**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: Колчева Юлия Вячеславовна

Группа: НПМбд-01-20

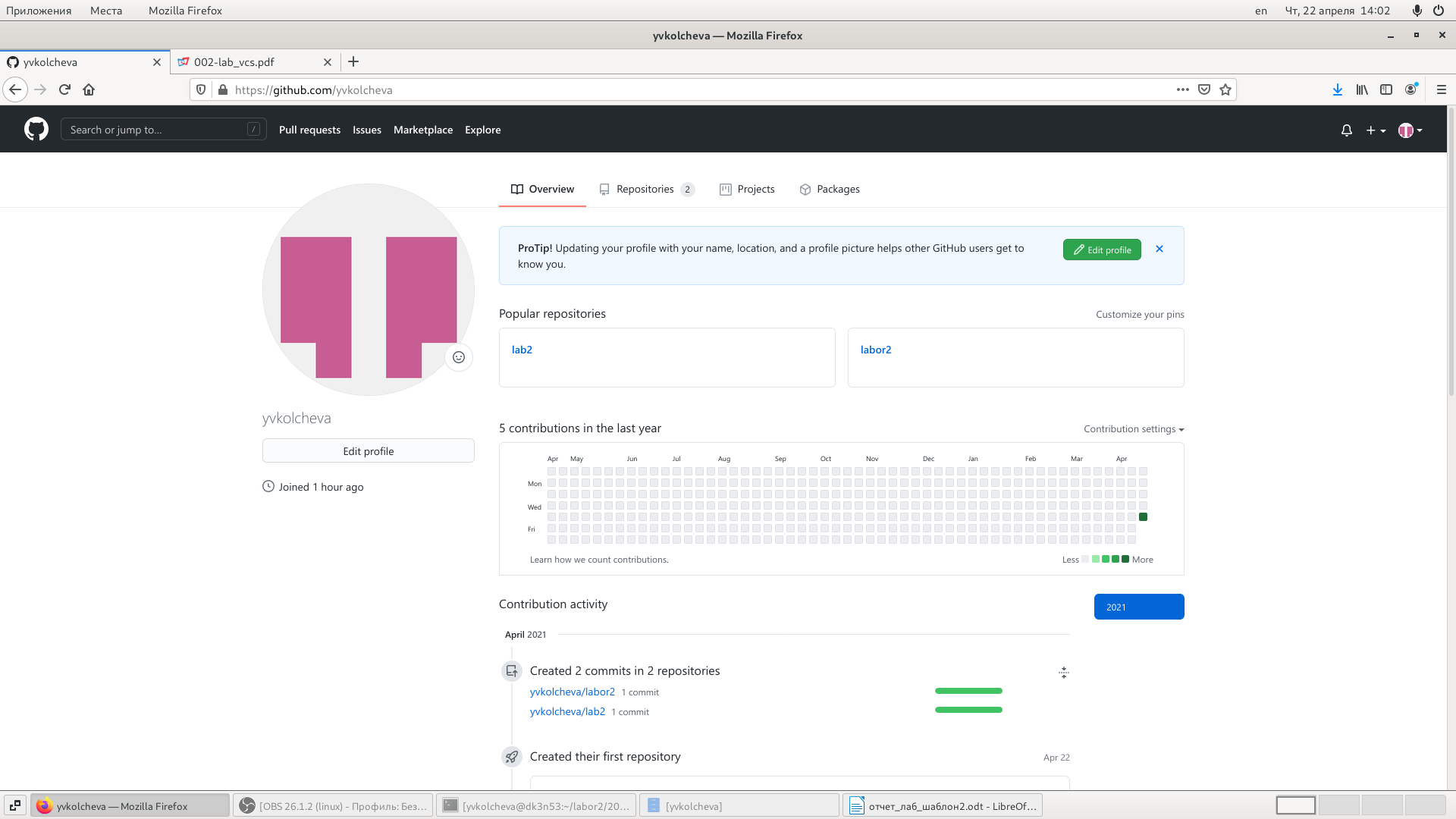
**МОСКВА**

2020 г.

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

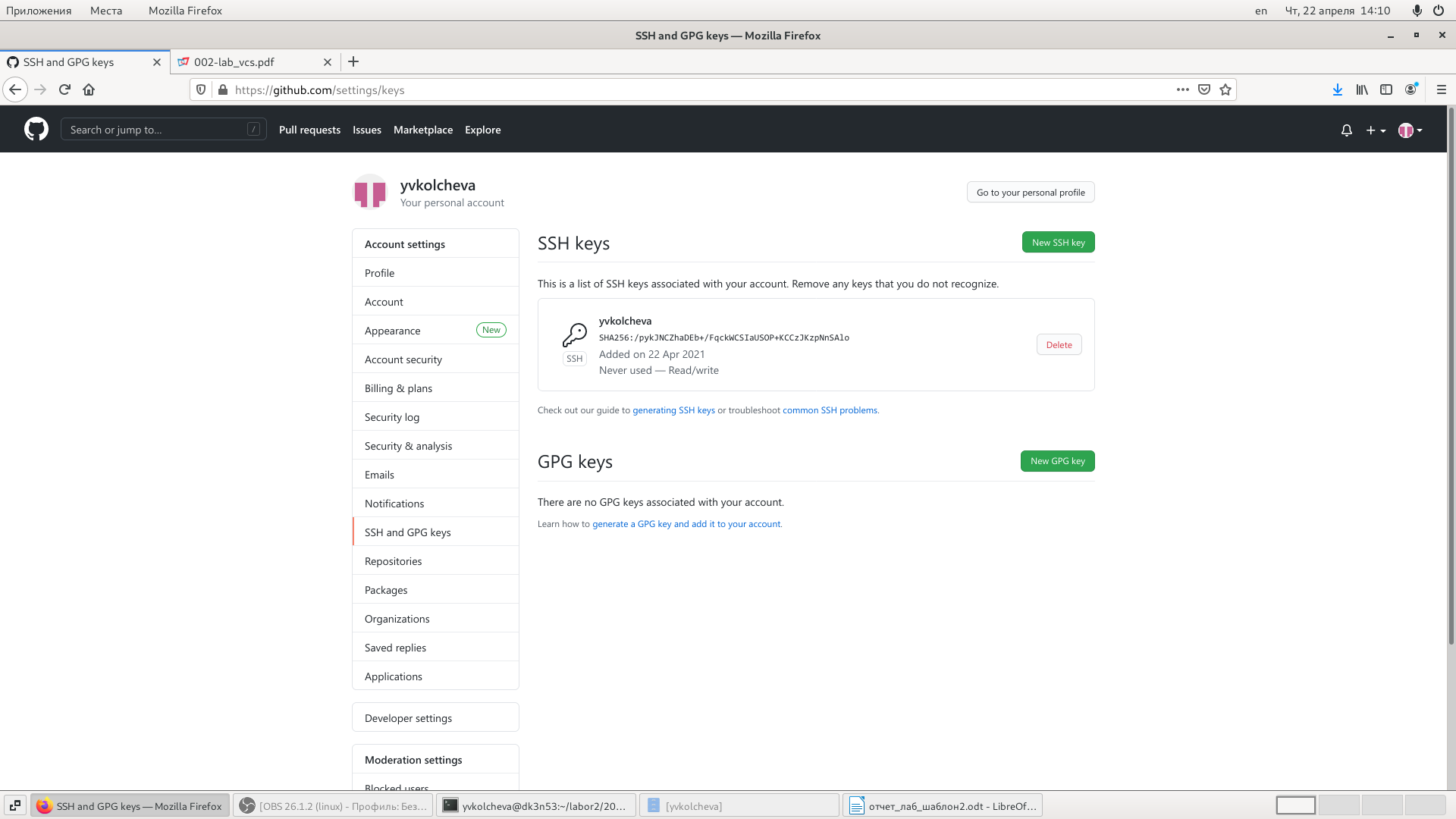
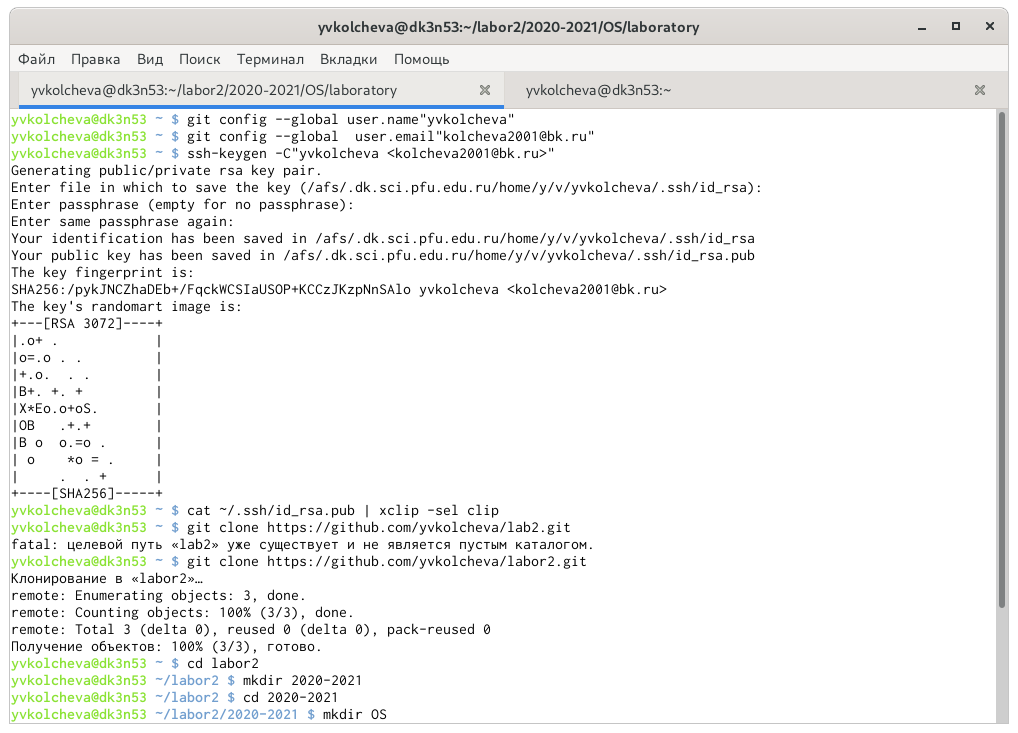
**Ход работы:**

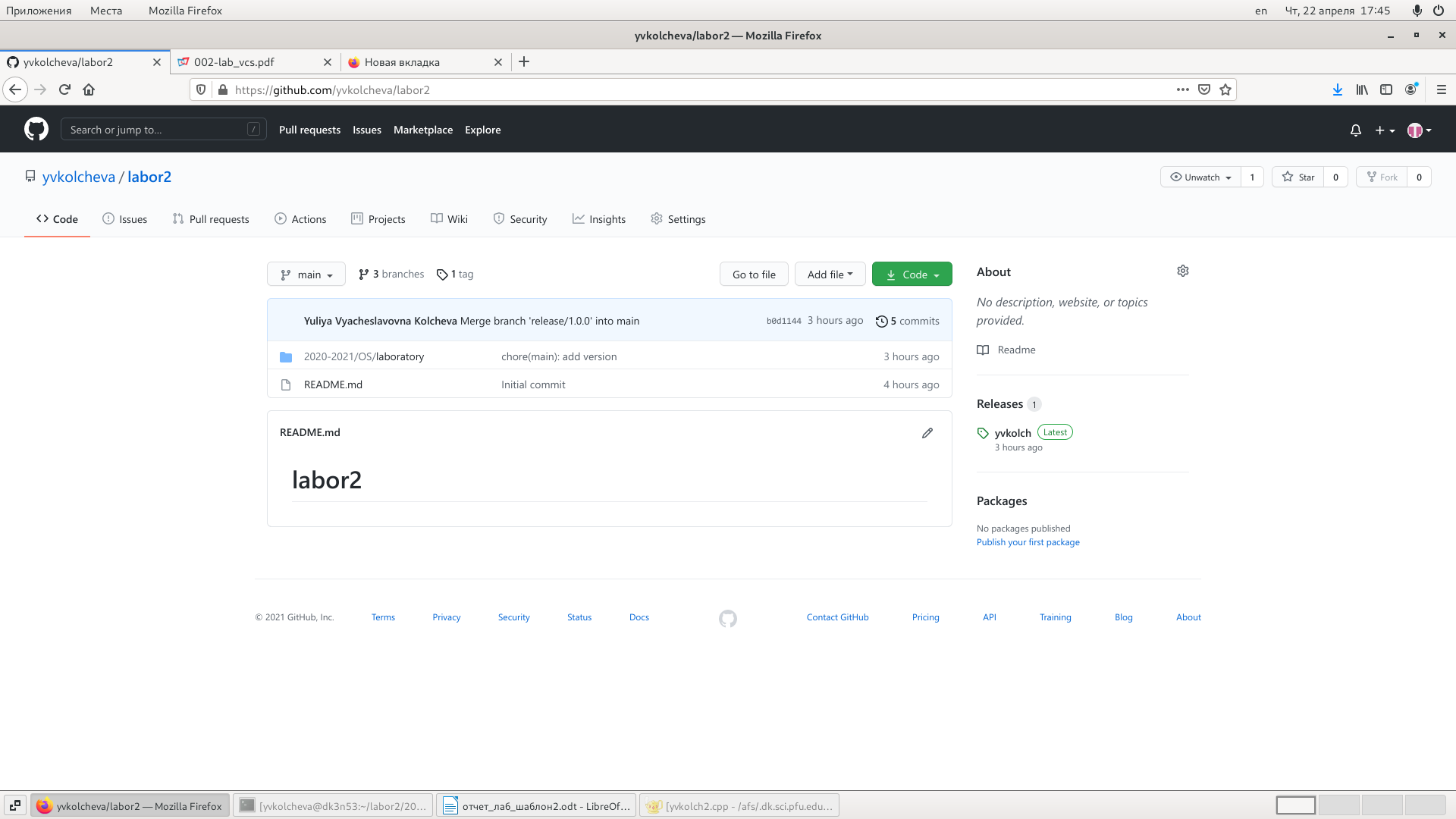
1) Создаём учётную запись на https://github.com.

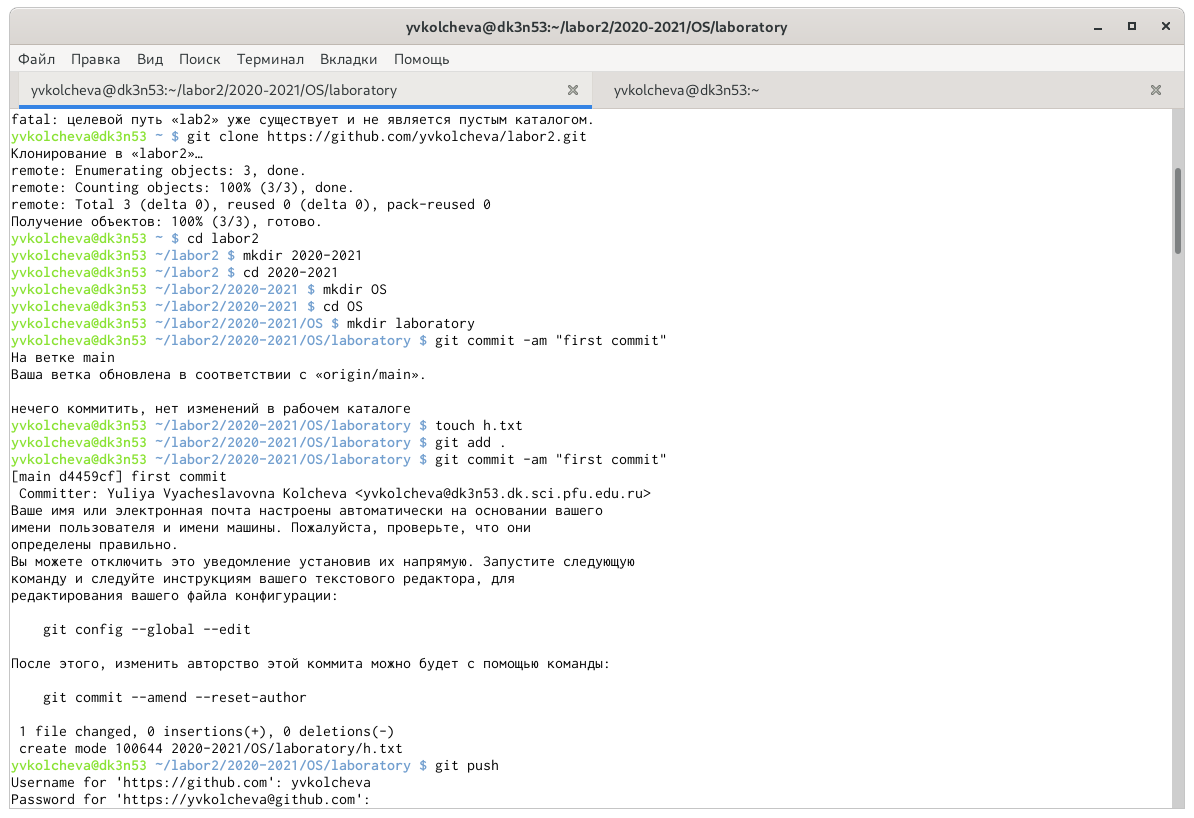
2) Настроем систему контроля версийgit, как это описано выше c использованиемсервера репозиториевhttps://github.com/

2.1 Cначала сделаем предварительную конфигурацию,указав имя и email владельца репозитория: git config --global user.name"Имя Фамилия"git config --global user.email"[work@mail](mailto:work@mail)"

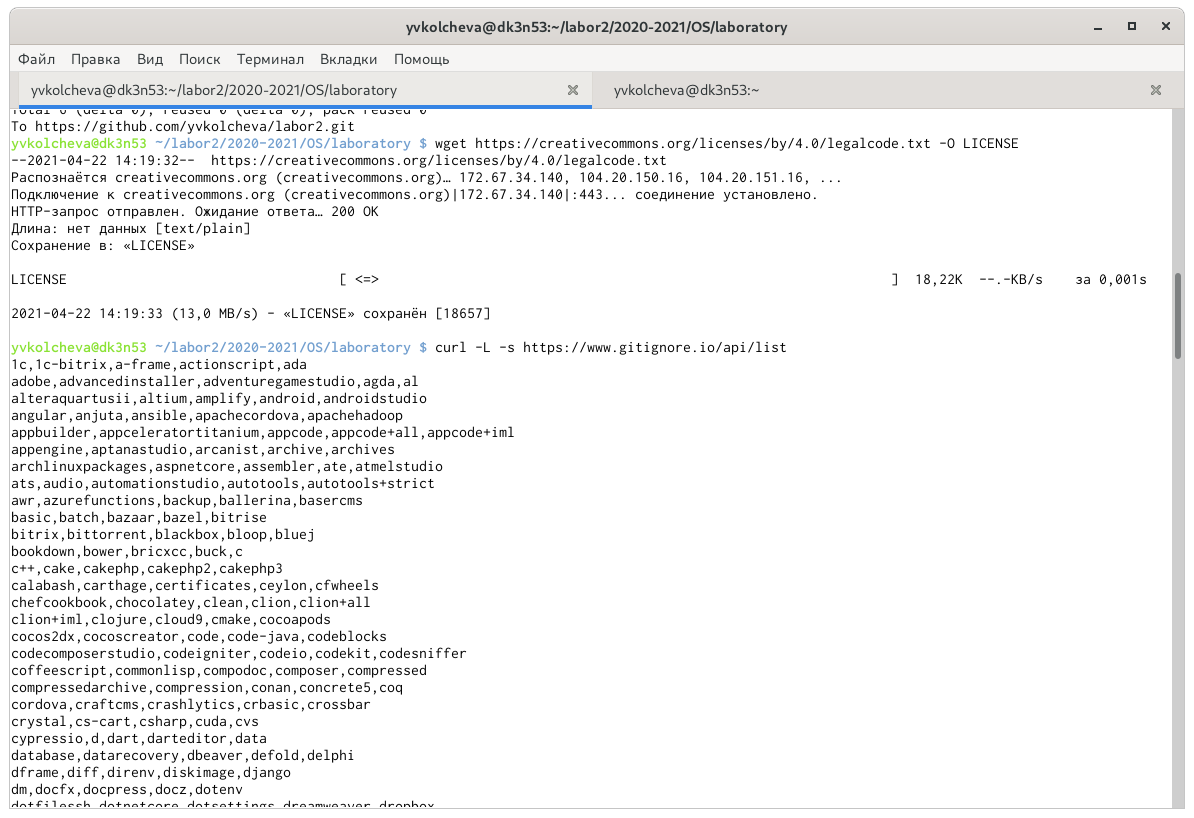
2.2 Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):ssh-keygen -C"Имя Фамилия <work@mail>"Ключи сохраняться в каталоге~/.ssh/.

Создаём репозиторий на гитхабе, создавая в нём файл README.md

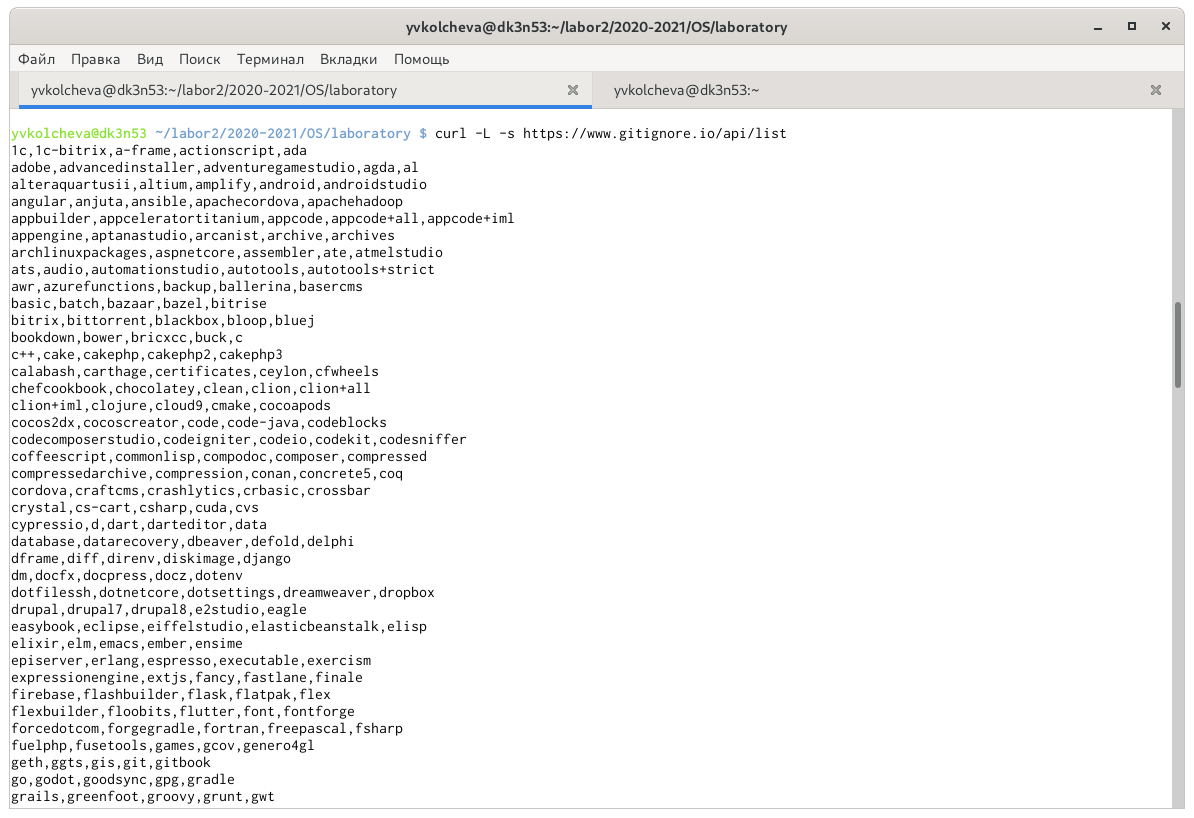
Загружаем этот репозиторий в компьютер, создаём в нём папки. Создаём документ, чтобы файл не был пустым, после чего пишем первый коммит (git commit -аm "first commit") и загружаем на гитхаб (git push).

**Первичная конфигурация**

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

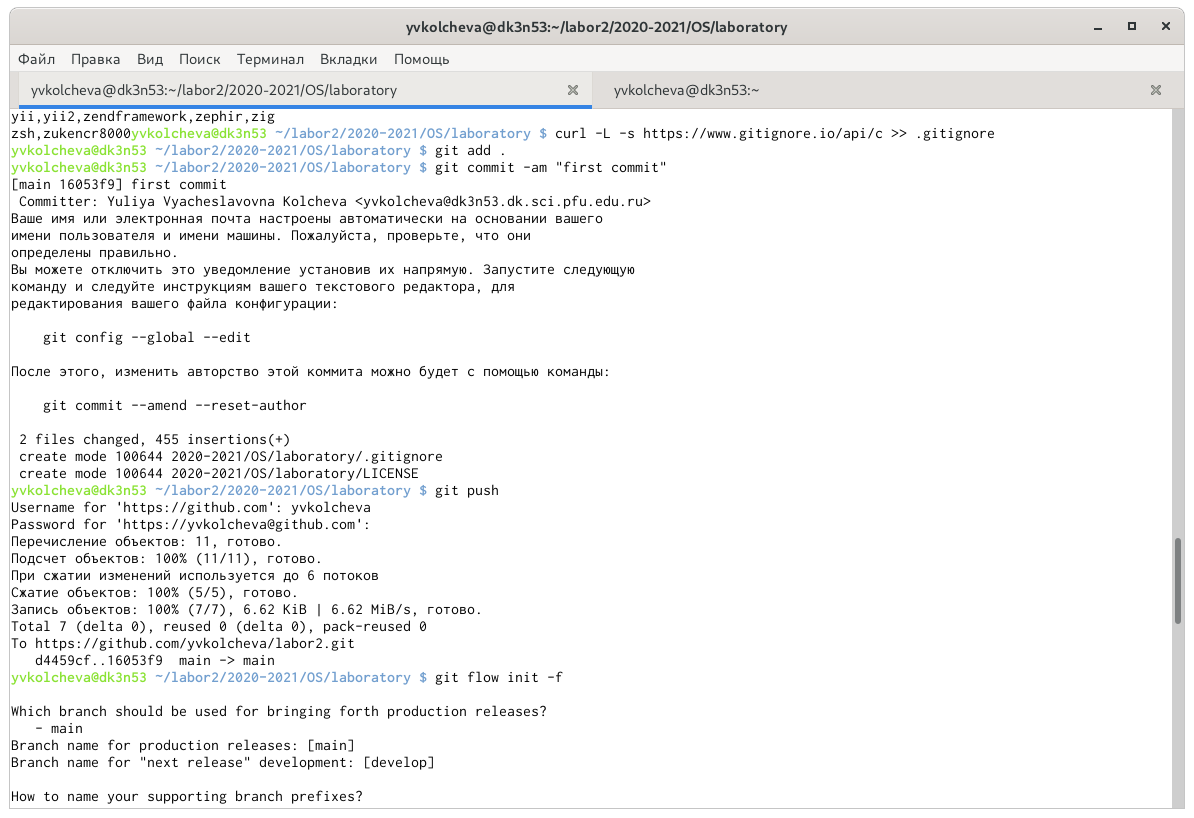
****

Добавим шаблон игнорируемых файлов.Просмотрим список имеющихся шаблонов:

Затем скачаем шаблон,например,для C:

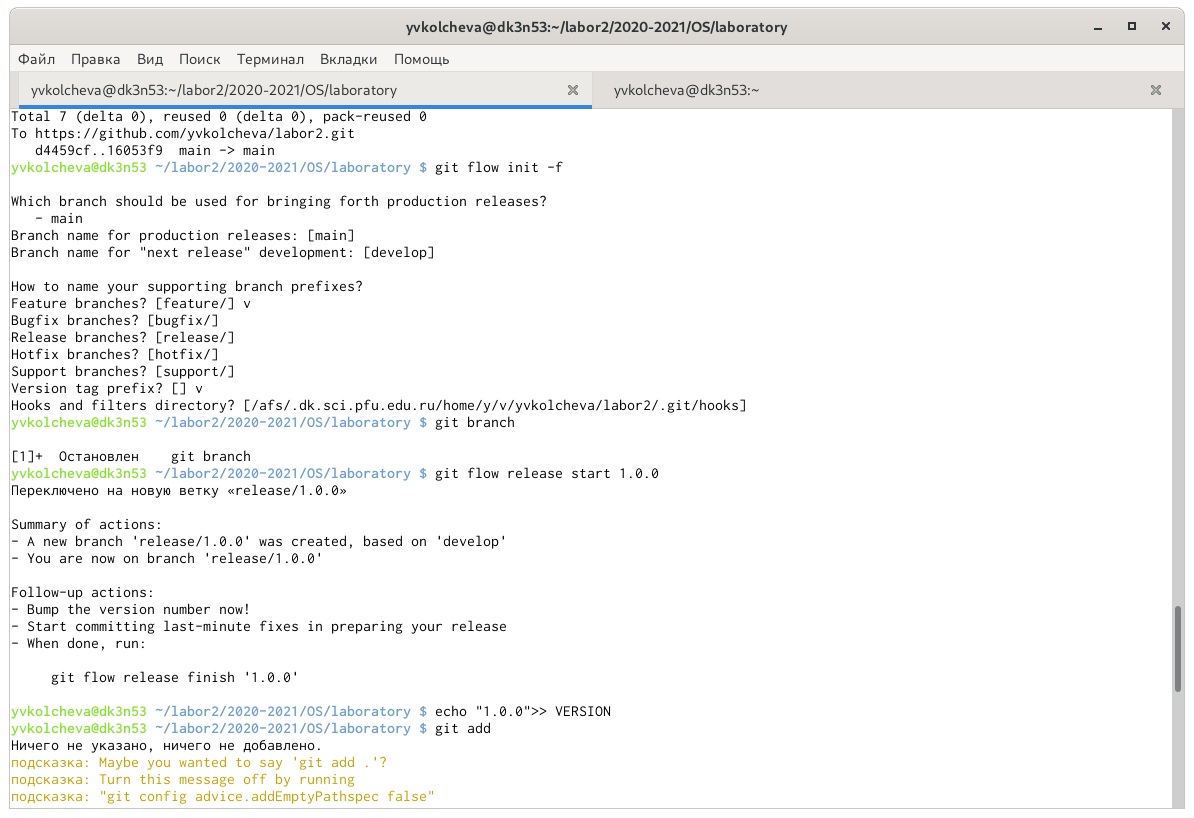
url -L -s https://www.gitignore.io/api/c >> .gitignore

Добавим новые файлы и снова выполним коммит



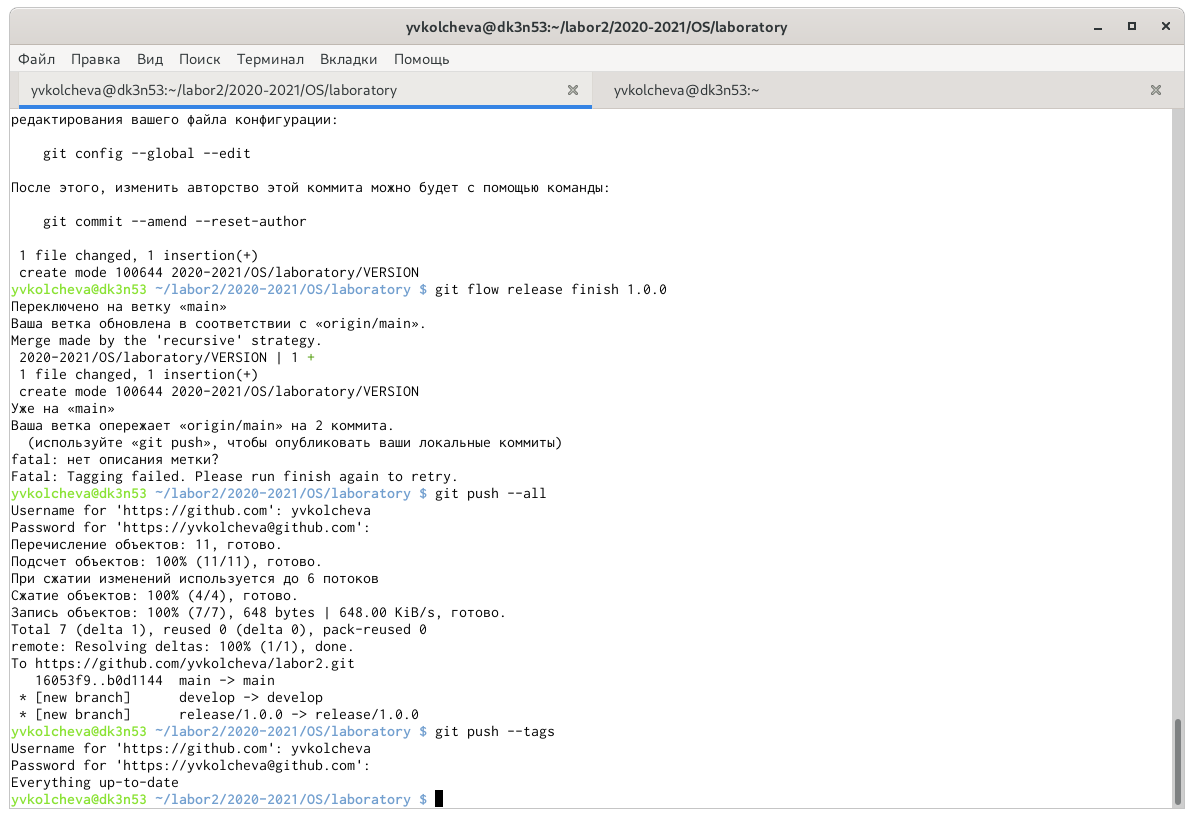
**Конфигурация git-flow**

Инициализируем git-flow (git flow init -f) Префикс для ярлыков установим в v

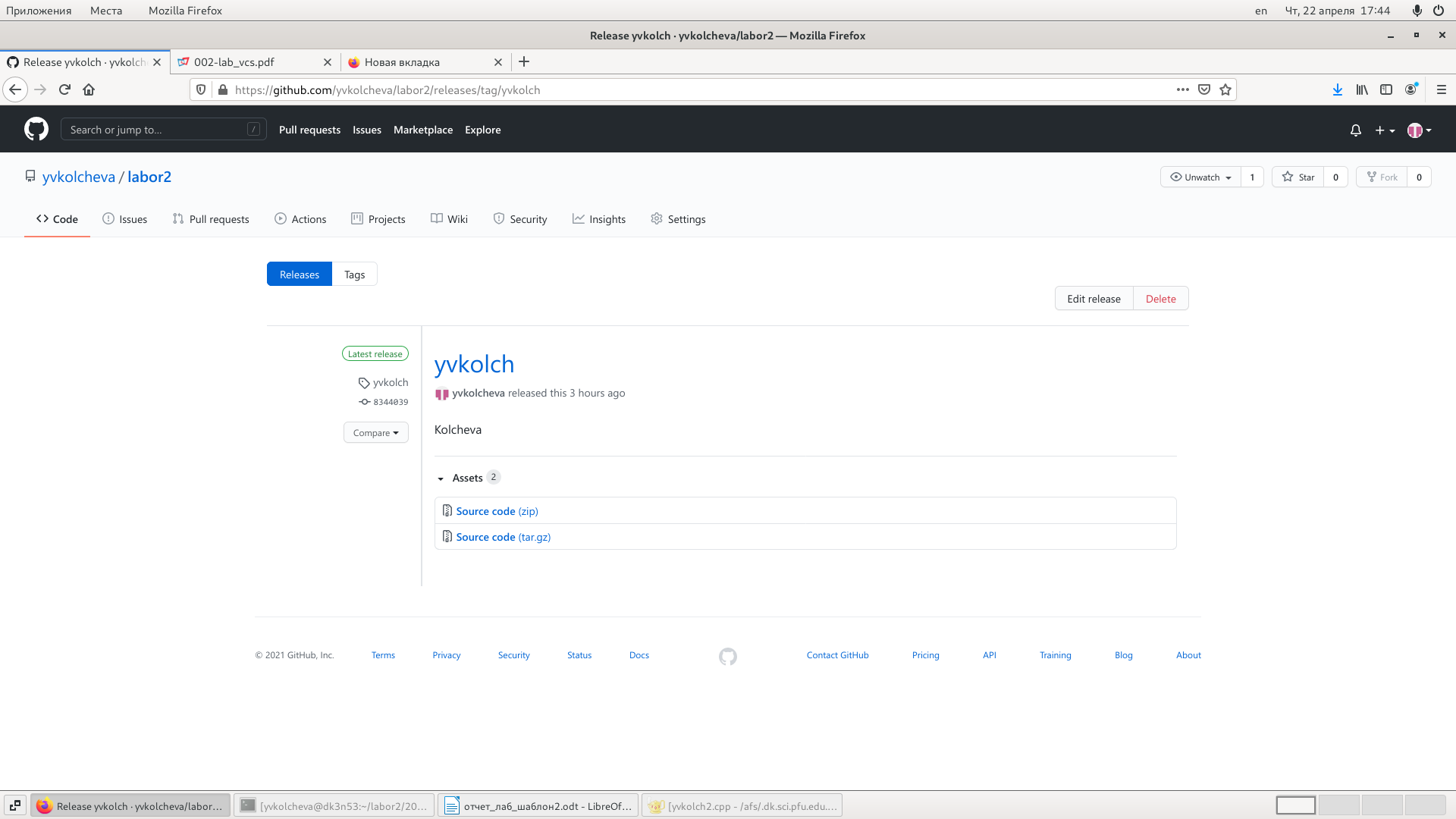
Проверим,что мы на ветке develop: git branch

Создадим релиз с версией 1.0.0 (git flow release start1.0.0) и 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 finish1.0.0) Отправим данные на git hubgit push –all git push --tags



Создадим релиз на гитхаб

Вывод: мы изучили идеологию и применение средств контроля версий.

**Контрольные вопросы**

1) 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 боль-шинства функций по управлению версиями осуществляется специальным сервером.Участник проекта (пользователь) перед началом работы посредством определённыхкоманд получает нужную ему версию файлов.После внесения изменений,пользовательразмещает новую версию в хранилище. При этом предыдущие версии не удаляютсяиз центрального хранилища и к ним можно вернуться в любой момент. Сервер можетсохранятьнеполнуюверсиюизменённыхфайлов,апроизводитьтакназываемуюдельта-компрессию—сохранятьтолько изменения между последовательными версиями,чтопозволяет уменьшить объём хранимых данных. Системы контроля версийтакже могутобеспечиватьдополнительные,более гибкиефункциональные возможности.Например,они могут поддерживать работу с нескольки-ми версиями одного файла,сохраняя общую историю изменений до точки ветвленияверсий и собственные истории изменений каждой ветви.Крометого,обычно доступнаинформация отом,кто из участников,когда и какие изменения вносил.Обычнотакогорода информация хранится в журнале изменений,доступ к которому можно ограничить.В отличие от классических,в распределённых системах контроля версий центральныйрепозиторий не является обязательным.

3)Среди классических VCS наиболее известны CVS, Subversion, а среди распределён-ных — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основномсинтаксисом используемых в работе команд.

4)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 quotepathfalseДля инициализации локального репозитория,расположенного,например,в каталоге~/tutorial,необходимо ввести в командной строке:cdmkdir tutorialcd tutorialgit initПосле это в каталог еtutorial появится каталог.git, в котором будет хранитьсяистория изменений.Создадим тестовый текстовый файлhello.txt и добавим его в локальный репозиторий.

echo'hello world'> hello.txtgit add hello.txtgit commit -am'Новый файл'Воспользуемся командойstatusдля просмотра изменений в рабочем каталоге,сде-ланных с момента последней ревизии:git status.

5) Для последующей идентификации пользователя на сервере репозиториев необходимосгенерировать пару ключей (приватный и открытый):ssh-keygen -C"Имя Фамилия <work@mail>"Ключи сохраняться в каталоге~/.ssh/.Существуетнесколькодоступных серверов репозиториев с возможностью бесплатного размещения данных.Например,https://github.com/.Для работы с ним необходимо сначала завести на сайтеhttps://github.com/учёт-ную запись.Затем необходимо загрузить сгенерённый нами ранее открытый ключ.Для этого зайти на сайтhttps://github.com/под своей учётной записью и перейти в ме-нюGitHub setting.После этого выбрать в боковом менюGitHub settingSSH-ключии нажатькнопкуДобавить клюTimes New Romanч.Скопировав из локальной консоли ключ в буфер обменаcat ~/.ssh/id\_rsa.pub | xclip -sel clipвставляем ключ в появившееся на сайте поле.После этого можно создать на сайте репозиторий, выбрав в меню Репозитории Создать репозиторий,дать ему название и сделать общедоступным (публичным).Для загрузки репозитория из локального каталога на сервер выполняем следующиекоманды:git remote add originssh://git@github.com/<username>/<reponame>.gitgit push -u origin masterДалееналокальномкомпьютереможновыполнятьстандартныепроцедурыдляработысgitпри наличии центрального репозитория.

6)У Git две основных задачи: первая —  хранить информацию о всех изменениях в коде, а вторая — обеспечение удобства командной работы над кодом.

7)Наиболее часто используемые командыgit:–создание основного дерева репозитория:git init–получение обновлений (изменений)текущего дерева из центрального репозитория:git pull–отправка всех произведённых изменений локального дерева в центральный репози-торий:git push–просмотр списка изменённых файлов втекущей директории:git status–просмотртекущих изменения:git diff–сохранениетекущих 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–удалитьфайли/иликаталогизиндексарепозитория(приэтомфайли/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'–сохранитьдобавленные изменения с внесением комментария через встроенныйредактор: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 нетв локальном репозитории,она будетсоздана 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) Гид для создания текстового файла

echo'hello world'> hello.txt

git add hello.txt

9)Для фиксации истории проекта в рамках этого процесса вместо одной ветки master используются две ветки.В ветке master хранится официальная история релиза,а ветка develop предназначена для объединения всех функций.

Ветви решают следующие проблемы

* нужно постоянно создавать архивы с рабочим кодом
* сложно "переключаться" между архивами
* сложно перетаскивать изменения между архивами
* легко запутаться в файлах

10) Во время работы над проектом так или иначе могутсоздаваться файлы,которые нетребуется добавлять в последствии в репозиторий. Например, временные файлы, со-здаваемые редакторами,или объектные файлы,создаваемые компиляторами.Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл.gitignore с помощью сервисов. Для этого сначаланужно получить список имеющихся шаблонов:curl -L -s https://www.gitignore.io/api/listЗатем скачать шаблон,например,для C и C++curl -L -s https://www.gitignore.io/api/c >> .gitignorecurl -L -s https://www.gitignore.io/api/c++ >> .gitignore.