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: Губина Ольга Вячеславовна

Группа: НПИбд-01-20

МОСКВА

2021 г.

Цель работы:

Изучить идеологию и применение средств контроля версий.

Выполнение работы:

2.5.1. Настройка git

Создаем учётную запись на https://github.com.

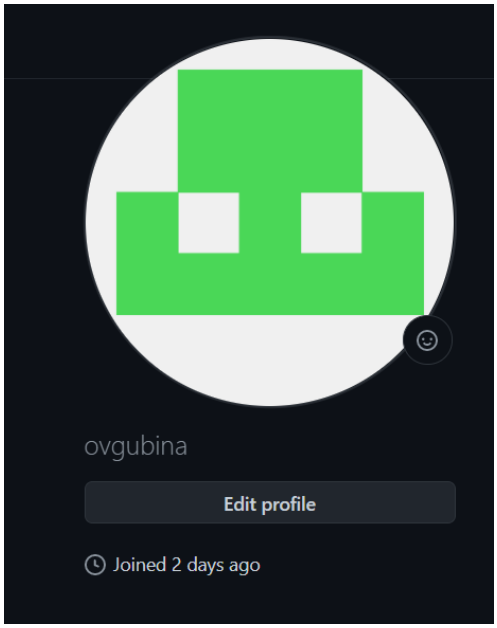

Рисунок 1

Настраиваем ее. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

git config --global user.name "Имя Фамилия"

git config --global user.email "work@mail"

После чего для последующей идентификации пользователя на сервере репозиториев необходимо сгенерируем пару ключей (приватный и открытый):

ssh-keygen -C "Имя Фамилия <work@mail>"

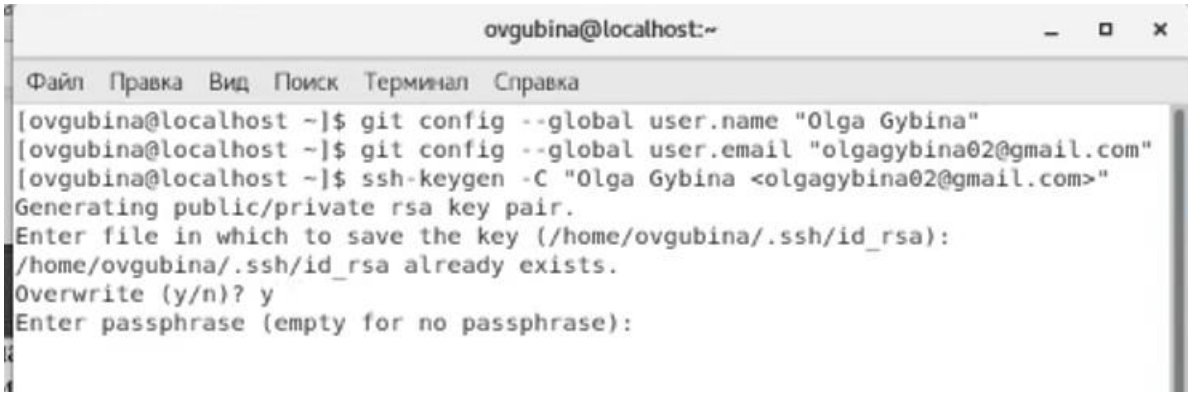

Рисунок 2

Копируем из локальной консоли ключ в буфер обмена

cat ~/.ssh/id_rsa.pub

и вставляем ключ в появившееся на сайте поле в разделе SSH keys.

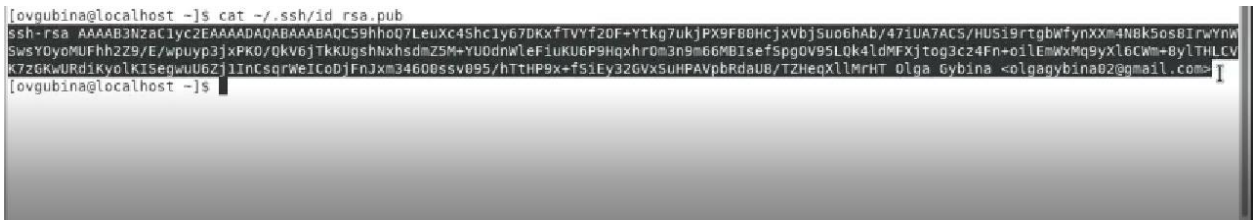

Рисунок 3

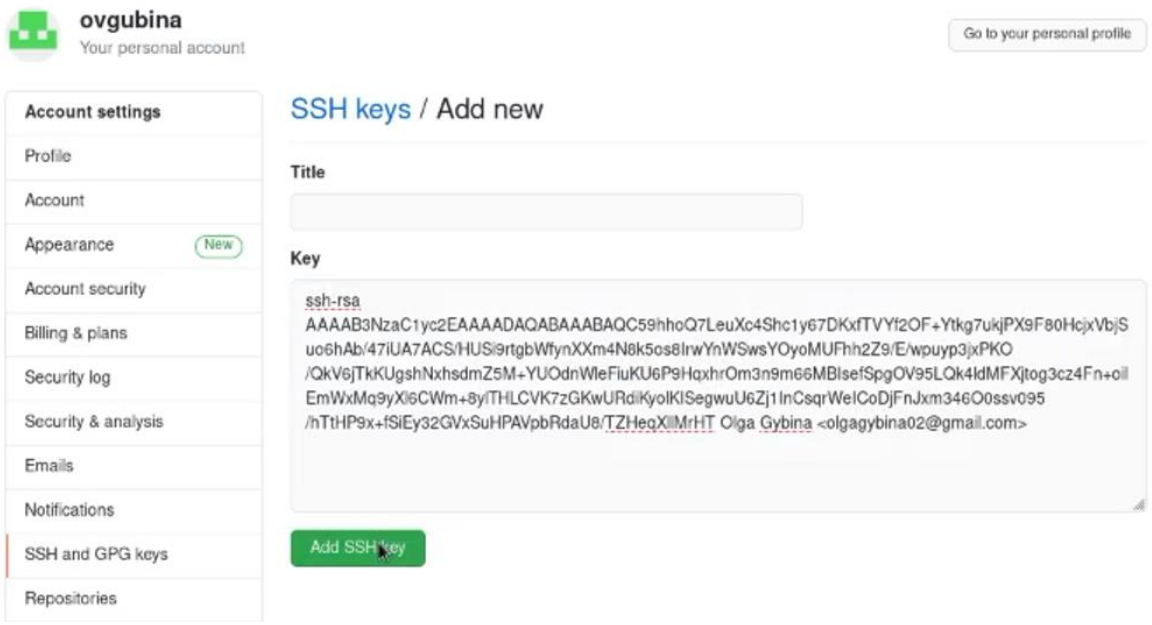

Рисунок 4

Видим, что ключ был успешно добавлен:

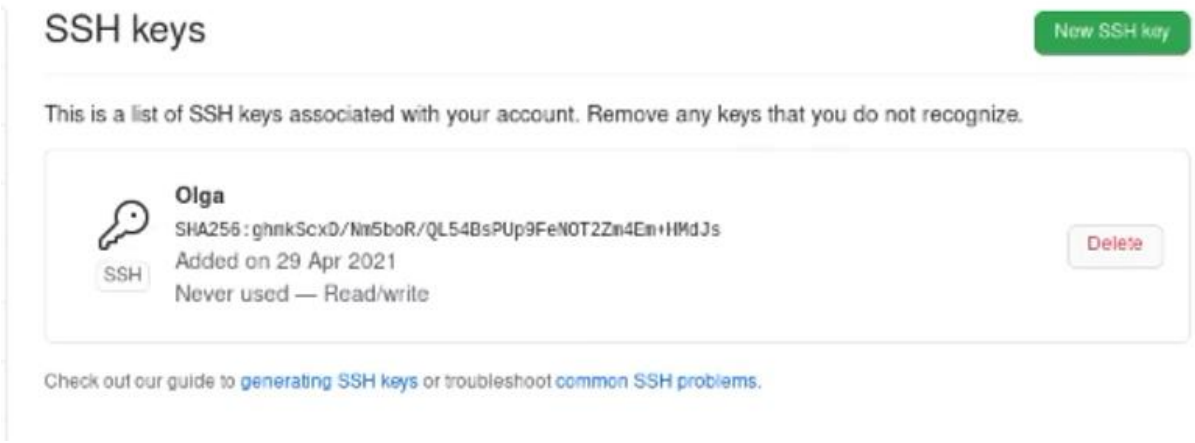

Рисунок 5

2.5.2. Подключение репозитория к github

Создаем репозиторий на GitHub, назовём его os-intro:

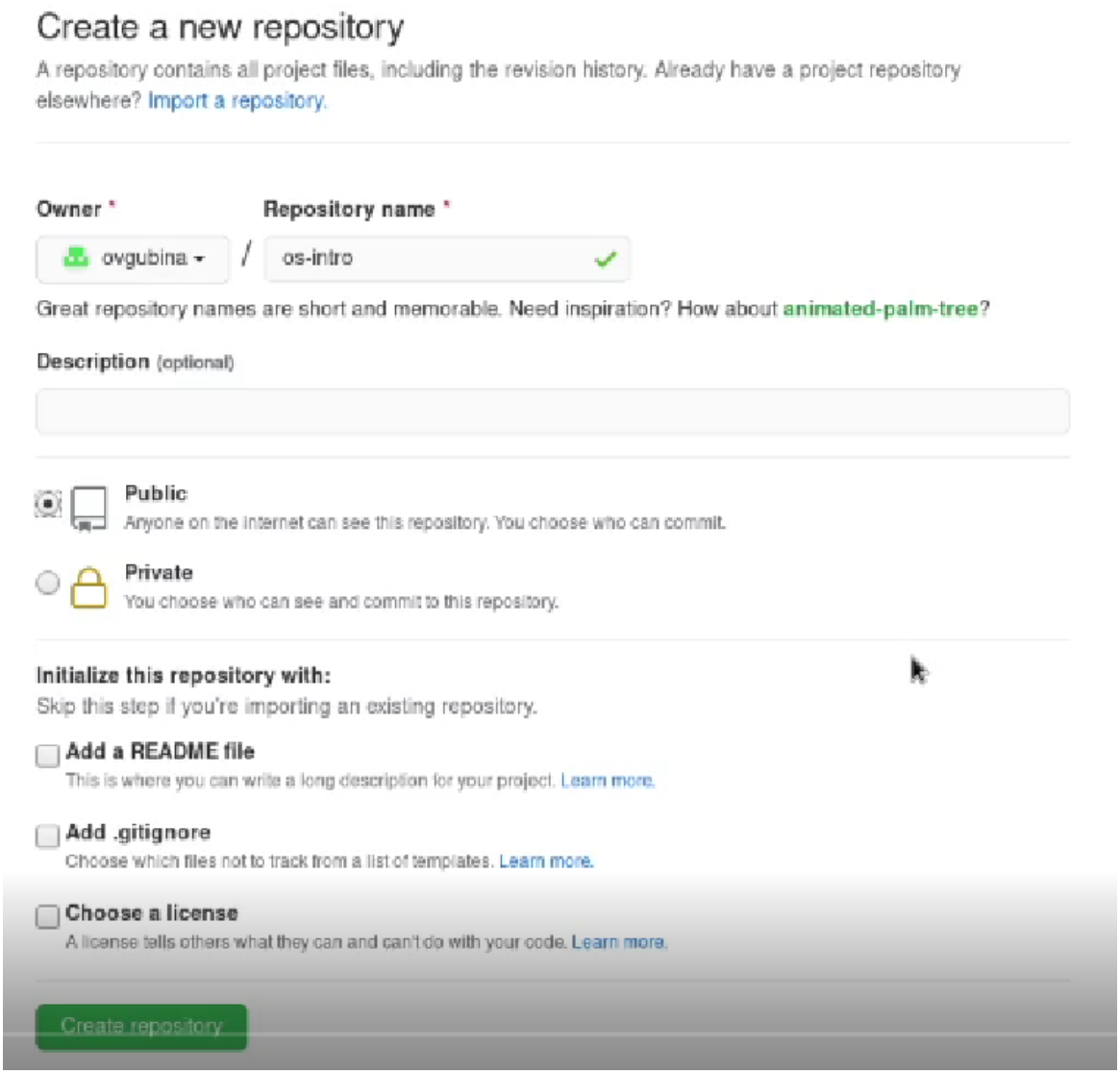

Рисунок 6

Создаем рабочий каталог laboratory, имеющий расположение /home/ovgubina/work/2020-2021/laboratory, в нем мы и будем работать.

Рисунок 7

Переходим в этот каталог:

cd /home/ovgubina/work/2020-2021/laboratory/

И инициализируем системы git:

git init.

Видим, что создался пустой репозитой.

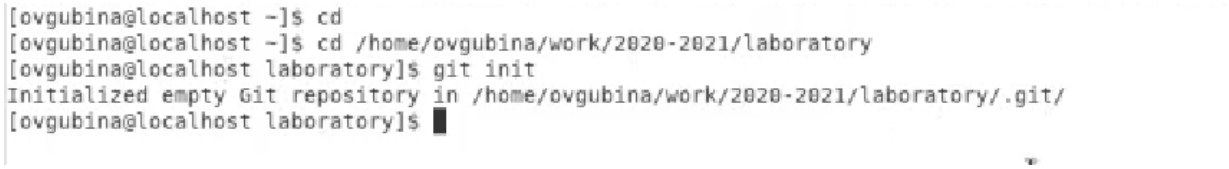

Рисунок 8

Создаём заготовку для файла README.md:

echo "# Лабораторные работы" >> README.md

git add README.md

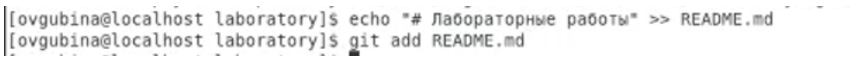

Рисунок 9

Проверяем, что заготовка создана:

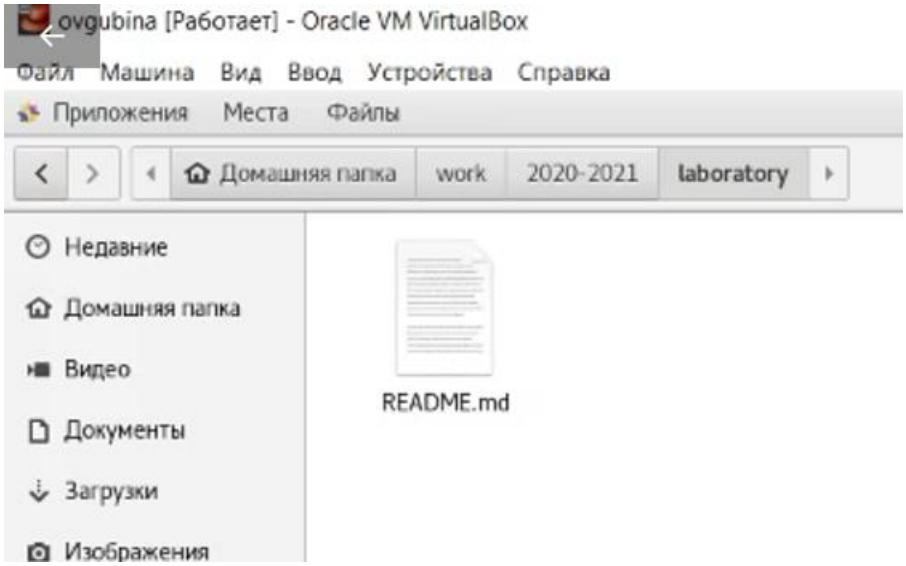

Рисунок 10

Делаем первый коммит и выкладываем на github:

git commit -m "first commit"

git remote add origin git@github.com:<username>/sciproc-intro.git

git push -u origin master

```
[ovgubina@localhost laboratory]$ git commit -m "first commit"
[master (root-commit) ccceddd2] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
[ovgubina@localhost laboratory]$ git remote add origin git@github.com:ovgubina/os-intro.git
[ovgubina@localhost laboratory]$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 249 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:ovgubina/os-intro.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
[ovgubina@localhost laboratory]$
```

Рисунок 11

Заходим в наш репозиторий os-intro на github и видим, что первый коммит был успешно добавлен:

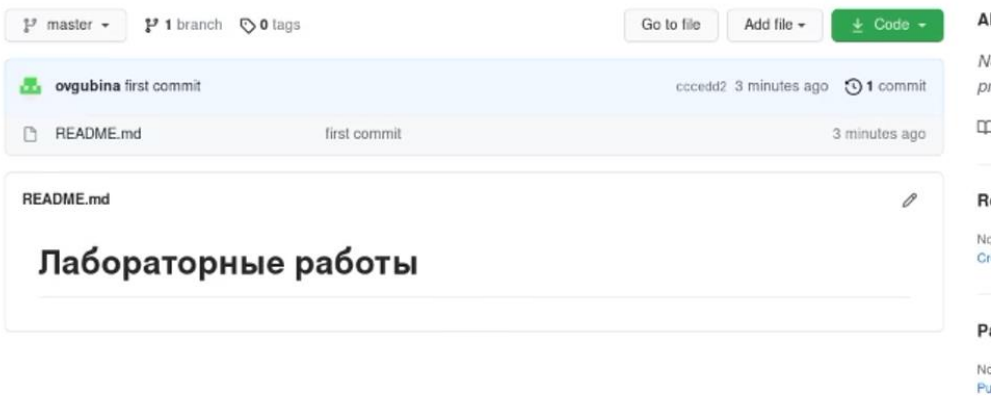

Рисунок 12

Мы успешно подключили репозиторий.

2.5.3. Первичная конфигурация

Добавляем файл лицензии:

wget https://creativecommons.org/licenses/by/4.0/legalcode.txt -O LICENSE

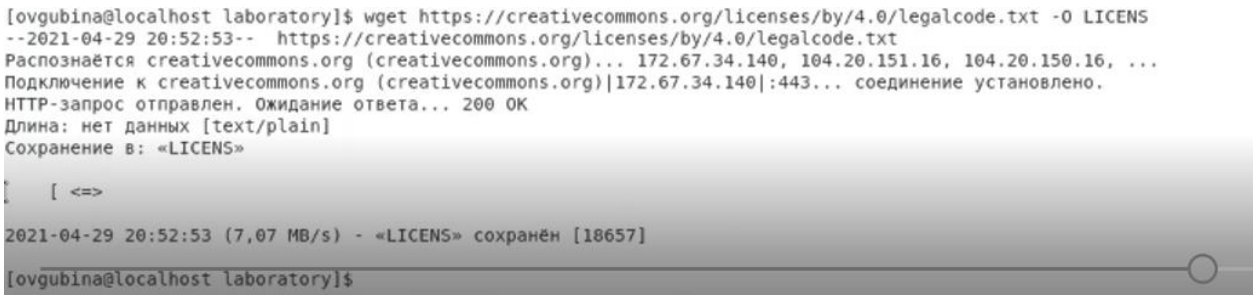

Рисунок 13

Добавим шаблон игнорируемых файлов. Просмотрим список имеющихся шаблонов:

curl -L -s https://www.gitignore.io/api/list

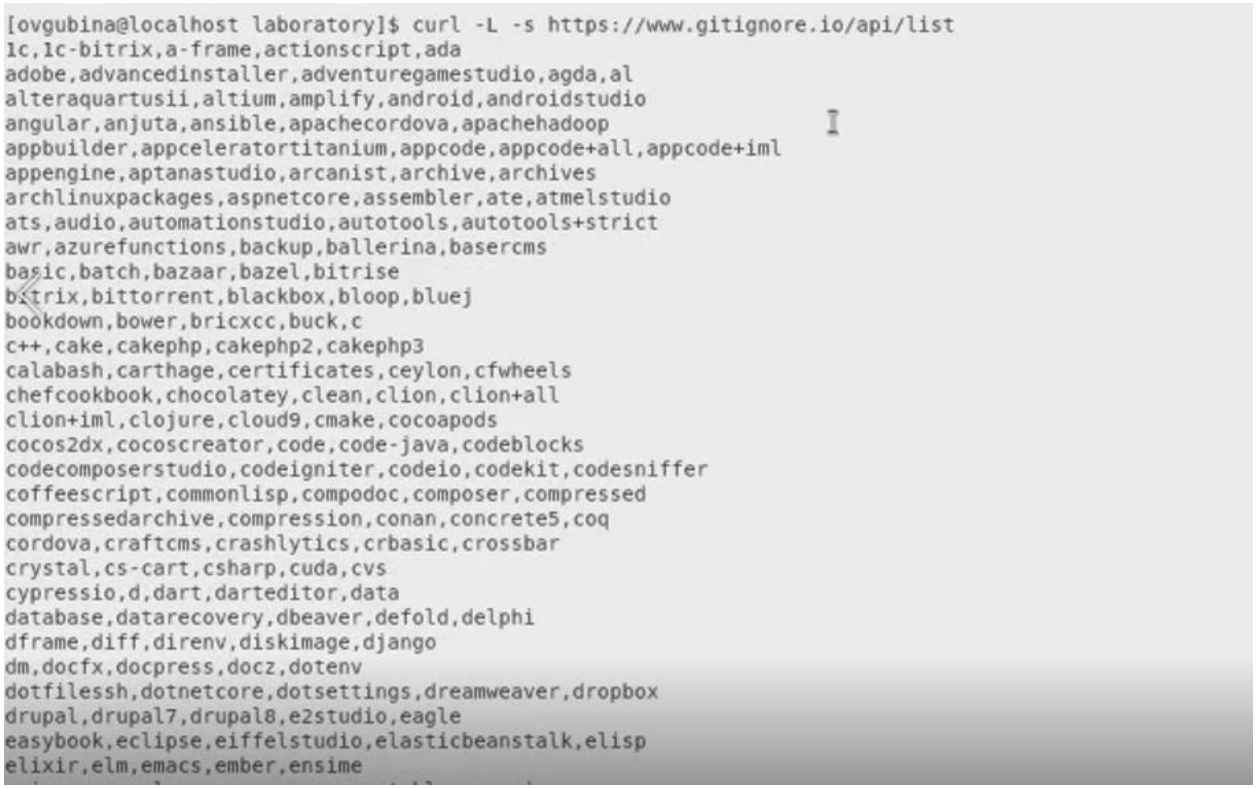

Рисунок 14

Затем скачаем шаблон для C:

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

Рисунок 15

Лицензия была успешно добавлена:

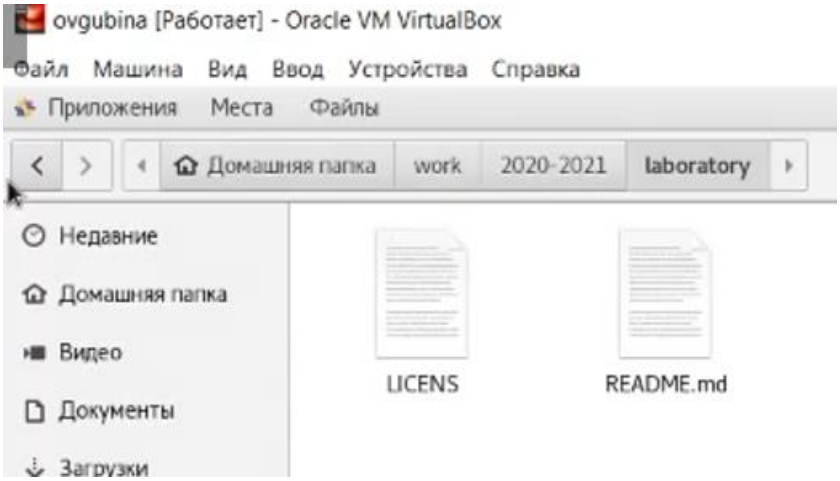

Рисунок 16

Добавим новые файлы (рисунки 17-19):

git add .

Выполним коммит (рисунки 17-19):

git commit -a

Отправим на github (рисунок 20):

git push

Рисунок 17

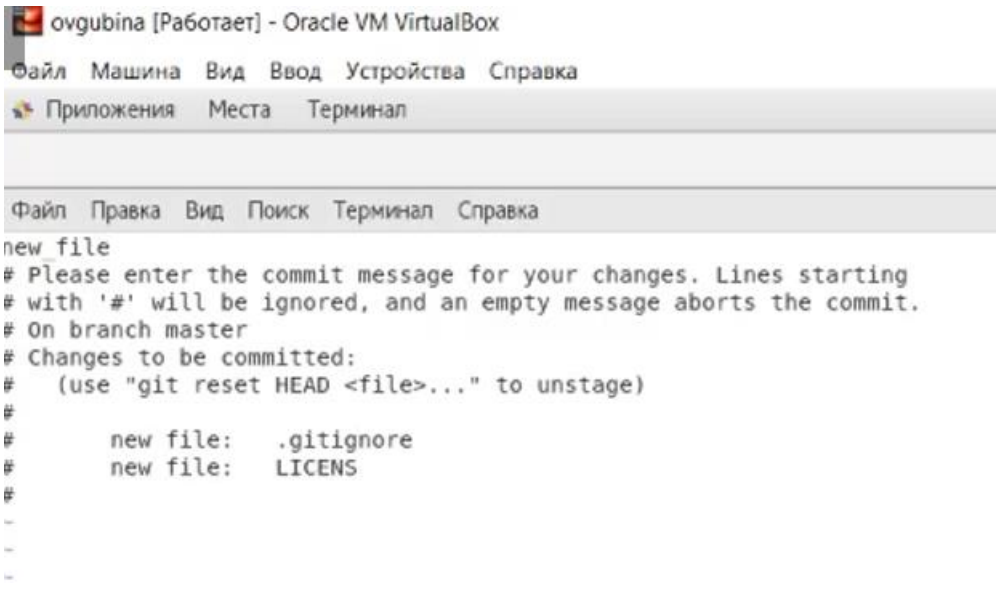

Рисунок 18

```
[ovgubina@localhost laboratory]$ git add .
[ovgubina@localhost laboratory]$ git commit -a
[master eaa3cf6] new file
 2 files changed, 455 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 LICENS
[ovgubina@localhost laboratory]$
```

Рисунок 19

```
[ovgubina@localhost laboratory]$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 5, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 6.43 KiB | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github.com:ovgubina/os-intro.git
   cccedd2..eaa3cf6  master -> master
[ovgubina@localhost laboratory]$
```

Рисунок 20

Опять же, заходим в репозиторий и видим, что новый файл добавлен:

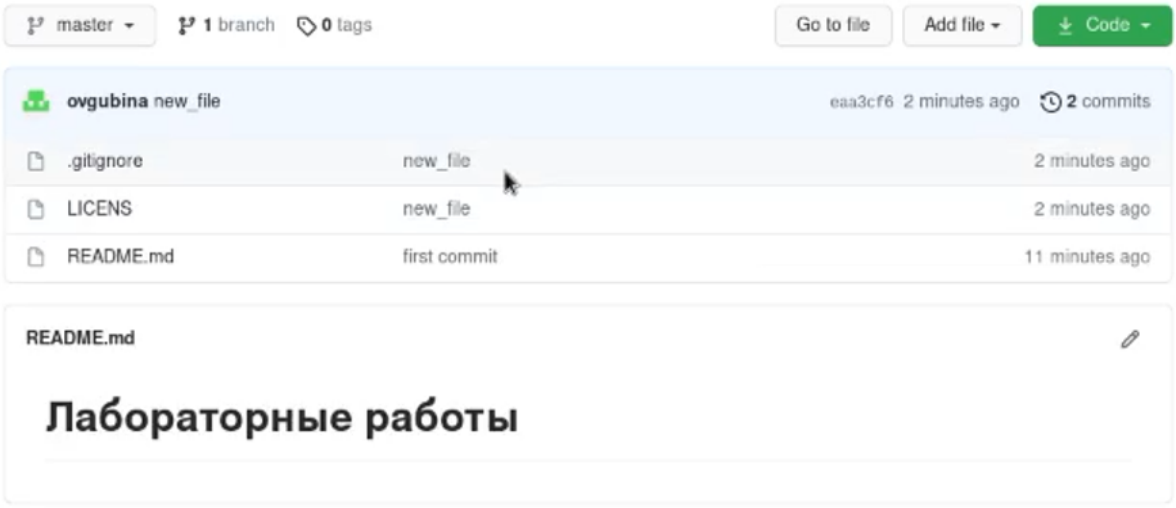

Рисунок 21

2.5.4. Конфигурация git-flow

Устанавливаем git-flow, ни одна из предложенных команд в материалах к лр№2 не подошла, поэтому используем команды со следующего сайта (Other Linuxes –> ~/bit):

https://github.com/nvie/gitflow/wiki/Linux

$ curl -OL https://raw.github.com/nvie/gitflow/develop/contrib/gitflow-installer.sh

$ chmod +x gitflow-installer.sh

$ sudo ./gitflow-installer.sh

$ INSTALL_PREFIX=~/bin ./gitflow-installer.sh

```
[ovgubina@localhost laboratory]$ curl -OL https://raw.github.com/nvie/gitflow/develop/contrib/gitflow-installer.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
100  2145  100  2145    0     0   1050      0  0:00:02  0:00:02 --:--:-- 1047k
[ovgubina@localhost laboratory]$ chmod +x gitflow-installer.sh
[ovgubina@localhost laboratory]$ sudo ./gitflow-installer.sh
[sudo] пароль для ovgubina:
ovgubina is not in the sudoers file.  This incident will be reported.
[ovgubina@localhost laboratory]$ INSTALL_PREFIX=~/bin ./gitflow-installer.sh
### gitflow no-make installer ###
Installing git-flow to /home/ovgubina/bin
Cloning repo from GitHub to gitflow
Cloning into 'gitflow'...
remote: Enumerating objects: 1407, done.
remote: Total 1407 (delta 0), reused 0 (delta 0), pack-reused 1407
Receiving objects: 100% (1407/1407), 484.34 KiB | 579.00 KiB/s, done.
Resolving deltas: 100% (796/796), done.
Updating submodules
Submodule 'shFlags' (git://github.com/nvie/shFlags.git) registered for path 'shFlags'
Cloning into 'shFlags'...
remote: Enumerating objects: 454, done.
remote: Total 454 (delta 0), reused 0 (delta 0), pack-reused 454
Receiving objects: 100% (454/454), 132.89 KiB | 0 bytes/s, done.
Resolving deltas: 100% (333/333), done.
Submodule path 'shFlags': checked out '2fb06af13de884e9680f14a00c82e52a67c867f1'
install: создание каталога «/home/ovgubina/bin»
«gitflow/git-flow» -> «/home/ovgubina/bin/git-flow»
«gitflow/git-flow-init» -> «/home/ovgubina/bin/git-flow-init»
«gitflow/git-flow-feature» -> «/home/ovgubina/bin/git-flow-feature»
«gitflow/git-flow-hotfix» -> «/home/ovgubina/bin/git-flow-hotfix»
«gitflow/git-flow-release» -> «/home/ovgubina/bin/git-flow-release»
«gitflow/git-flow-support» -> «/home/ovgubina/bin/git-flow-support»
«gitflow/git-flow-version» -> «/home/ovgubina/bin/git-flow-version»
«gitflow/gitflow-common» -> «/home/ovgubina/bin/gitflow-common»
«gitflow/gitflow-shFlags» -> «/home/ovgubina/bin/gitflow-shFlags»
[ovgubina@localhost laboratory]$
```

Рисунок 22

Инициализируем git-flow, притом префикс для ярлыков установим в v:

git flow init.

```
[ovgubina@localhost laboratory]$ git flow init

Which branch should be used for bringing forth production releases?
   - master
Branch name for production releases: [master]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? [] v
[ovgubina@localhost laboratory]$ gi
```

Рисунок 23

Проверяем, что мы находимся на ветке develop:

git branch.

Видим, что все верно:

```
[ovgubina@localhost laboratory]$ git branch
* develop
  master
```

Рисунок 24

Создадим релиз с версией 1.0.0:

git flow release start 1.0.0

```
[ovgubina@localhost laboratory]$ git flow release start 1.0.0
Switched to a new branch 'release/1.0.0'

Summary of actions:
- A new branch 'release/1.0.0' was created, based on 'develop'
- You are now on branch 'release/1.0.0'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your release
- When done, run:

     git flow release finish '1.0.0'

[ovgubina@localhost laboratory]$
```

Рисунок 25

Запишем версию:

echo "1.0.0" >> VERSION

После добавим в индекс:

git add .

git commit -am 'chore(main): add version'

```
[ovgubina@localhost laboratory]$ echo "1.0.0" >> VERSION
[ovgubina@localhost laboratory]$ git add .
[ovgubina@localhost laboratory]$ git commit -am 'chore(main): add version'
[release/1.0.0 12883f8] chore(main): add version
 3 files changed, 80 insertions(+)
 create mode 100644 VERSION
 create mode 160000 gitflow
 create mode 100755 gitflow-installer.sh
```

Рисунок 26

Зальём релизную ветку в основную ветку:

git flow release finish 1.0.0

```
[ovgubina@localhost laboratory]$ git flow release finish 1.0.0
warning: unable to rmdir gitflow: ??????? ?? ????
Switched to branch 'master'
Merge made by the 'recursive' strategy.
 VERSION              |  1 +
 gitflow              |  1 +
 gitflow-installer.sh | 78 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 80 insertions(+)
 create mode 100644 VERSION
 create mode 160000 gitflow
 create mode 100755 gitflow-installer.sh
warning: unable to rmdir gitflow: ??????? ?? ????
Switched to branch 'develop'
Merge made by the 'recursive' strategy.
 VERSION              |  1 +
 gitflow              |  1 +
 gitflow-installer.sh | 78 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 80 insertions(+)
 create mode 100644 VERSION
 create mode 160000 gitflow
 create mode 100755 gitflow-installer.sh
Deleted branch release/1.0.0 (was 12883f8).

Summary of actions:
- Latest objects have been fetched from 'origin'
- Release branch has been merged into 'master'
- The release was tagged 'v1.0.0'
- Release branch has been back-merged into 'develop'
- Release branch 'release/1.0.0' has been deleted

[ovgubina@localhost laboratory]$ git push --all
Counting objects: 7, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 1.38 KiB | 0 bytes/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To git@github.com:ovgubina/os-intro.git
   eaa3cf6..69d307c  master -> master
 * [new branch]      develop -> develop
[ovgubina@localhost laboratory]$
```

Рисунок 27

Отправим данные на github:

git push --all

git push --tags

```
[ovgubina@localhost laboratory]$ git push --all
Counting objects: 7, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 1.38 KiB | 0 bytes/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To git@github.com:ovgubina/os-intro.git
   eaa3cf6..69d307c  master -> master
 * [new branch]      develop -> develop
[ovgubina@localhost laboratory]$ git push --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 164 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:ovgubina/os-intro.git
 * [new tag]         v1.0.0 -> v1.0.0
[ovgubina@localhost laboratory]$
```

Рисунок 28

Проверим, что файлы действительно были переданы на github и посмотрим файлы в каталоге (рисунки 29-30):

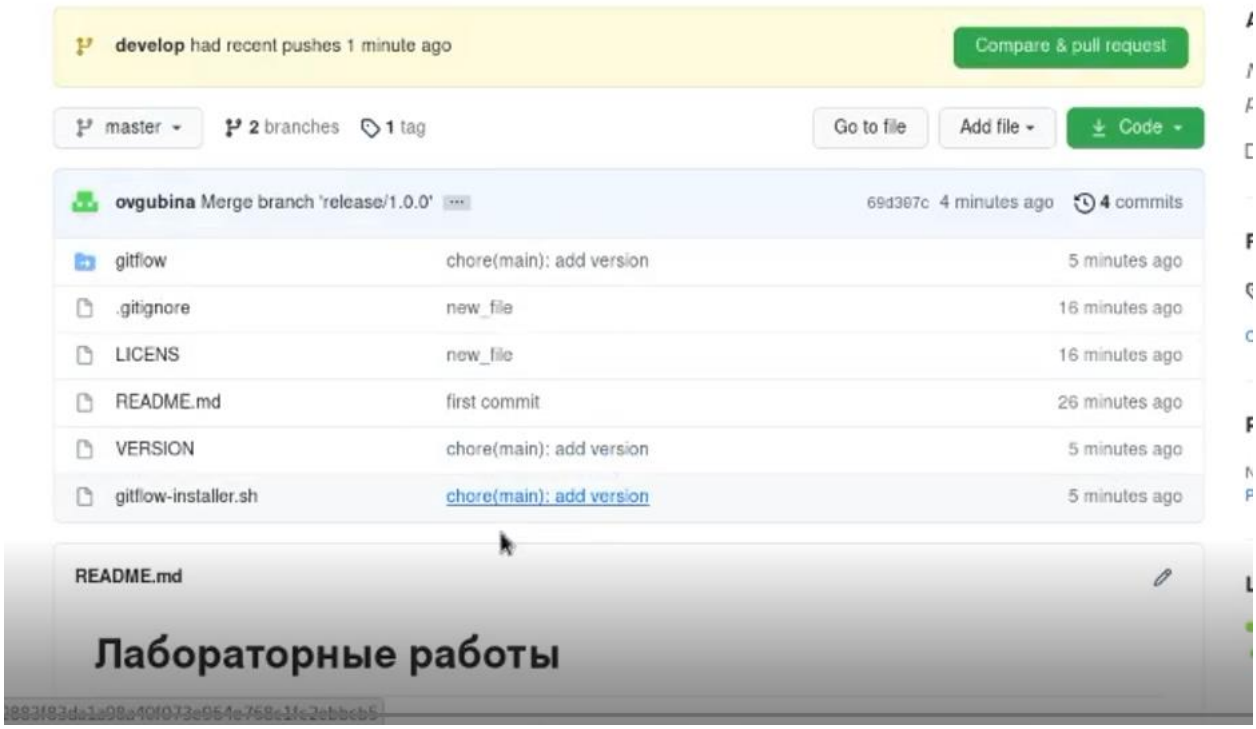

Рисунок 29

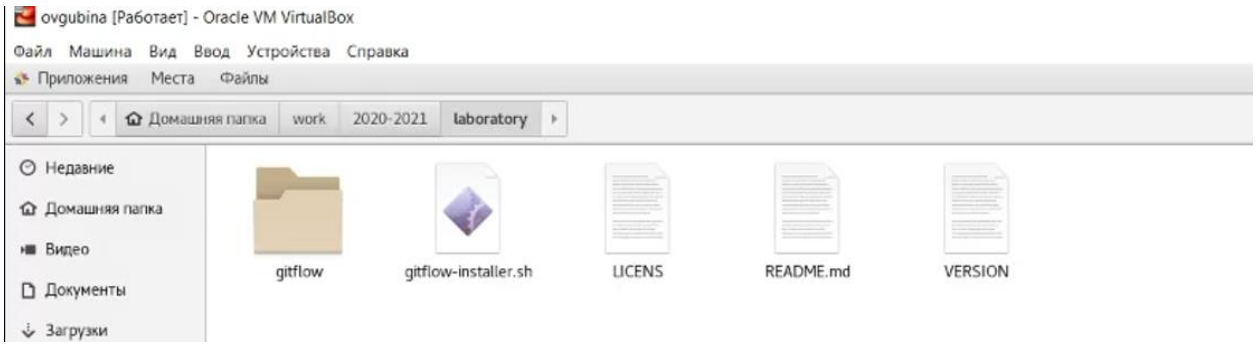

Рисунок 30

2.7. Контрольные вопросы

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

Системы контроля версий (VCS) - программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Предназначены для работы нескольких человек над одним проектом, а также при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы.

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

Хранилище – место «памяти», в котором будет храниться новая версия файла после его изменения пользователем.

Commit – это основной объект в любой системе управления версиями. В нем содержится описание тех изменений, которые вносит пользователь в код приложения.

История –история изменений. 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

Рабочая копия – это копия, которую мы выписали в свою рабочую зону, это то, над чем мы работаем в данный момент. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом.

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

Централизованные VCS предполагают наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером.

Децентрализованные VCS не имеют единого репозитория, он у каждого пользователя свой. Помимо того, они были созданы для обмена изменениями, а не для их объединения. Не имеют какой-то жестко заданной структуры репозиториев с центральным сервером.

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

При единоличной работе с vcs сохраняются не все предыдущие версии. Изменения сохраняются по системе: одно предыдущее + новая информация.

5. Опишите порядок работы с общим хранилищем VCS.

1. Создать репозиторий.

2. Скачать проект из репозитория.

3. Обновить проект, забрать последнюю версию из репозитория.

4. Внести изменения в проект.

5. Запушить код, т.е изменить код в общем хранилище.

6. Создать ветку.

7. Теперь, если нужно закоммитить изменения.

6. Каковы основные задачи, решаемые инструментальным средством git?

- Сохранение файлов с исходным кодом
- Защита от случайных исправлений и удалений
- Отмена изменений и удалений, если они оказались некорректными
- Одновременная поддержка рабочей версии и разработка новой
- Возврат к любой версии кода из прошлого
- Просмотр истории изменений
- Совместная работа без боязни потерять данные или затереть чужую работу

7. Назовите и дайте краткую характеристику командам git.

git init – создание основного дерева репозитория

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

git push – отправка всех произведённых изменений локального дерева в центральный репозиторий

git status – просмотр списка изменённых файлов в текущей директории

git diff – просмотр текущих изменения

git add – добавить все изменённые и/или созданные файлы и/или каталоги

git add имена_файлов – добавить конкретные изменённые и/или созданные файлы и/или каталоги

git rm имена_файлов – удалить файл и/или каталог из индекса репозитория

git commit -am 'Описание коммита' – сохранить все добавленные изменения и все изменённые файлы

git commit – сохранить добавленные изменения с внесением комментария через встроенный редактор

git checkout -b имя_ветки – создание новой ветки, базирующейся на текущей:

git checkout имя_ветки – переключение на некоторую ветку

git push origin имя_ветки – отправка изменений конкретной ветки в центральный репозиторий

git merge --no-ff имя_ветки – слияние ветки с текущим деревом

git branch -d имя_ветки – удаление локальной уже слитой с основным деревом ветки

git branch -D имя_ветки – принудительное удаление локальной ветки

git push origin :имя_ветки – удаление ветки с центрального репозитория

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

Локальный репозиторий – она же директория ".git". В ней хранятся коммиты и другие объекты.

Удаленный репозиторий – тот самый репозиторий, который считается общим, в который мы можем передать свои коммиты из локального репозитория, чтобы остальные пользователи могли их увидеть. Удаленных репозиториев может быть несколько, но обычно он бывает один.

Локальный репозиторий мы используем, когда работаем одни и нам нужно сохранить свои же изменения.

Удаленный же репозиторий используется для групповой работы, когда в личном репозитории скопилось достаточно коммитов, мы делимся ими в удаленном для того, чтобы другие пользователи могли видеть наши изменения. Также из удаленного репозитория мы можем скачать чужие изменения.

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

Ветка – это подвижный указатель на один из коммитов. Обычно ветка указывает на последний коммит в цепочке коммитов. Ветка берет свое начало от какого-то одного коммита.

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

Файлы, которые можно игнорировать, включают в себя:

- Файлы с конфиденциальной информацией, такой как пароли или ключи API.
- Файлы времени выполнения, такие как журнал, блокировка, кэш или

временные файлы.

- Скомпилированный код, такой как .class или .o.
- Каталоги зависимостей, такие как /vendor или /node_modules.
- Создавать папки, такие как /public, /out или /dist.
- Системные файлы, такие как .DS_Store или Thumbs.db
- Конфигурационные файлы IDE или текстового редактора.

Личные правила игнорирования

Шаблоны, специфичные для вашего локального репозитория и не подлежащие распространению в другие репозитории, должны быть установлены в файле .git/info/exclude .
Например, вы можете использовать этот файл, чтобы игнорировать файлы, сгенерированные из ваших личных инструментов проекта.

Глобальный .gitignore

Git также позволяет вам создать глобальный файл .gitignore , в котором вы можете определить правила игнорирования для каждого репозитория Git в вашей локальной системе.
Файл можно назвать как угодно и хранить в любом месте. Чаще всего этот файл хранится в домашнем каталоге. Вам придется вручную создать файл и настроить Git для его использования.

Заключение:

Исходя из приведенных выше скриншотов, свидетельствующих о результатах выполнения команд, можно сказать, что работа выполнена успешно. Был освоен GitHub, который понадобится в дальнейшей работе.

Вывод:

Изучила идеологию и применение средств контроля версий. Освоила Github.

Библиографический список:

1. https://gist.github.com/rdnvndr/cb21a06c5a71fd71213aed1619380b8e