Отчёт лабораторной работы №3

Дисциплина: Операционные системы

Касьянов Даниил Владимирович

Содержание

Список таблиц

Список иллюстраций

Цель работы

Целью данной работы является изучение идеологии применения средств контроля версий.

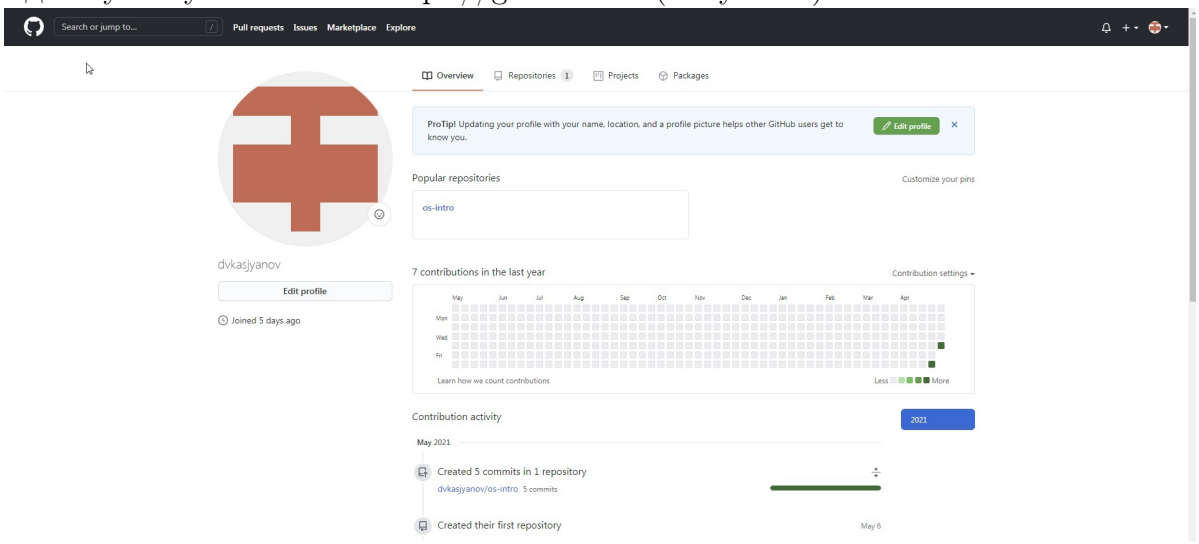
Задание

– 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.).

Выполнение лабораторной работы

Создаем учётную запись на https://github.com (Рисунок 1).



(Рисунок 1)

Настраиваем систему контроля версий git с использованием сервера репозиториев https://github.com/. Сделаем предварительную конфигурацию, указав имя и email владельца репозитория (dvkasjyanov, kasyanoff.daniil@yandex.ru) (Рисунок 2).

```
[dvkasjyanov@dvkasjyanov ~]$ git config --global user.name "dvkasjyanov"
[dvkasjyanov@dvkasjyanov ~]$ git config --global user.email "kasyanoff.daniil@ya
ndex.ru"
[dvkasjyanov@dvkasjyanov ~]$
```

(Рисунок 2)

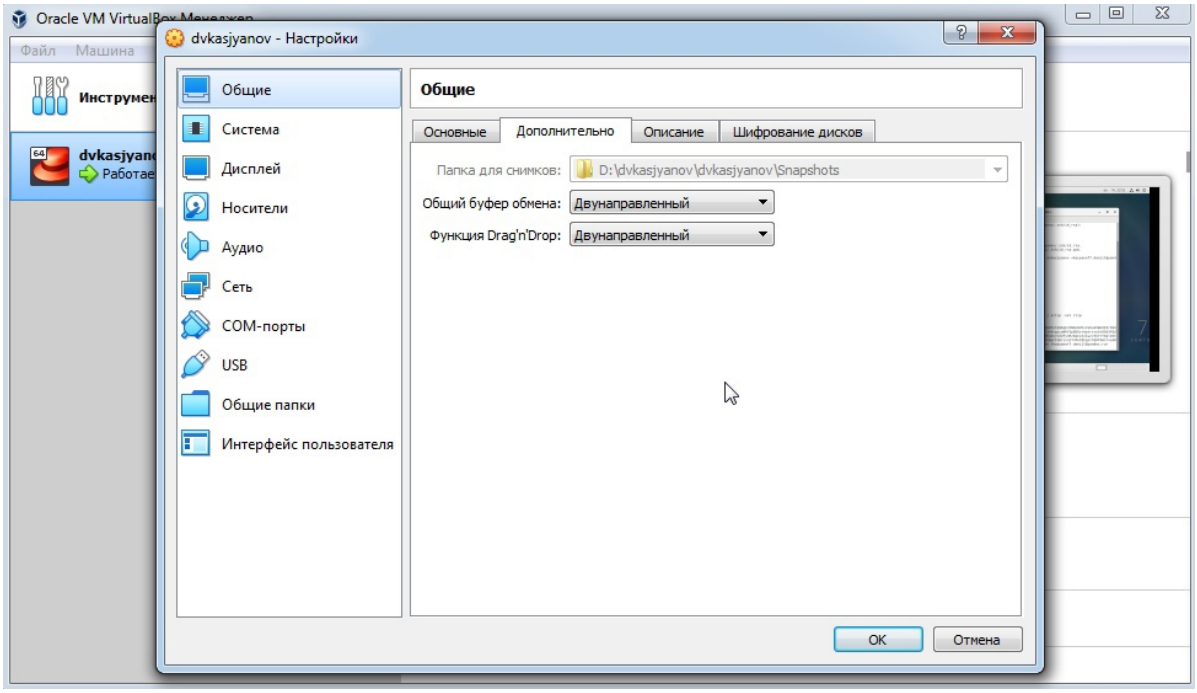
Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый) (Рисунок 3):

ssh-keygen -C "dvkasjyanov <kasyanoff.daniil@yandex.ru>"

```
[dvkasjyanov@dvkasjyanov ~]$ git config --global user.name "dvkasjyanov"
[dvkasjyanov@dvkasjyanov ~]$ git config --global user.email "kasyanoff.daniil@yandex.
ru"
[dvkasjyanov@dvkasjyanov ~]$ ssh-keygen -C "dvkasjyanov <kasyanoff.daniil@yandex.ru>"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/dvkasjyanov/.ssh/id_rsa):
/home/dvkasjyanov/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/dvkasjyanov/.ssh/id_rsa.
Your public key has been saved in /home/dvkasjyanov/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:mRPbpouBMcDOWB8RQuvLrC49XHRaaD3qsRCkhWKZIVw dvkasjyanov <kasyanoff.daniil@yand
ex.ru>
The key's randomart image is:
+---[RSA 2048]----+
|o+BEo.           |
|+*oo .           |
|o+= .o  .        |
|.*.o+.=  *       |
|. +++= .S o      |
| o..=+   +       |
| o++.o. .        |
|..+ o  o .       |
|+. .  . .        |
+----[SHA256]-----+
[dvkasjyanov@dvkasjyanov ~]$
```

(Рисунок 3)

Перехожу в «Настройки» → «Общие» → «Дополнительно» в виртуальной машине. В параметрах «Общий буфер обмена» и «Функция Drag'n'Drop» выбираю «Двунаправленный» (Рисунок 4). Это необходимо для того, чтобы ключ, сгенерированный в терминале Linux, можно было вставить в браузере в Windows.



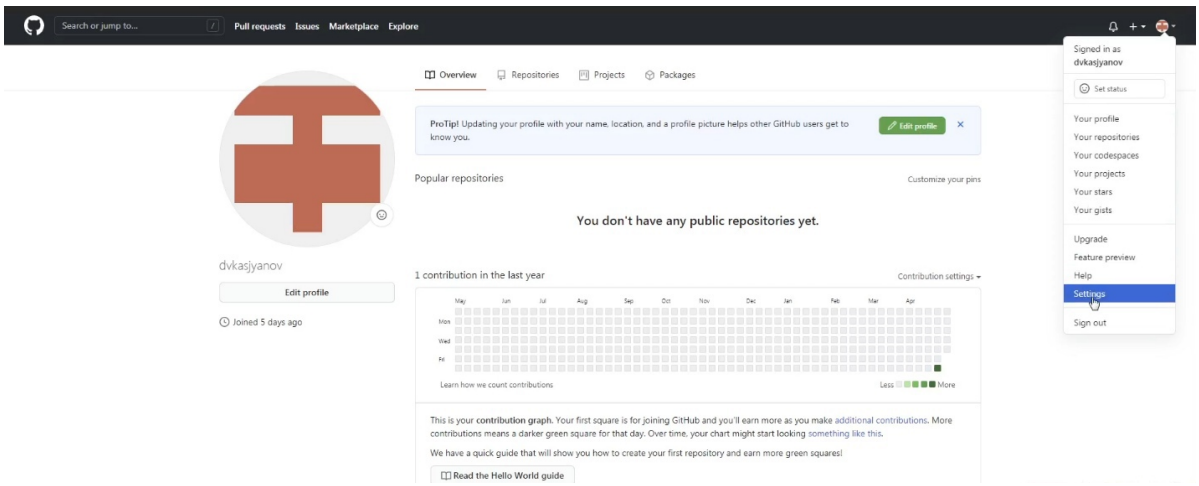
(Рисунок 4)

Используем команду cat ~/.ssh/id_rsa.pub | xclip-sel clip для копирования ключа в буфер обмена. В моем случае команда выдает ошибку, поэтому я использую команду cat ~/.ssh/id_rsa.pub и копирую сгенерированный ключ самостоятельно (Рисунок 5).

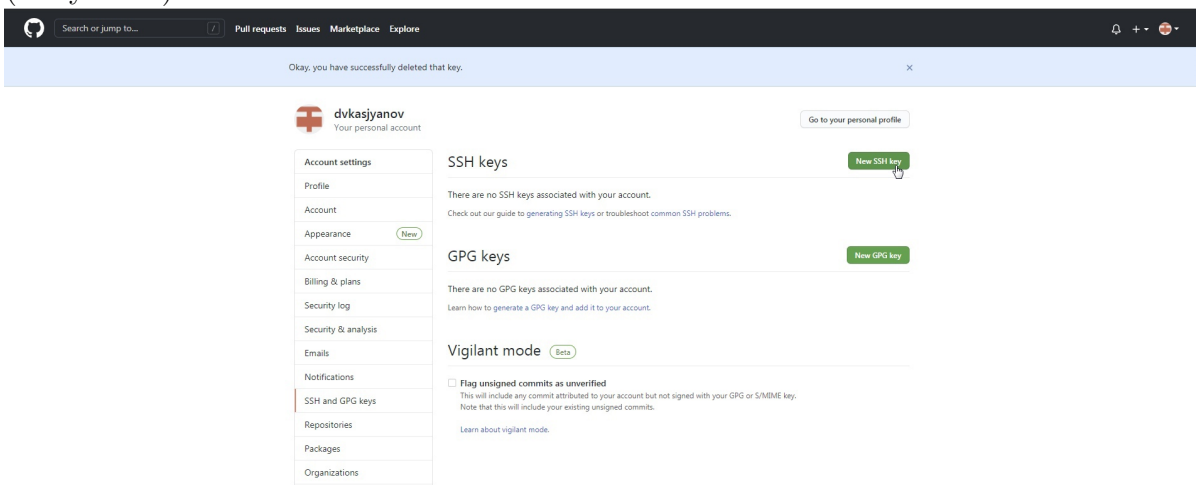
```
[dvkasjyanov@dvkasjyanov ~]$ cat ~/.ssh/id_rsa.pub | xclip -sel clip
bash: xclip: команда не найдена...
[dvkasjyanov@dvkasjyanov ~]$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDGpWjaLnUPTj4OXPstGQKqErKNbn0VhvYeOu4VWo9IOrOSQ
qVmA3j1+JbuaBIUhs33yVl2KfaOfxUZQFMBsmHFdhpeP41N/U471rHrgica0KY5pDEUz+Aptrs+yVz9Sd3fQd
ZN8QBk5HmgyyqvACRgZzrsAxnedPyz6CakaecFtG9pF5DUhohSZEiDv+CXZFynRzbqnzZJiwz/5kY+T4q+aVS
8IZ9pq7O6VJTMJoDVq4neHbPZBb/6VRTHHB1c1vxlhKRfD4IdvJCmpcTxDrisn2TrRnYQFgsltQDfdHJrxL6R
f5WBG9MX00UbmYbGBXlm/ZEpXtnIgv6AsF97tivJ dvkasjyanov <kasyanoff.daniil@yandex.ru>
```

(Рисунок 5)

Загрузим ключ на сайте https://github.com/. Переходим в меню «GitHub setting», выбираем в боковом меню «GitHub setting» (Рисунок 6), «SSH-ключи» и нажимаем кнопку «Добавить ключ» (Рисунок 7).

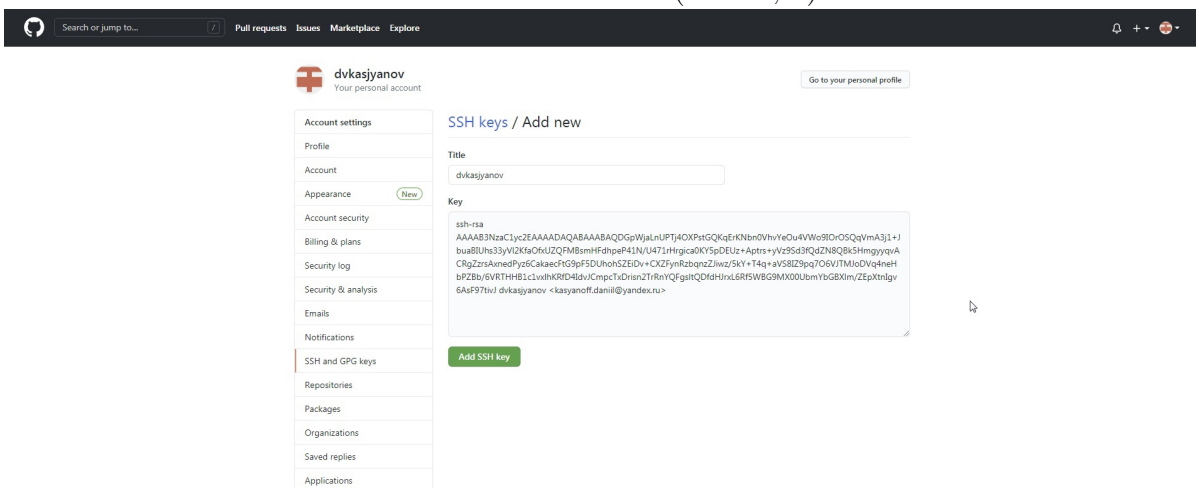


(Рисунок 6)

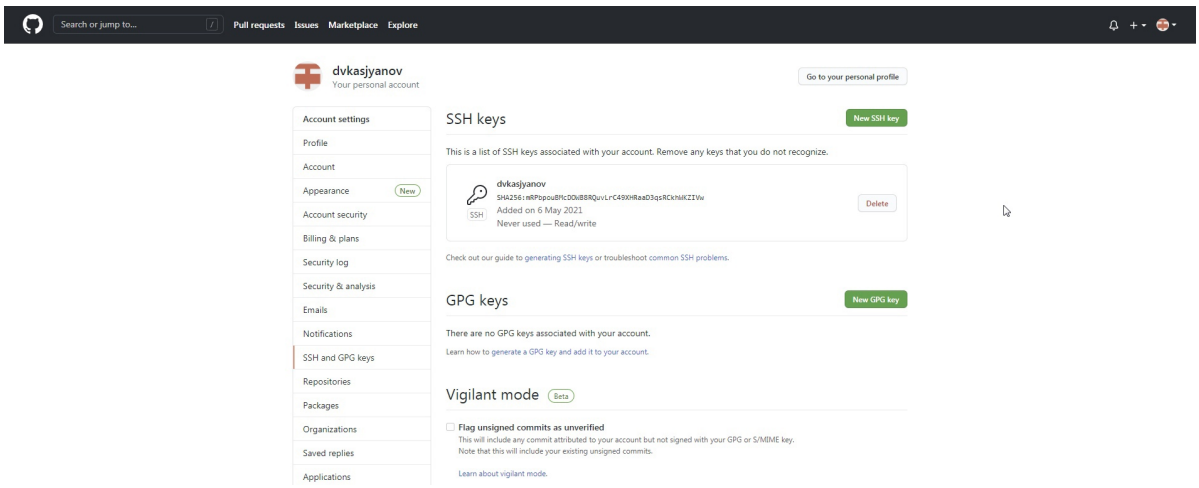


(Рисунок 7)

Вставляем ключ в появившееся на сайте поле (Рис. 8, 9).

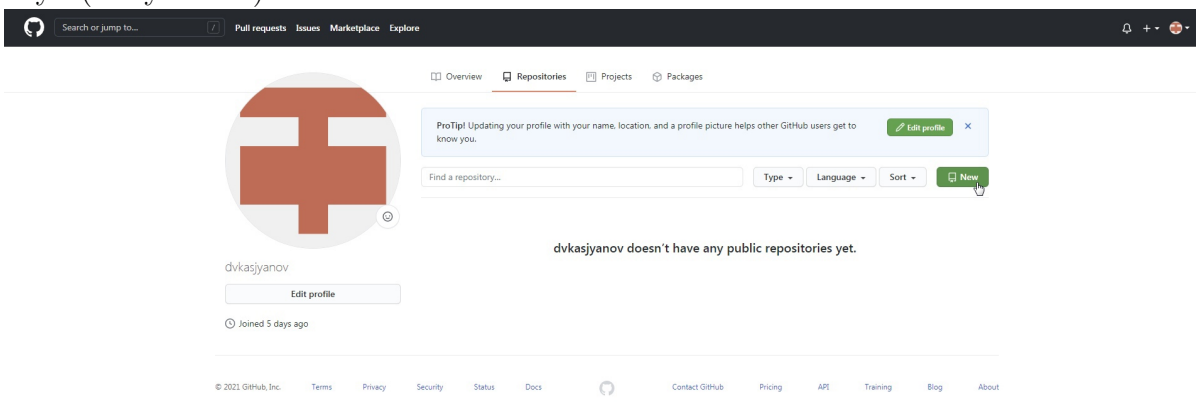


(Рисунок 8)

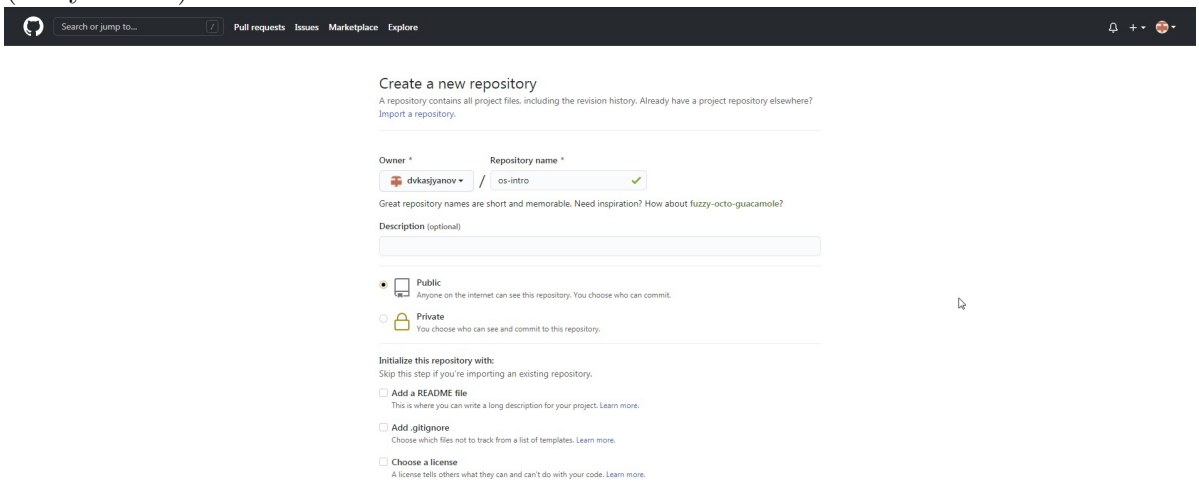


(Рисунок 9)

Создаём репозиторий на https://github.com/, выбрав в меню «Репозитории» → «Создать репозиторий» (Рисунок 10). Назову репозиторий os-intro и открою общий доступ (Рисунок 11).



(Рисунок 10)



(Рисунок 11)

Создадим рабочий каталог laboratory, указав его расположение согласно соглашению об именовании (Рисунок 12).

```
[dvkasjyanov@dvkasjyanov ~]$ cd work
[dvkasjyanov@dvkasjyanov work]$ cd 2020-2021
[dvkasjyanov@dvkasjyanov 2020-2021]$ cd OS
[dvkasjyanov@dvkasjyanov OS]$ cd laboratory
[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 12)

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 (Рисунок 13):

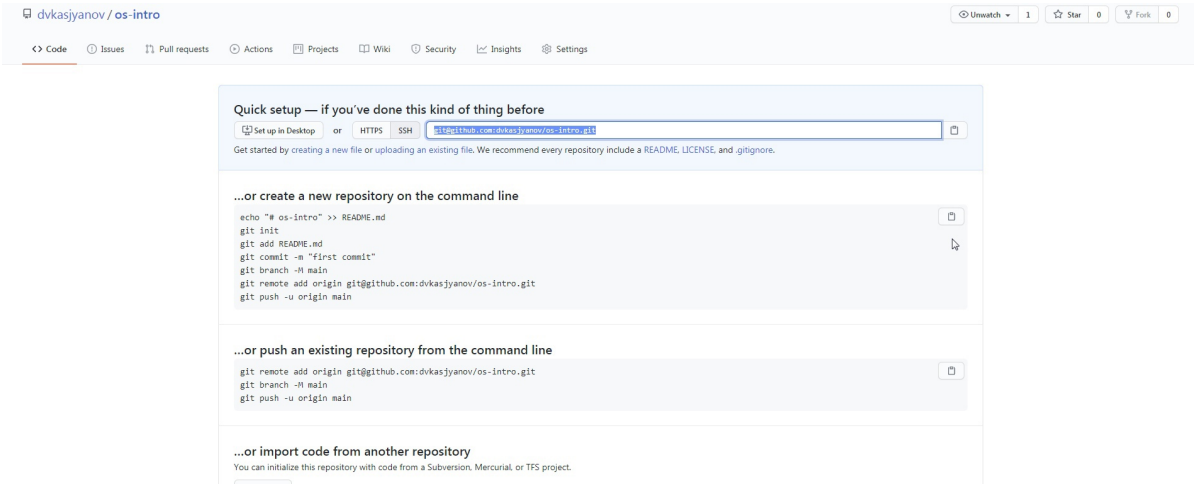
git commit -m "first commit"

```
[dvkasjyanov@dvkasjyanov laboratory]$ git init
Initialized empty Git repository in /home/dvkasjyanov/work/2020-2021/OS/laboratory/.git/
[dvkasjyanov@dvkasjyanov laboratory]$ echo "# Лабораторные работы" >> README.md
[dvkasjyanov@dvkasjyanov laboratory]$ git add README.md
[dvkasjyanov@dvkasjyanov laboratory]$ git commit -m "first commit"
[master (root-commit) 740cfe0] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 13)

Копируем SSH ссылку на репозиторий с сайта github (Рисунок 14), выкладываем созданный репозиторий на github, используя ссылку в формате SSH (Рисунок 15):

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



(Рисунок 14)

```
[dvkasjyanov@dvkasjyanov laboratory]$ git remote add origin git@github.com:dvkasjyanov/os-intro.git
[dvkasjyanov@dvkasjyanov laboratory]$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 250 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:dvkasjyanov/os-intro.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 15)

Проведём первичную конфигурацию. Добавим файл лицензии (Рисунок 16):

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

```
[dvkasjyanov@dvkasjyanov laboratory]$ wget https://creativecommons.org/licenses/by/4.0/legalcode.txt -O
 LICENSE
--2021-05-06 22:59:26--  https://creativecommons.org/licenses/by/4.0/legalcode.txt
Распознаётся creativecommons.org (creativecommons.org)... 104.20.150.16, 172.67.34.140, 104.20.151.16,
...
Подключение к creativecommons.org (creativecommons.org)|104.20.150.16|:443... соединение установлено.
HTTP-запрос отправлен. Ожидание ответа... 200 OK
Длина: нет данных [text/plain]
Сохранение в: «LICENSE»

    [ <=>                                                   ] 18 657      --.-K/s   за 0,02s

2021-05-06 22:59:28 (1,04 MB/s) - «LICENSE» сохранён [18657]

[dvkasjyanov@dvkasjyanov laboratory]$
```

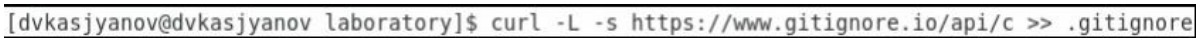
(Рисунок 16)

Добавим шаблон игнорируемых файлов. Просмотреть список имеющихся шаблонов можно с помощью команды:

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

Скачаем шаблон для C (Рисунок 17):

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



(Рисунок 17)

Добавим новые файлы:

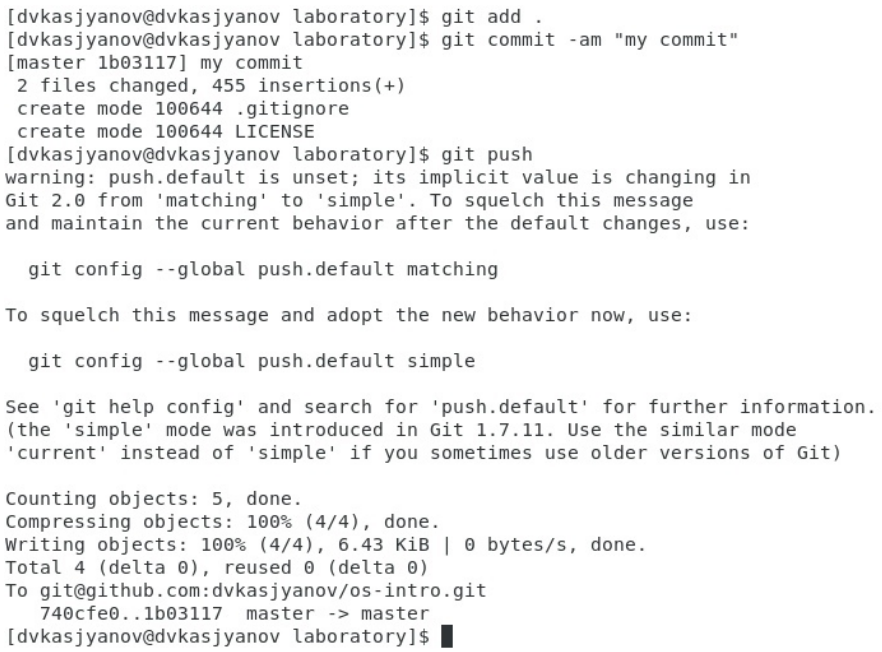
git add .

Выполним коммит:

git commit –am 'my commit'

Отправим на github (Рисунок 18):

git push



(Рисунок 18)

Инициализируем git-flow:

git flow init

Оставим префиксы master, develop, feature/, release/, hotfix/, support/ без изменений; префикс для ярлыков установим в v (Рисунок 19).

```
[dvkasjyanov@dvkasjyanov laboratory]$ git flow init

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

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

(Рисунок 19)

Проверим, что мы на ветке develop: git branch (Рисунок 20).

```
[dvkasjyanov@dvkasjyanov laboratory]$ git branch
* develop
  master
[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 20)

Создадим релиз с версией 1.0.0: git flow release start 1.0.0 (Рисунок 21).

```
[dvkasjyanov@dvkasjyanov 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'

[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 21)

Запишем версию: echo "1.0.0" >> VERSION (Рисунок 22).

```
[dvkasjyanov@dvkasjyanov laboratory]$ echo "1.0.0" >> VERSION
```

(Рисунок 22)

Добавим в индекс:

git add .

Сделаем commit (Рисунок 23):

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

```
[dvkasjyanov@dvkasjyanov laboratory]$ git add .
[dvkasjyanov@dvkasjyanov laboratory]$ git commit -am "chore(main): add version"
[release/1.0.0 69b8853] chore(main): add version
 1 file changed, 1 insertion(+)
 create mode 100644 VERSION
[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 23)

Зальём релизную ветку в основную ветку (Рисунок 24):

git flow release finish 1.0.0

```
[dvkasjyanov@dvkasjyanov laboratory]$ git flow release finish 1.0.0
Branches 'master' and 'origin/master' have diverged.
And local branch 'master' is ahead of 'origin/master'.
Switched to branch 'develop'
Merge made by the 'recursive' strategy.
 VERSION | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 VERSION
Deleted branch release/1.0.0 (was 69b8853).

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

[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 24)

В открывшихся окнах вводим произвольные сообщения (Рис. 25, 26).

```
Finish
#
# Write a tag message
# Lines starting with '#' will be ignored.
~
```

(Рисунок 25)

```
Merge branch 'release/1.0.0' into develop

it's necessary

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
```

(Рисунок 26)

Отправим данные на github (Рисунок 27):

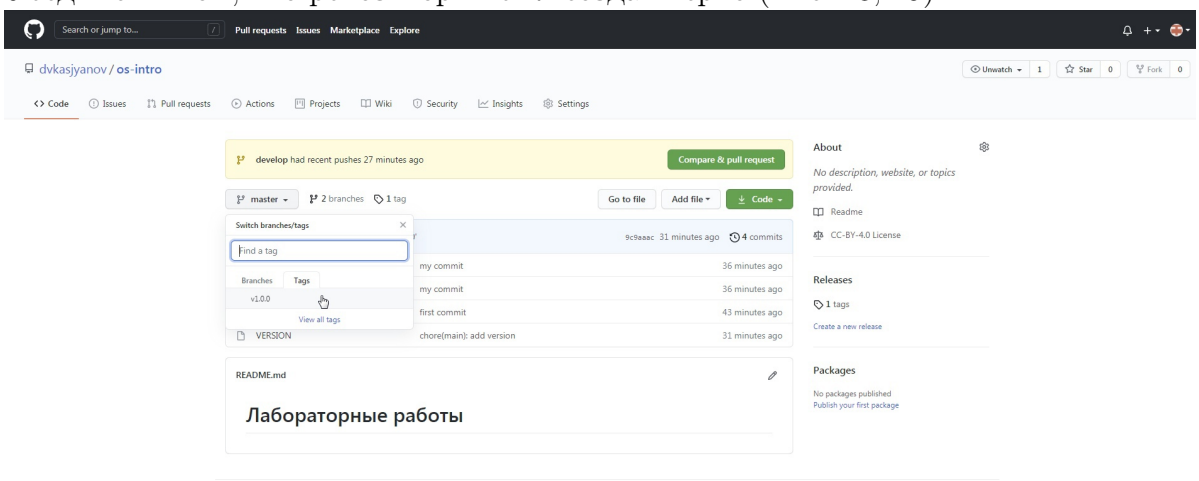
git push --all

git push –tags

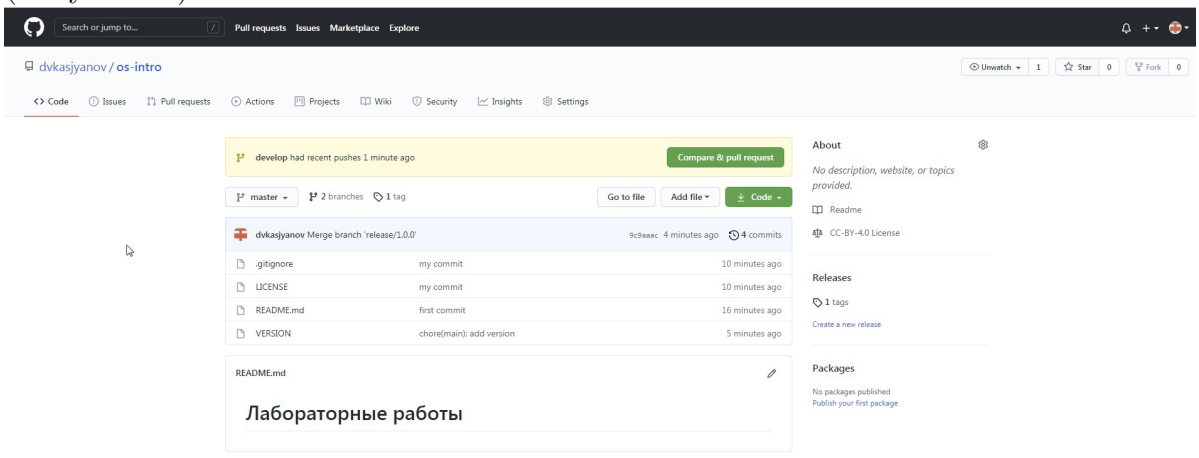
```
[dvkasjyanov@dvkasjyanov laboratory]$ git push --all
Counting objects: 6, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 436 bytes | 0 bytes/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
To git@github.com:dvkasjyanov/os-intro.git
   1b03117..9c9aaac  master -> master
 * [new branch]      develop -> develop
[dvkasjyanov@dvkasjyanov 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:dvkasjyanov/os-intro.git
 * [new tag]         v1.0.0 -> v1.0.0
[dvkasjyanov@dvkasjyanov laboratory]$
```

(Рисунок 27)

Убедимся в том, что репозиторий был создан верно (Рис. 28, 29).



(Рисунок 28)



(Рисунок 29)

Выводы

Я изучил идеологию и применение средств контроля версий.