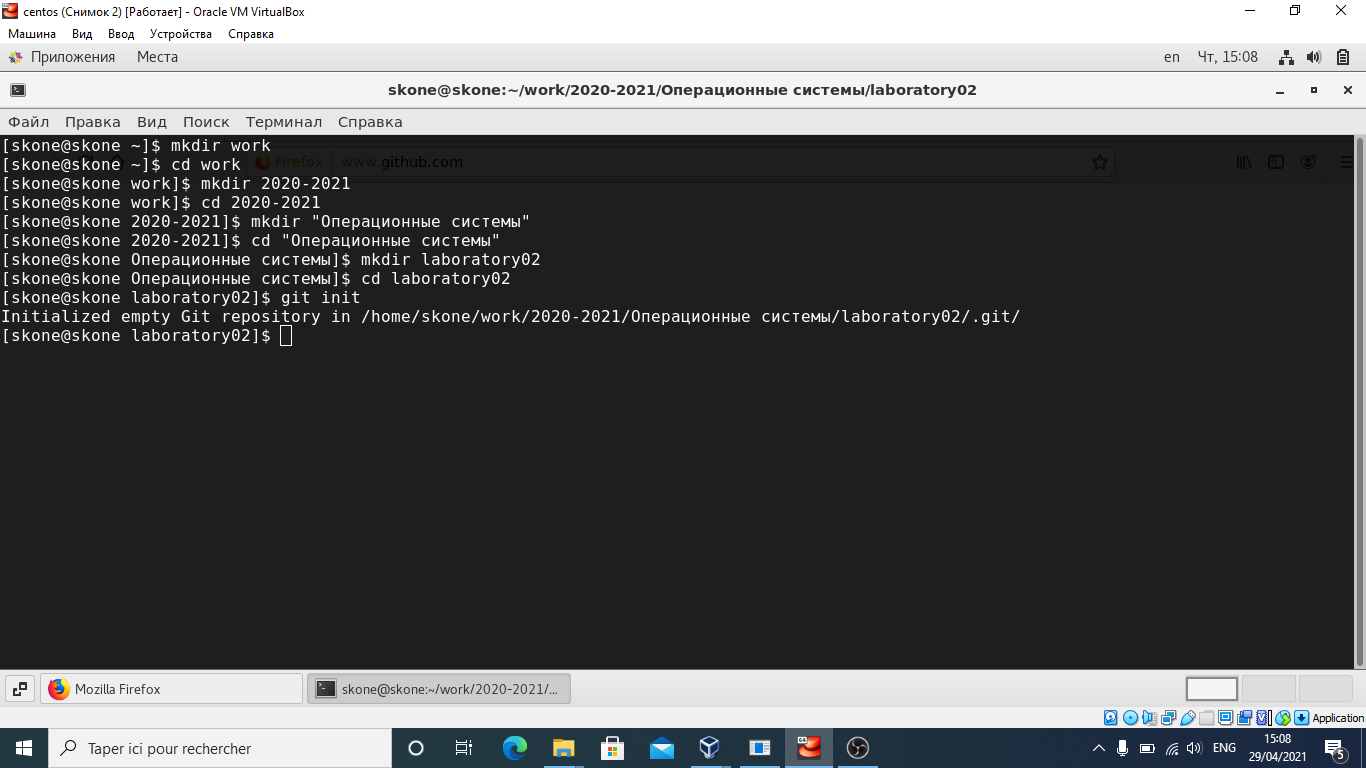


4- Подключение репозитория к github

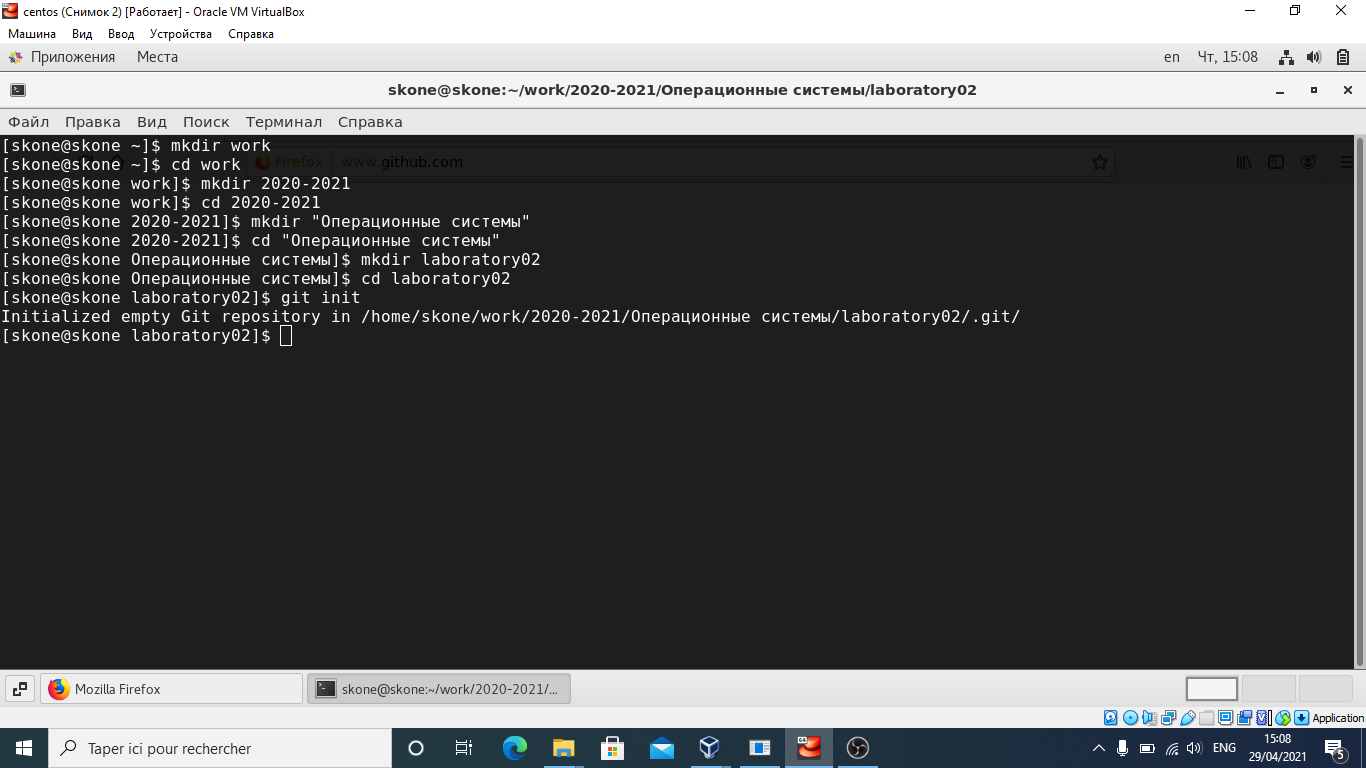
– Создайте репозиторий на GitHub. Для примера назовём его os-intro.



– Рабочий каталог будем обозначать как laboratory. Вначале нужно перейти в этот каталог: cd laboratory

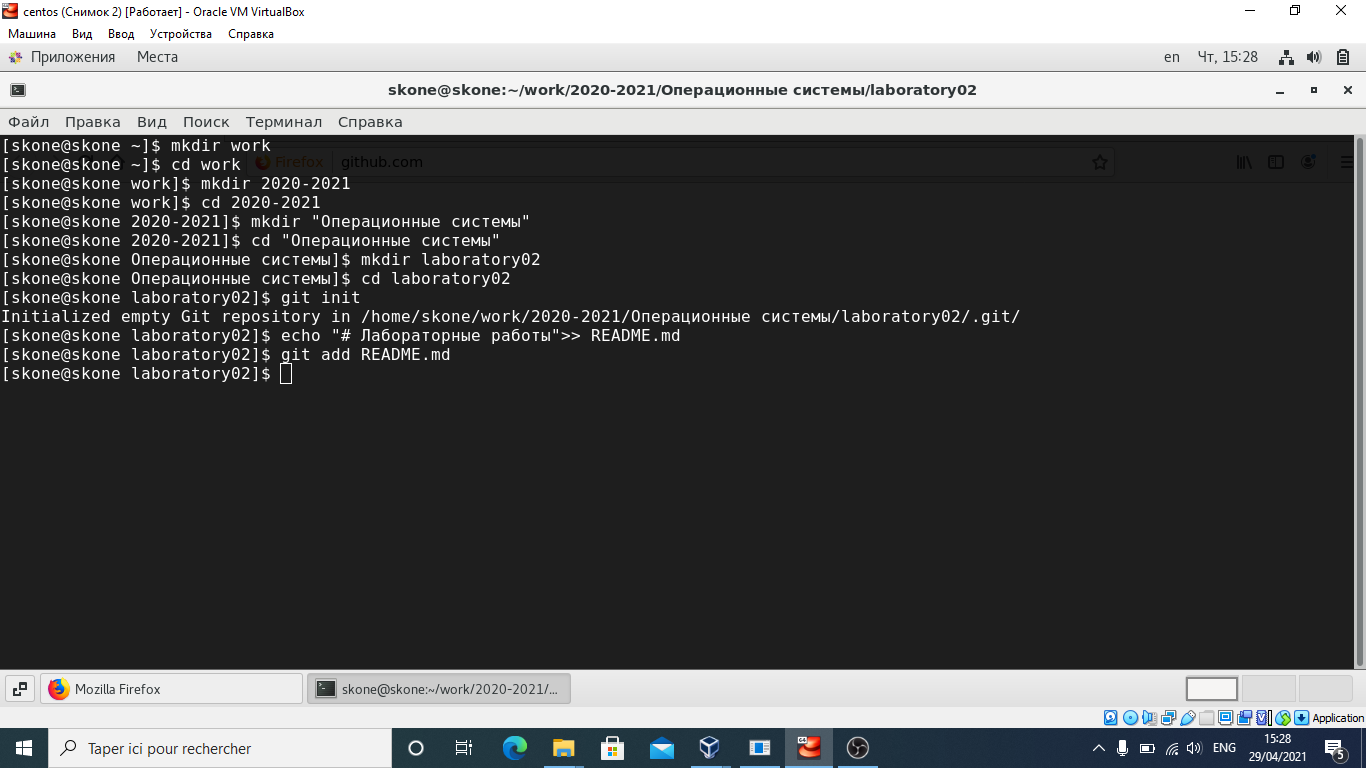


– Инициализируем системы git: git init



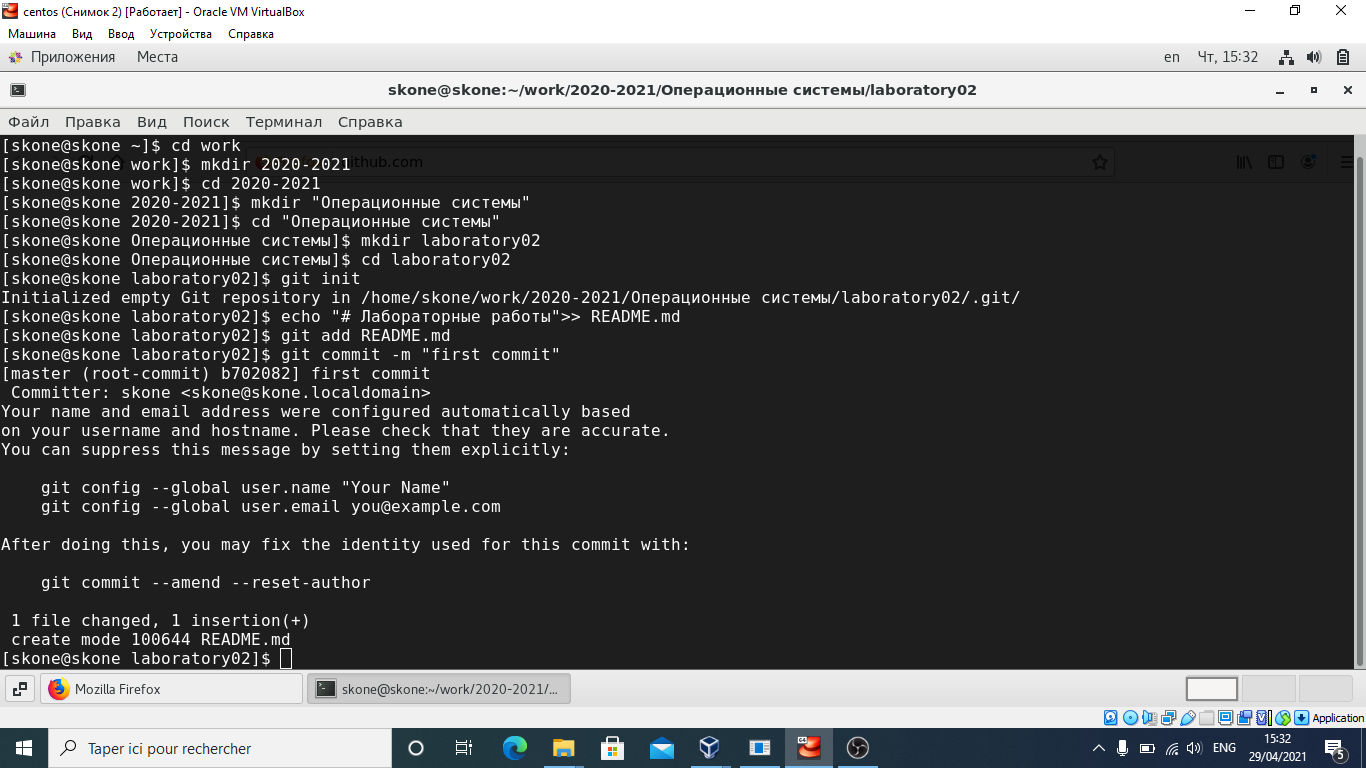
– Создаём заготовку для файла README.md:

echo "# Лабораторные работы" >> README.md git add README.md



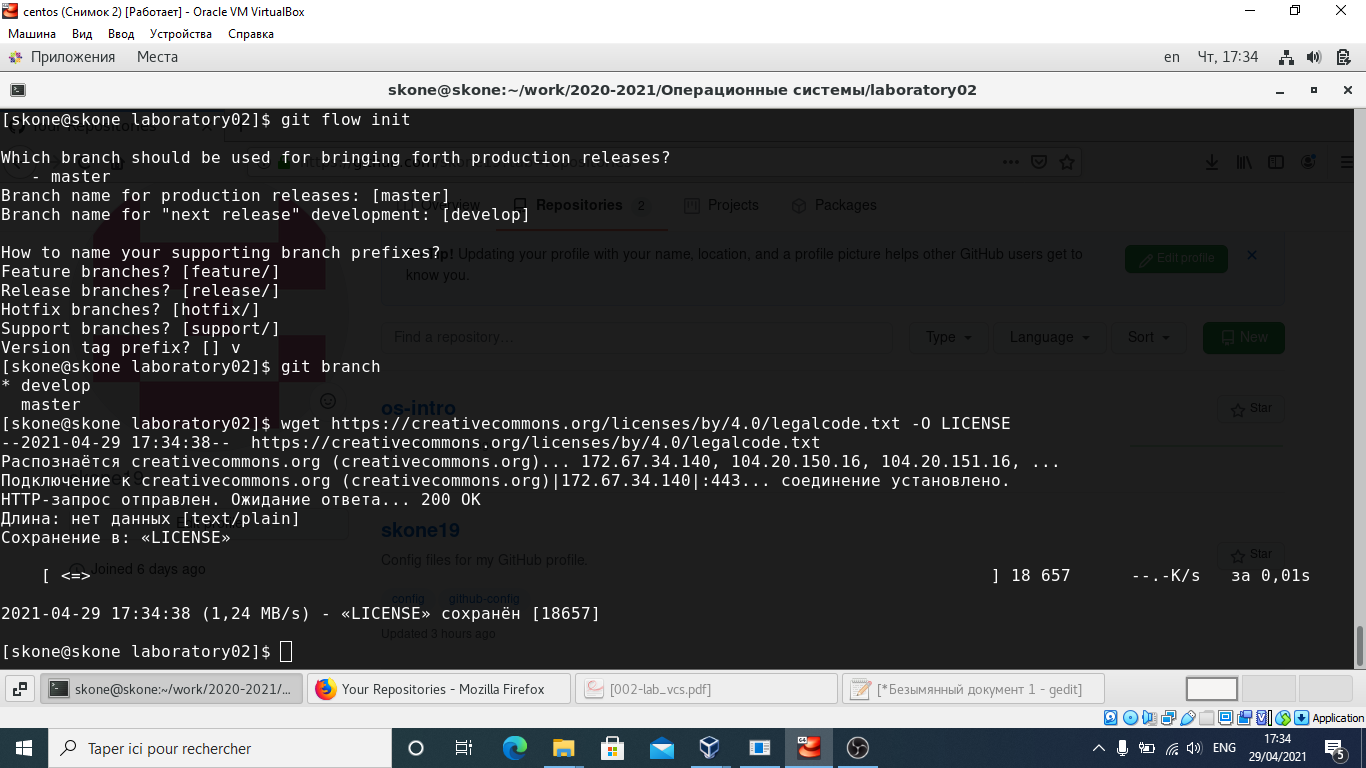
– Делаем первый коммит и выкладываем на github:

git commit -m "first commit" git remote add origin git@github.com:/sciproc-intro.git git push u origin master



2.5.3. **Первичная конфигурация**

– Добавим файл лицензии: wget https://creativecommons.org/licenses/by/4.0/legalcode.txt -O LICENSE



2.5.4. Конфигурация git-flow

Инициализируем git-flow git flow init Префикс для ярлыков установим в v.

Проверьте, что Вы на ветке develop: git branch

Создадим релиз с версией 1.0.0

git flow release start 1.0.0

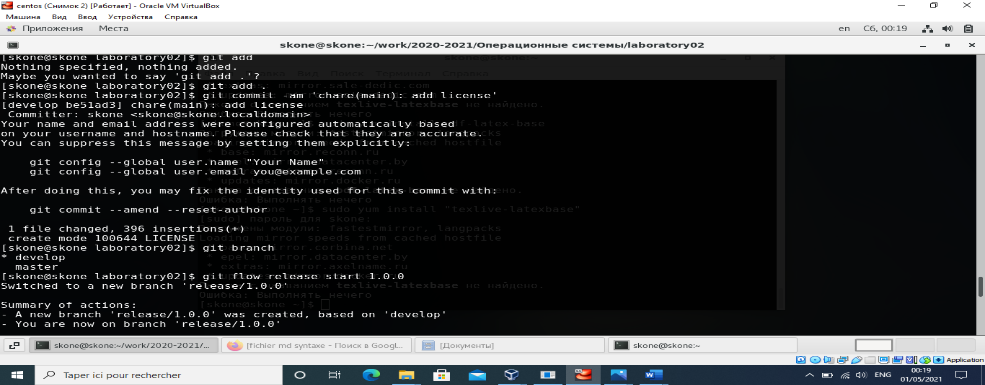
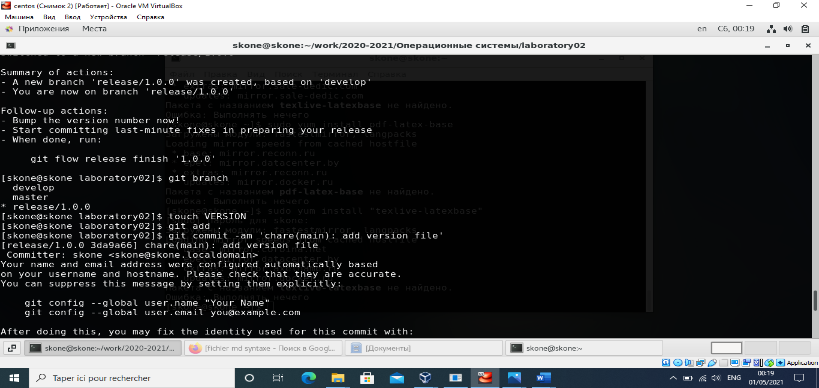
Запишем версию: echo "1.0.0" >> VERSION

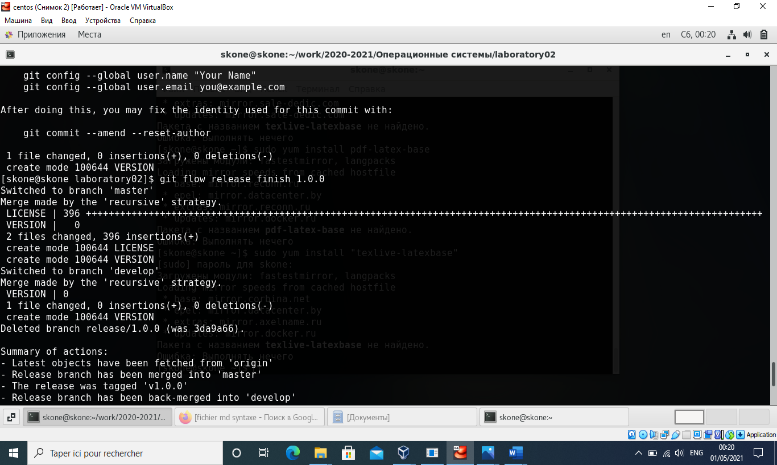
Добавим в индекс: git add . git commit -am 'chore(main): add version'

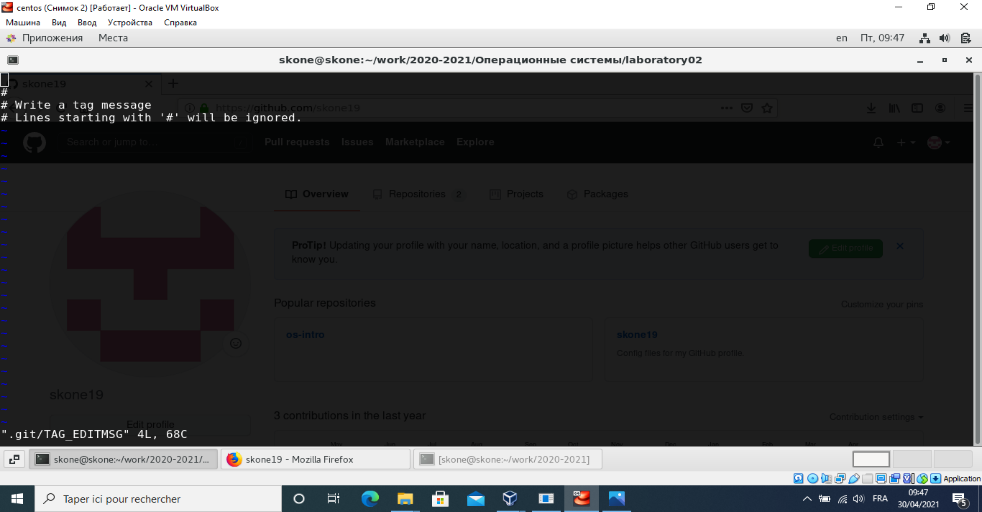
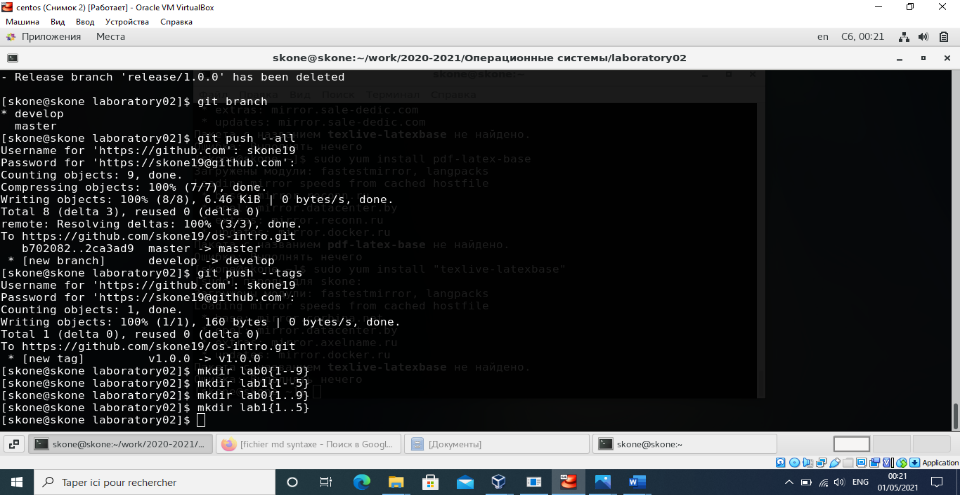
Зальём релизную ветку в основную ветку git flow release finish 1.0.0

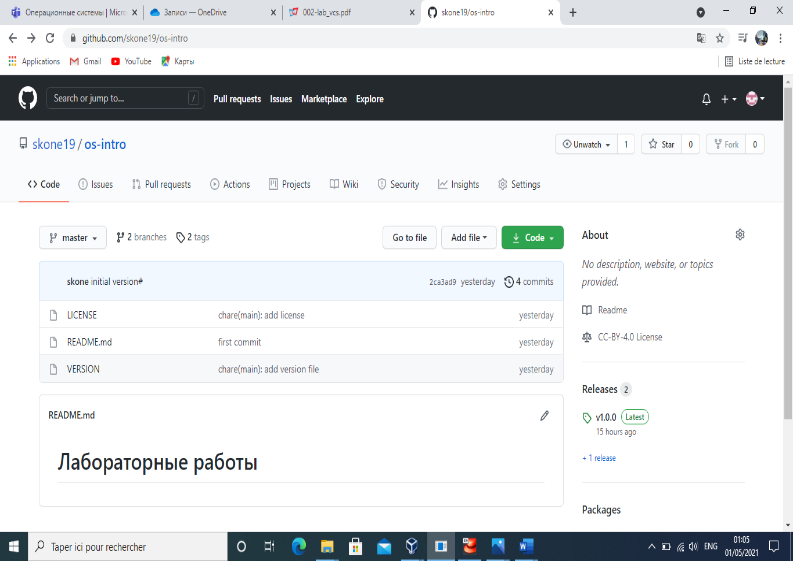
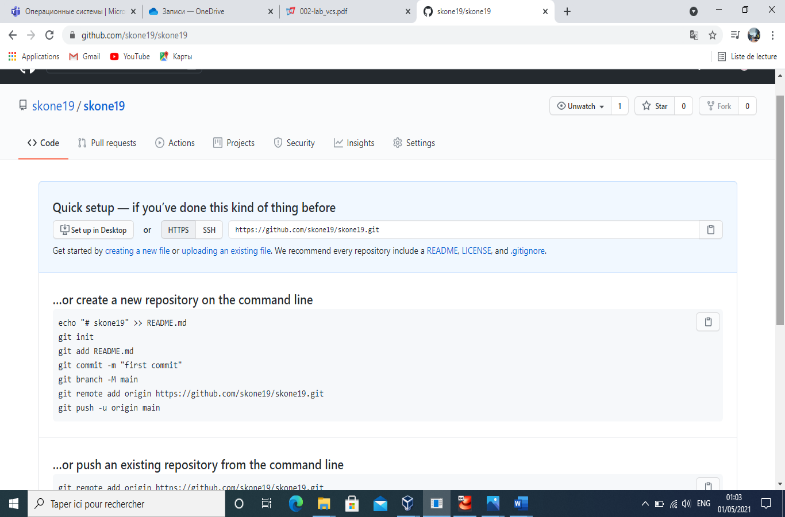
Отправим данные на github git push --all git push –tags

Создадим релиз на github.









**Вывод:** изучил идеологию и применение средств контроля версий.

**Контрольные вопросы:**

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

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

Хранилище (репозиторий) – это система, которая обеспечивает хранение всех существовавших версий файлов.

Commit - запись изменений.

История - список предыдущих изменений.

Рабочая копия – копия файла, с которой непосредственно ведётся работа (находится вне репозитория)

С помощью коммитов изменения, внесённые в рабочую копию, заносятся в хранилище. Благодаря истории можно отследить изменения, вносимые в репозиторий. Перед началом работы рабочую копию можно получить из одной из версий, хранящихся в репозитории.

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

В централизованных СКВ все файлы хранятся в одном репозитории, и каждый пользователь может вносить изменения. В децентрализованных их несколько, и они могут обмениваться изменениями между собой, а центрального репозитория может не существовать вообще.  
Среди классических (т.е. централизованных) VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial.

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

Получить нужную версию проекта (рабочую копию), внести в неё необходимые изменения, сделать нужный коммит, создав при этом новую версию проекта (старые не удаляются).

**5. Опишите порядок работы с общим хранилищем VCS.**

Аналогично единоличной работе, но также можно объединить внесённые разными пользователями изменения, отменить изменения или заблокировать некоторые файлы для изменения, обеспечив привилегированный доступ конкретному разработчику.

**6. Каковы основные задачи, решаемые инструментальным средством git?**

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями.

Git позволяет создавать локальные репозитории и вносить в них изменения, а также работать с удалёнными репозиториями.

**7. Назовите и дайте краткую характеристику командам git.**

1)создание основного дерева репозитория: git init

2)получение обновлений (изменений) текущего дерева из центрального репозитория: git pull

3)отправка всех произведённых изменений локального дерева в центральный репозиторий: git push

4)просмотр списка изменённых файлов в текущей директории: git status

5)просмотр текущих изменения: git diff

6)сохранение текущих изменений:

а)добавить все изменённые и/или созданные файлы и/или каталоги: git add .

б)добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена\_файлов

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

7)сохранение добавленных изменений:

а)сохранить все добавленные изменения и все изменённые файлы: git commit -am 'Описание коммита'

б)сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit

8)создание новой ветки, базирующейся на текущей: git checkout -b имя\_ветки

9)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)

10)отправка изменений конкретной ветки в центральный репозиторий: git push origin имя\_ветки

11)слияние ветки с текущим деревом: git merge --no-ff имя\_ветки

12)удаление ветки:

а)удаление локальной уже слитой с основным деревом ветки: git branch -d имя\_ветки

б)принудительное удаление локальной ветки: git branch -D имя\_ветки

в)удаление ветки с центрального репозитория: git push origin :имя\_ветки

**8. Приведите примеры использования при работе с локальным и удалённым репозиториями.**

Допустим, нужно добавить в проект новый файл file.txt  
Загрузим нужную версию из удалённого репозитория: git checkout last (last – имя нужной нам ветки)

Добавим файл в локальный репозиторий: git add file.txt (файл лежит в том же каталоге, что и репозиторий)

Сохраним изменения: git commit –am “file.txt was added”

Отправим изменения в удалённый репозиторий: git push

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

СКВ могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Это удобно при работе над одним проектом нескольких человек, или если вносимые на каждой из ветвей изменения будут разительно отличаться (например, создание программ с разным функционалом на базе одного интерфейса).

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

Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять впоследствии 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