

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение  
высшего образования**

**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Институт цифрового развития**

**Кафедра информационных систем и технологий**

Отчет по лабораторной работе №2.

Дисциплина: «Основы программной инженерии»

**Выполнил:**

Студент группы ПИЖ-б-о-22-1,  
направление подготовки: 09.03.04  
«Программная инженерия»

ФИО: Франк Дмитрий Денисович

**Проверил:**

Богданов С.С.

Доцент кафедры инфокоммуникаций

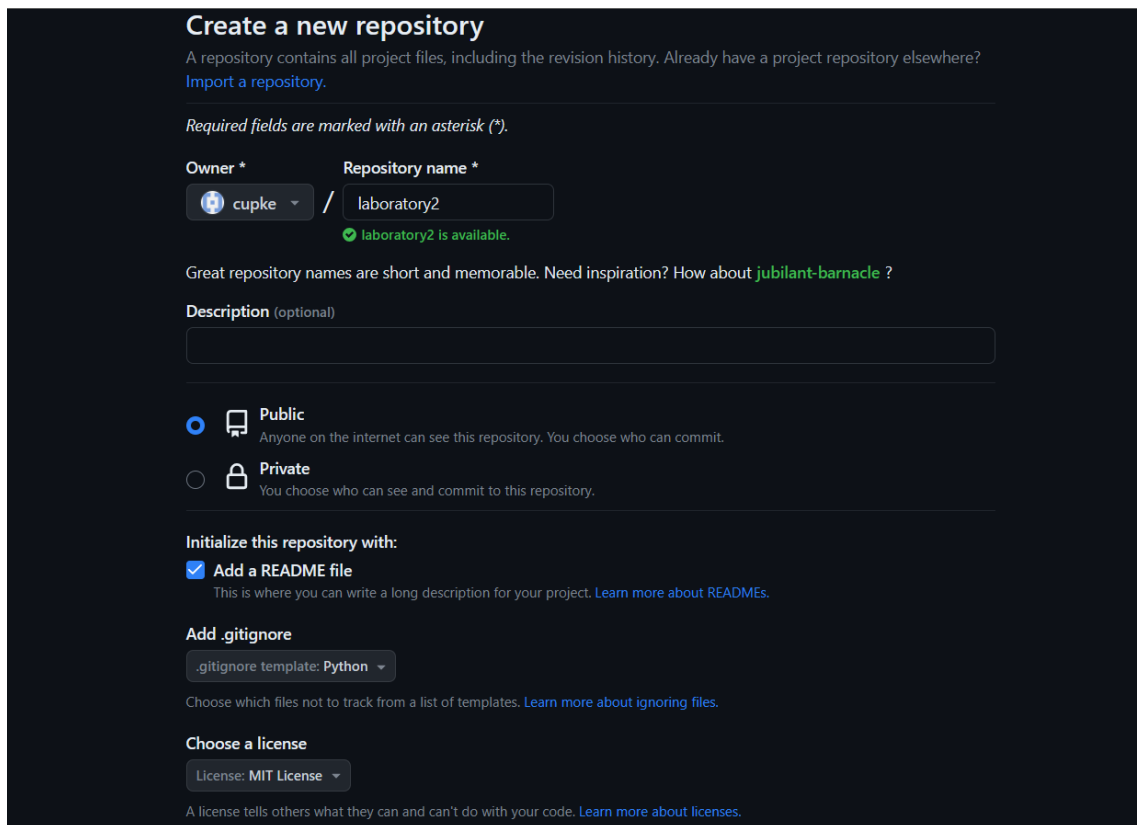
Ставрополь 2022

Тема: Исследование возможностей Git для работы с локальными репозиториями.

Цель работы: исследовать базовые возможности системы контроля версий Git для работы с локальными репозиториями.

Выполнение работы:

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и выбранный язык программирования (python).




**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*


**Owner \*** **Repository name \***


 cupke / laboratory2

✔ laboratory2 is available.

Great repository names are short and memorable. Need inspiration? How about [jubilant-barnacle](#) ?

**Description** (optional)


☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**


☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

**Add .gitignore**

 **.gitignore template: Python**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

**Choose a license**

 **License: MIT License**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

Рисунок 2.1 — Создание репозитория

3. Выполнил клонирование созданного репозитория на рабочий компьютер.

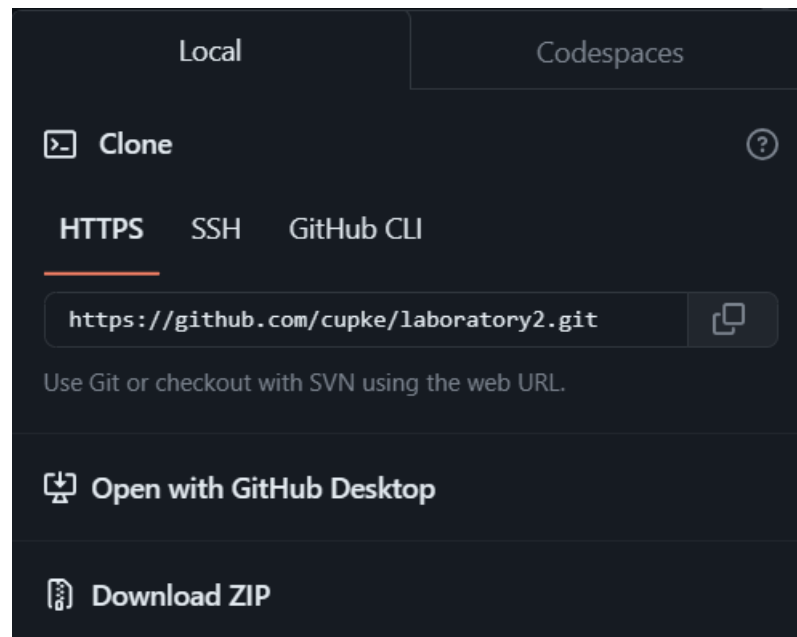


Рисунок 2.2 — Ссылка удаленного репозитория

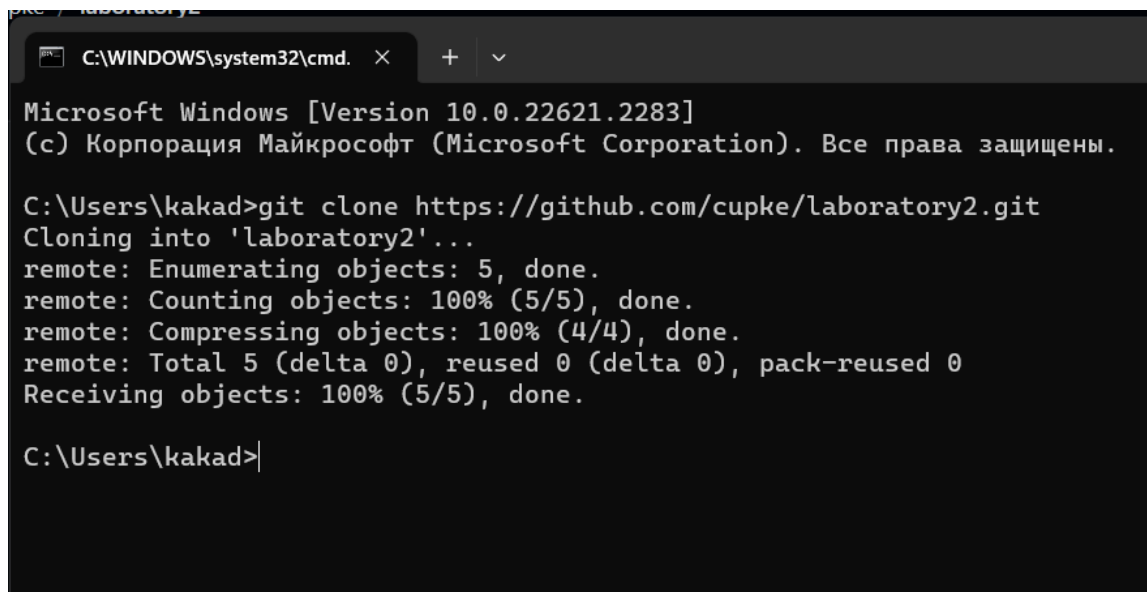
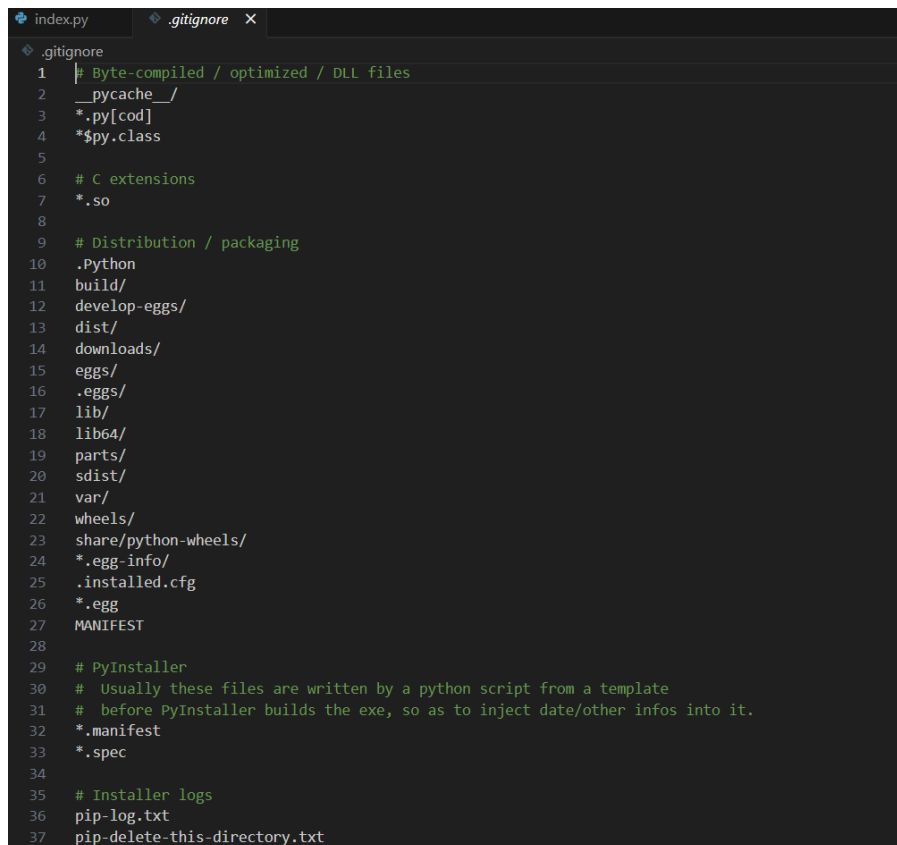


Рисунок 2.3 — Клонирования репозитория

4. Дополнил файл. gitignore необходимыми правилами для выбранного языка программирования и интегрированной среды разработки.



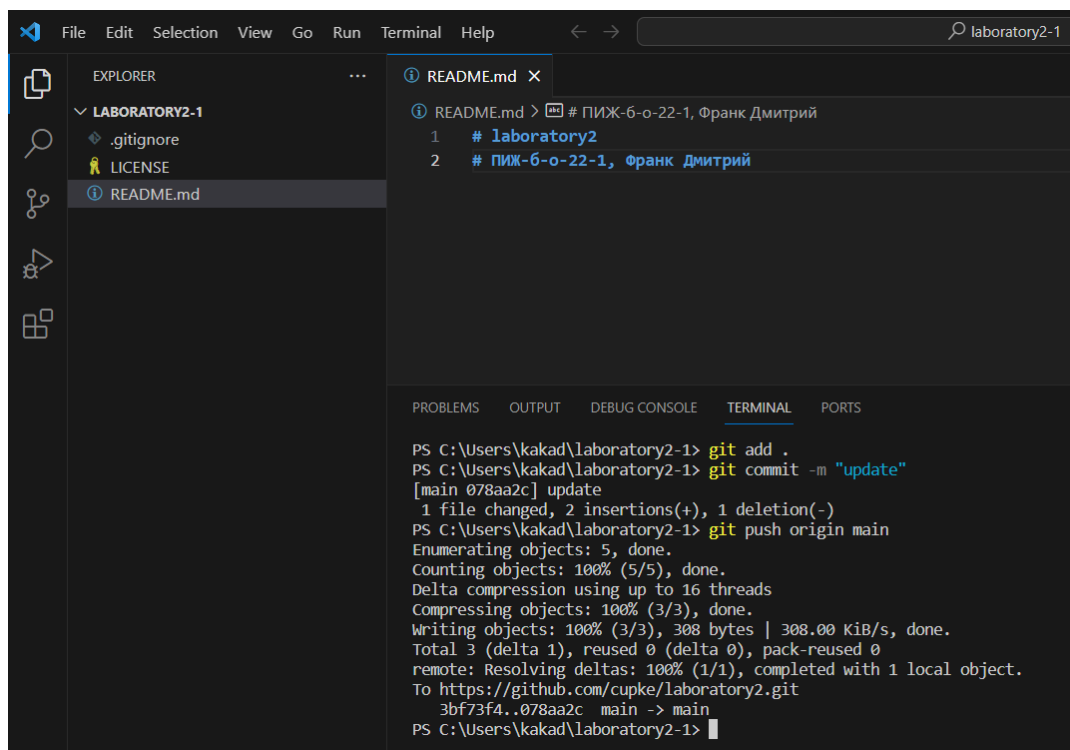
```

1  # Byte-compiled / optimized / DLL files
2  __pycache__ /
3  *.py[.cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
29 # PyInstaller
30 # Usually these files are written by a python script from a template
31 # before PyInstaller builds the exe, so as to inject date/other infos into it.
32 *.manifest
33 *.spec
34
35 # Installer logs
36 pip-log.txt
37 pip-delete-this-directory.txt

```

Рисунок 2.4 — Файл. gitignore

5. Добавил в файл README.md информацию о дисциплине, группе и ФИО студента, выполняющего лабораторную работу.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows the file structure of a project named 'LABORATORY2-1', with files like '.gitignore', 'LICENSE', and 'README.md'. The 'README.md' file is selected and its content is shown in the main editor area. The content of 'README.md' is as follows:

```

1  # laboratory2
2  # ПИЖ-6-о-22-1, Франк Дмитрий

```

At the bottom, the Terminal pane shows the output of the following commands:

```

PS C:\Users\kakad\laboratory2-1> git add .
PS C:\Users\kakad\laboratory2-1> git commit -m "update"
[main 078aa2c] update
1 file changed, 2 insertions(+), 1 deletion(-)
PS C:\Users\kakad\laboratory2-1> git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/cupke/laboratory2.git
3bf73f4..078aa2c main -> main
PS C:\Users\kakad\laboratory2-1>

```

Рисунок 2.5 — внесение изменений в файл README.md

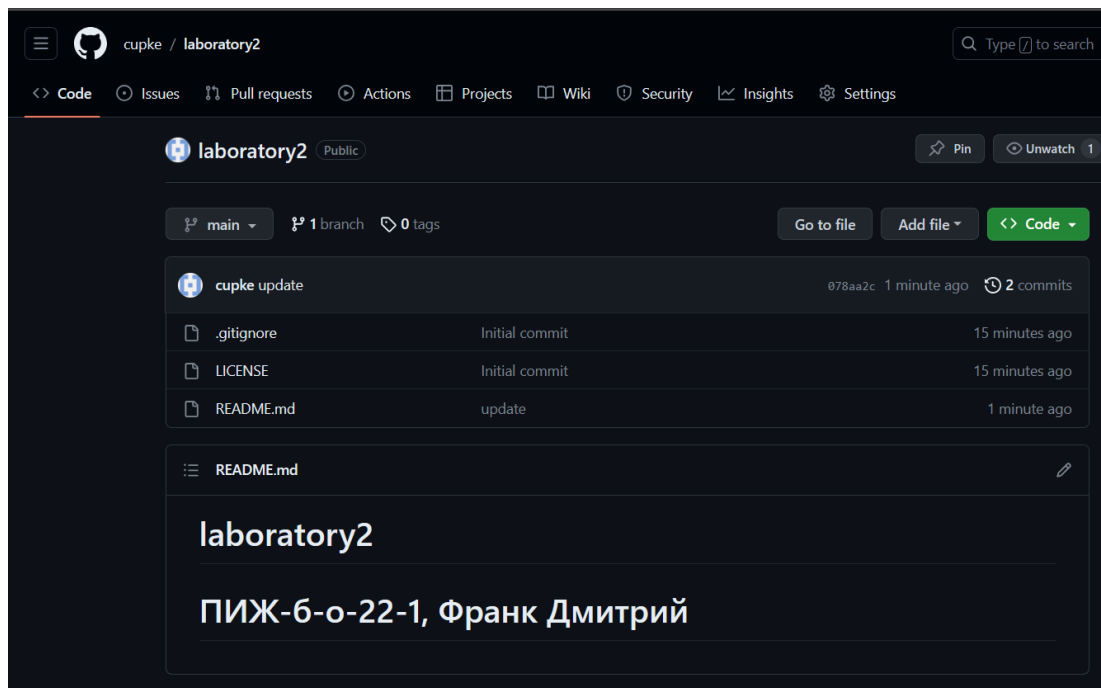


Рисунок 2.6 — Результат внесенных изменений в файл README.md

6. Написал небольшую программу на выбранном языке программирования. Сделал 3 тега.

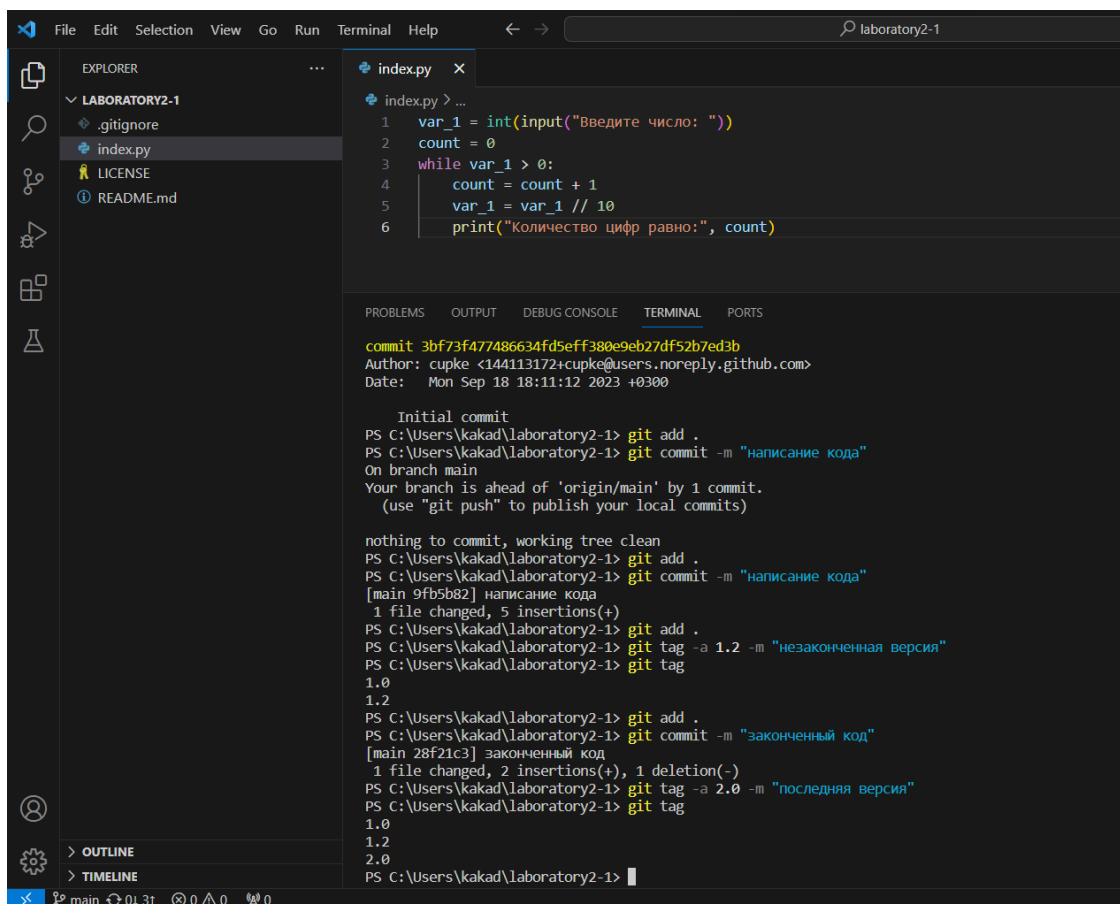
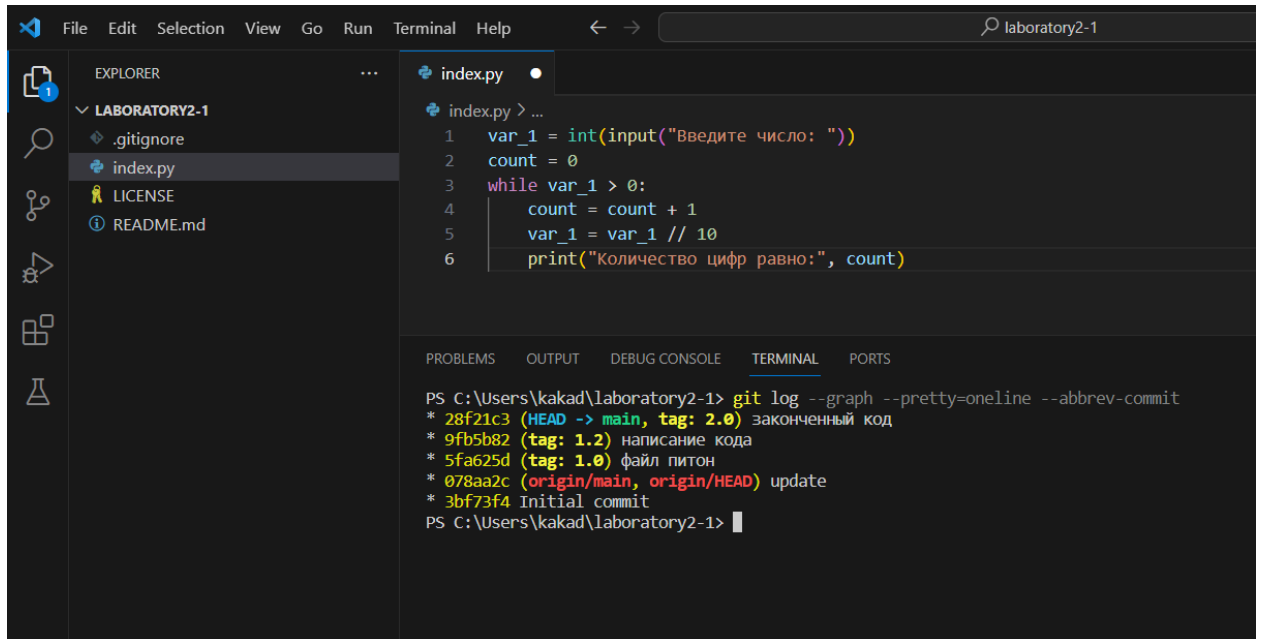


Рисунок 2.7 — Программа на python

7. Посмотрел историю хранилища командой «git log --graph --pretty=oneline --abbrev-commit».

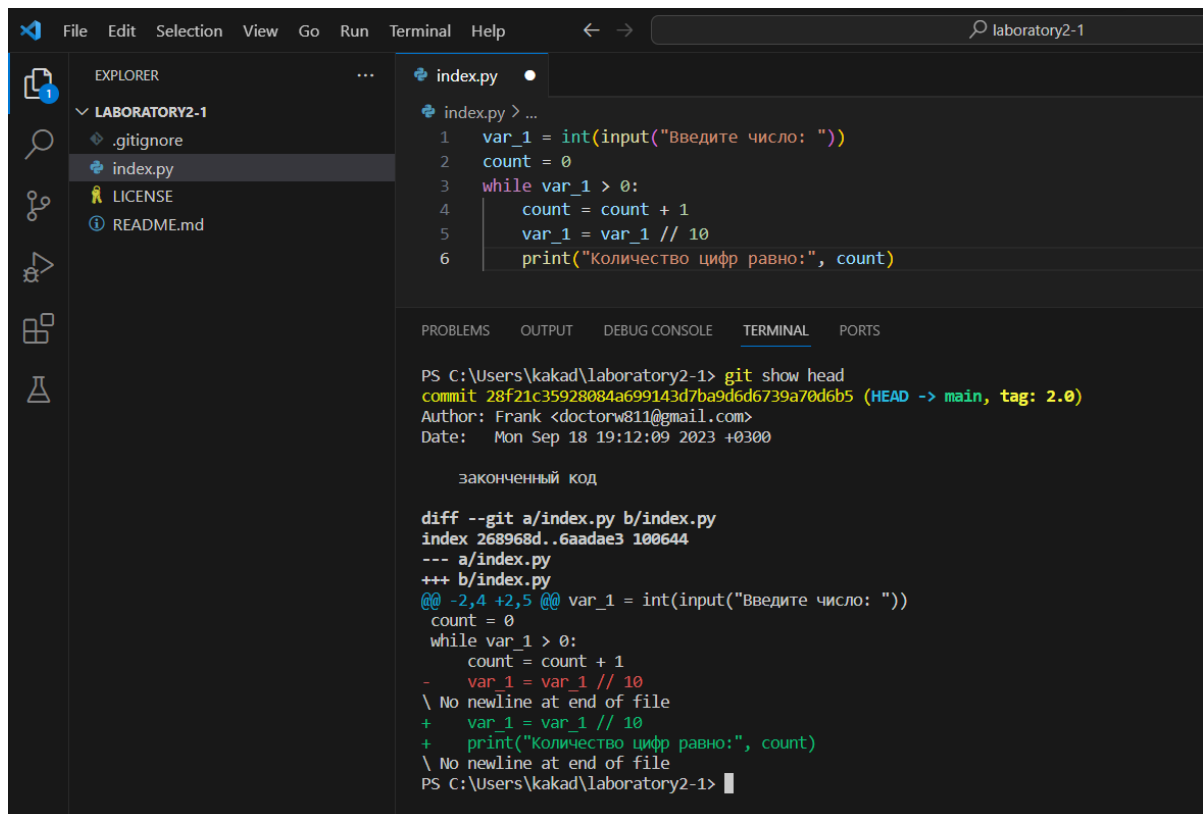


The screenshot shows the Visual Studio Code interface with the Explorer panel on the left displaying the file structure of 'LABORATORY2-1' (including .gitignore, index.py, LICENSE, and README.md). The main editor area shows the code for 'index.py'. The bottom panel, labeled 'TERMINAL', displays the output of the command 'git log --graph --pretty=oneline --abbrev-commit'. The output shows a list of commits with their hashes, branch names, and descriptions.

```
PS C:\Users\kakat\laboratory2-1> git log --graph --pretty=oneline --abbrev-commit
* 28f21c3 (HEAD -> main, tag: 2.0) законченный код
* 9fb5b82 (tag: 1.2) написание кода
* 5fa625d (tag: 1.0) файл питон
* 078aa2c (origin/main, origin/HEAD) update
* 3bf73f4 Initial commit
PS C:\Users\kakat\laboratory2-1>
```

Рисунок 2.8 — История хранилища

8. Посмотрел содержимое коммитов командой «git show head».



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left displaying the file structure of 'LABORATORY2-1'. The main editor area shows the code for 'index.py'. The bottom panel, labeled 'TERMINAL', displays the output of the command 'git show head'. The output shows the commit hash, author, date, and the diff of the commit.

```
PS C:\Users\kakat\laboratory2-1> git show head
commit 28f21c35928084a699143d7ba9d6d6739a70d6b5 (HEAD -> main, tag: 2.0)
Author: Frank <doctorw811@gmail.com>
Date: Mon Sep 18 19:12:09 2023 +0300

законченный код

diff --git a/index.py b/index.py
index 268968d..6aadae3 100644
--- a/index.py
+++ b/index.py
@@ -2,4 +2,5 @@ var_1 = int(input("Введите число: "))
count = 0
while var_1 > 0:
    count = count + 1
-   var_1 = var_1 // 10
\ No newline at end of file
+   var_1 = var_1 // 10
+   print("Количество цифр равно:", count)
\ No newline at end of file
PS C:\Users\kakat\laboratory2-1>
```

Рисунок 2.9 — Содержание коммитов

9. Удали весь код из файла `index.py` репозитория и сохранил изменения. Далее удалил все несохраненные изменения в файле командой: «`git checkout -- index.py`».

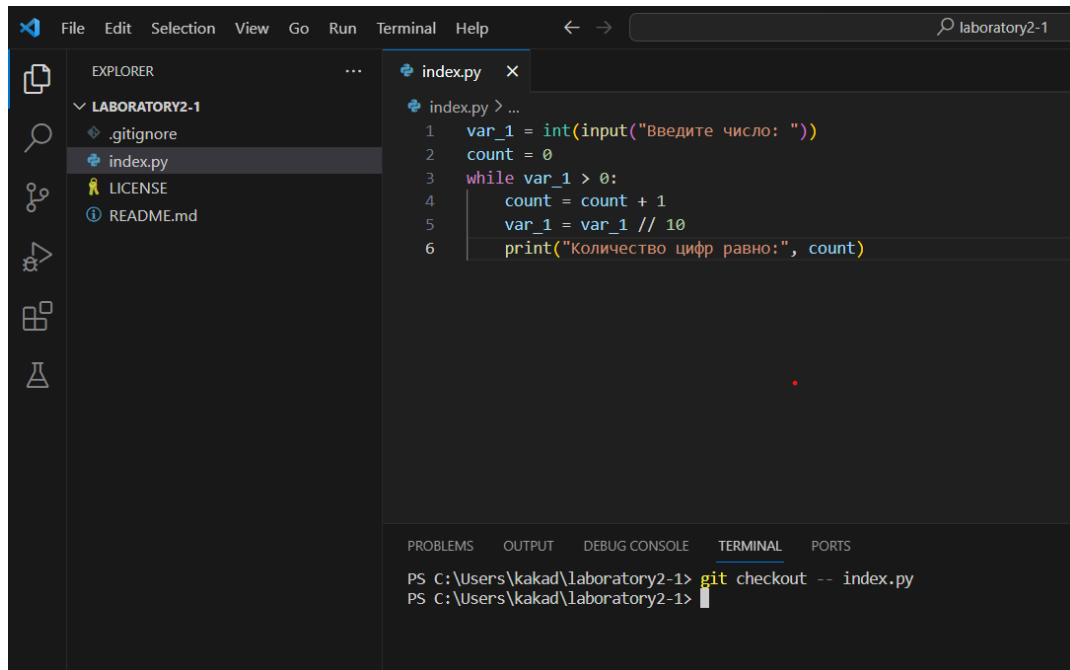


Рисунок 2.10 — Результат изменений файла `index.py`

Снова удалил код из файла `index.py` и сохранил изменения. Далее откатился состояние хранилища к предыдущей версии командой: «`git reset --hard HEAD~1`».

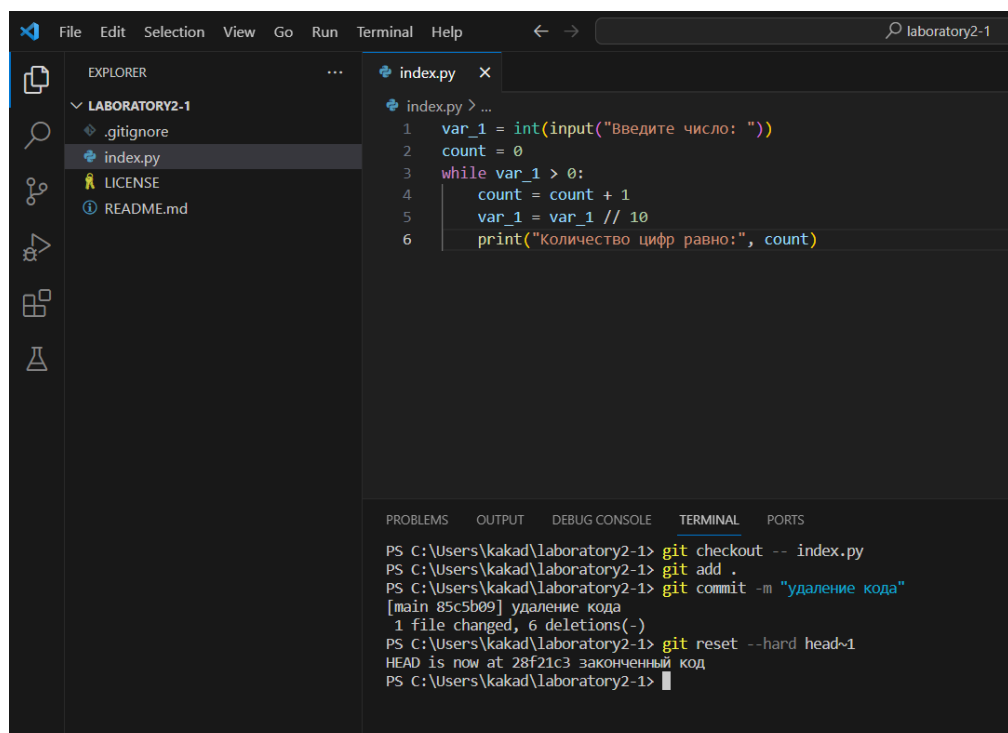


Рисунок 2.11 — Результат изменений файла `index.py`

## Ответы на контрольные вопросы:

1. Как выполнить историю коммитов в Git? Какие существуют дополнительные опции для просмотра истории коммитов?

С помощью команды «git log».

Одним из самых полезных аргументов является -p или --patch , который показывает разницу (выводит патч), внесенную в каждый коммит. Так же вы можете ограничить количество записей в выводе команды; используйте параметр -2 для вывода только двух записей. Так же есть возможность использовать серию опций для обобщения. Например, если вы хотите увидеть сокращенную статистику для каждого коммита, вы можете использовать опцию --stat . Следующей действительно полезной опцией является --pretty . Эта опция меняет формат вывода. Существует несколько встроенных вариантов отображения. Опция oneline выводит каждый коммит в одну строку, что может быть очень удобным если вы просматриваете большое количество коммитов.

2. Как ограничить вывод при просмотре истории коммитов?

В дополнение к опциям форматирования вывода, команда git log принимает несколько опций для ограничения вывода — опций, с помощью которых можно увидеть определённое подмножество коммитов. Вы уже видели одну из таких опций — это опция -2, которая показывает только последние два коммита. В действительности вы можете использовать -<n>, где n — это любое натуральное число и представляет собой n последних коммитов. На практике вы не будете часто использовать эту опцию, потому что Git по умолчанию использует постраничный вывод и вы будете видеть только одну страницу за раз.

3. Как внести изменения в уже сделанный коммит?

Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту. Если



вы хотите переделать коммит — внесите необходимые изменения, добавьте их в индекс и сделайте коммит ещё раз, указав параметр `–amend`.

4. Как отменить индексацию файла в Git?

С помощью команды `«git reset»`.

5. Как отменить изменения в файле?

С помощью команды `«git checkout <имя файла>»`.

6. Что такое удаленный репозиторий Git?

Удалённые репозитории представляют собой версии вашего проекта, сохранённые в интернете или ещё где-то в сети. У вас может быть несколько удалённых репозиториев, каждый из которых может быть доступен для чтения или для чтения-записи. Взаимодействие с другими пользователями предполагает управление удалёнными репозиториями, а также отправку и получение данных из них. Управление репозиториями включает в себя как умение добавлять новые, так и умение удалять устаревшие репозитории, а также умение управлять различными удалёнными ветками, объявлять их отслеживаемыми или нет и так далее. В данном разделе мы рассмотрим некоторые из этих навыков.

7. Как выполнить просмотр удаленных репозиториев данного локального репозитория?

Для того, чтобы просмотреть список настроенных удалённых репозиториев, вы можете запустить команду `git remote`. Она выведет названия доступных удалённых репозиториев. Если вы клонировали репозиторий, то увидите как минимум `origin` — имя по умолчанию, которое Git даёт серверу, с которого производилось клонирование.

8. Как добавить удаленный репозиторий для данного локального репозитория?

Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (`shortname`), просто выполните команду `git remote add <shortname> <url>`.

9. Как выполнить отправку/получение изменений с удаленного репозитория?

Как вы только что узнали, для получения данных из удалённых проектов, следует выполнить: `git fetch <remote-name>` Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет. После того как вы выполнили команду, у вас должны появиться ссылки на все ветки из этого удалённого проекта, которые вы можете просмотреть или слить в любой момент.

#### 10. Как выполнить просмотр удаленного репозитория?

Если хотите получить побольше информации об одном из удалённых репозиториях, вы можете использовать команду `git remote show <remote>` Выполнив эту команду с некоторым именем, например, `origin`

#### 11. Каково назначение тэгов Git?

Как и большинство СКВ, Git имеет возможность помечать определённые моменты в истории как важные. Как правило, эта функциональность используется для отметки моментов выпуска версий (v1.0, и т. п.). Такие пометки в Git называются тегами.

#### 12. Как осуществляется работа с тэгами Git?

Git использует два основных типа тегов: легковесные и аннотированные. Легковесный тег — это что-то очень похожее на ветку, которая не изменяется — просто указатель на определённый коммит. А вот аннотированные теги хранятся в базе данных Git как полноценные объекты. Они имеют контрольную сумму, содержат имя автора, его e-mail и дату создания, имеют комментарий и могут быть подписаны и проверены с помощью GNU Privacy Guard (GPG). Обычно рекомендуется создавать аннотированные теги, чтобы иметь всю перечисленную информацию; но если вы хотите сделать временную метку или по какой-то причине не хотите сохранять остальную информацию, то для этого годятся и легковесные.