

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1
дисциплины «Программирование на Python»

Выполнил:
Мотовилов Вадим Борисович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Информатика и вычислительная
техника», очная форма обучения

(подпись)

Руководитель практики:
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Цель: исследовать базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub.

Порядок выполнения работы:

1. Создал новый репозиторий

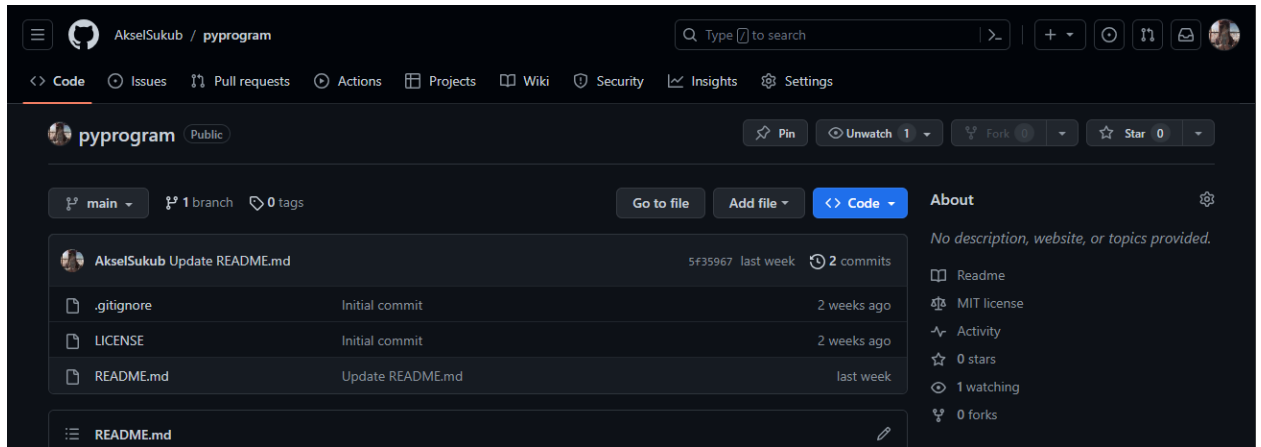


Рисунок 1. Созданный репозиторий

2. Копировал репозиторий на компьютер

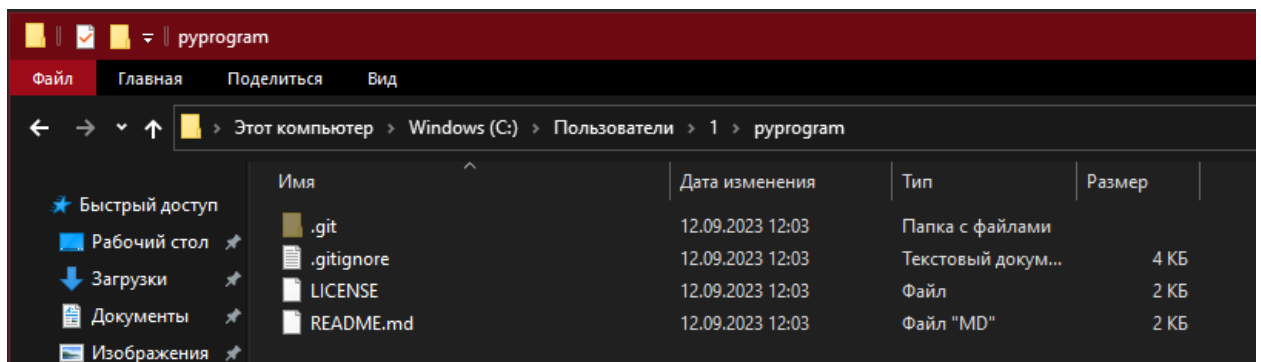


Рисунок 2. Размещение репозитория на компьютер

3. Изменил файл .gitignore

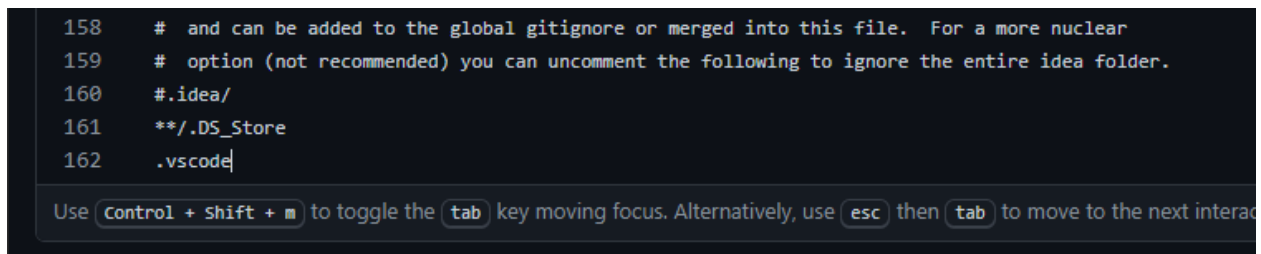


Рисунок 3. Измененный файл .gitignore

4. Добавил текст в README.md

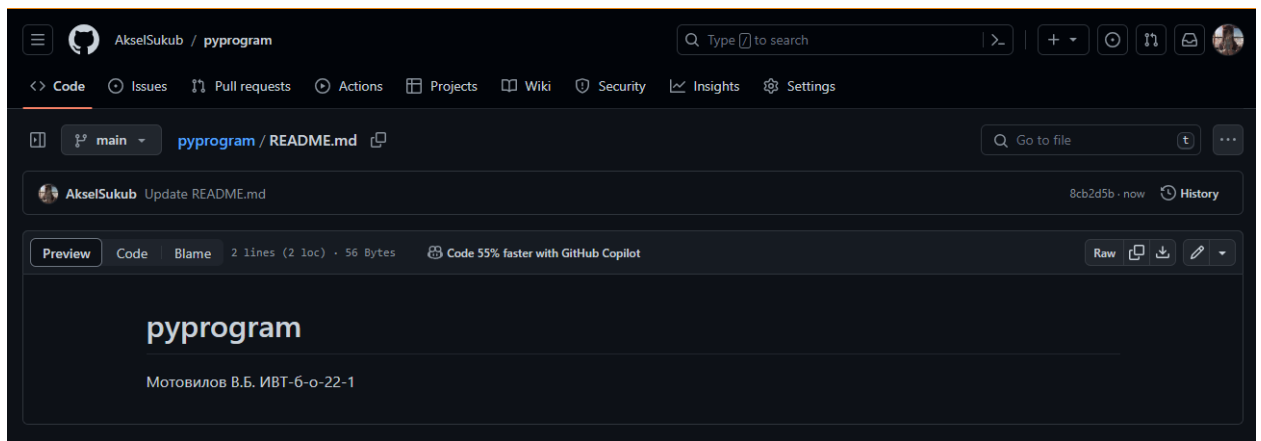


Рисунок 4. Добавленные изменения в README.md

5. Внес изменения в программу

```
1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram (main)
$ git push
Everything up-to-date

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram (main)
$ git add .

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram (main)
$ git commit -m "NewCommit1"
[main ef0a580] NewCommit1
1 file changed, 1 insertion(+)
create mode 100644 nc/C.py

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram (main)
$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 317 bytes | 105.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/AkselSukub/pyprogram.git
4f35884..ef0a580 main -> main

1@LAPTOP-4CDC56NI MINGW64 ~/pyprogram (main)
$ |
```

Рисунок 5. Создание нового коммита

6. Создал не менее 7 коммитов

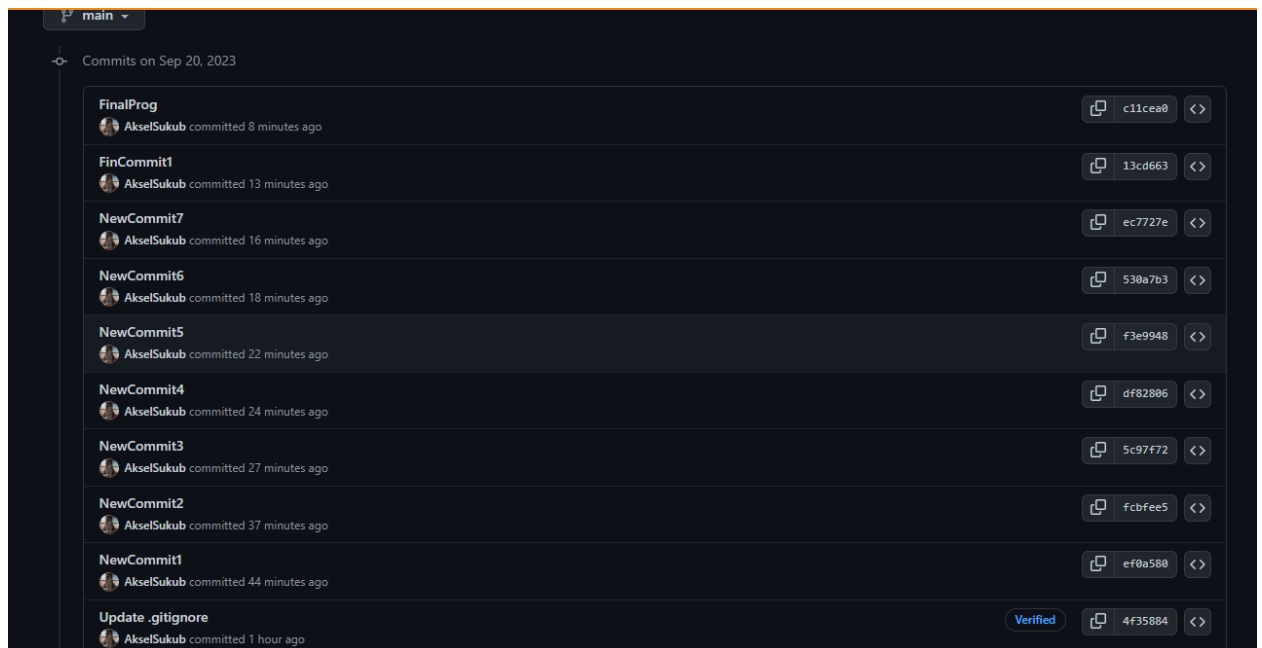
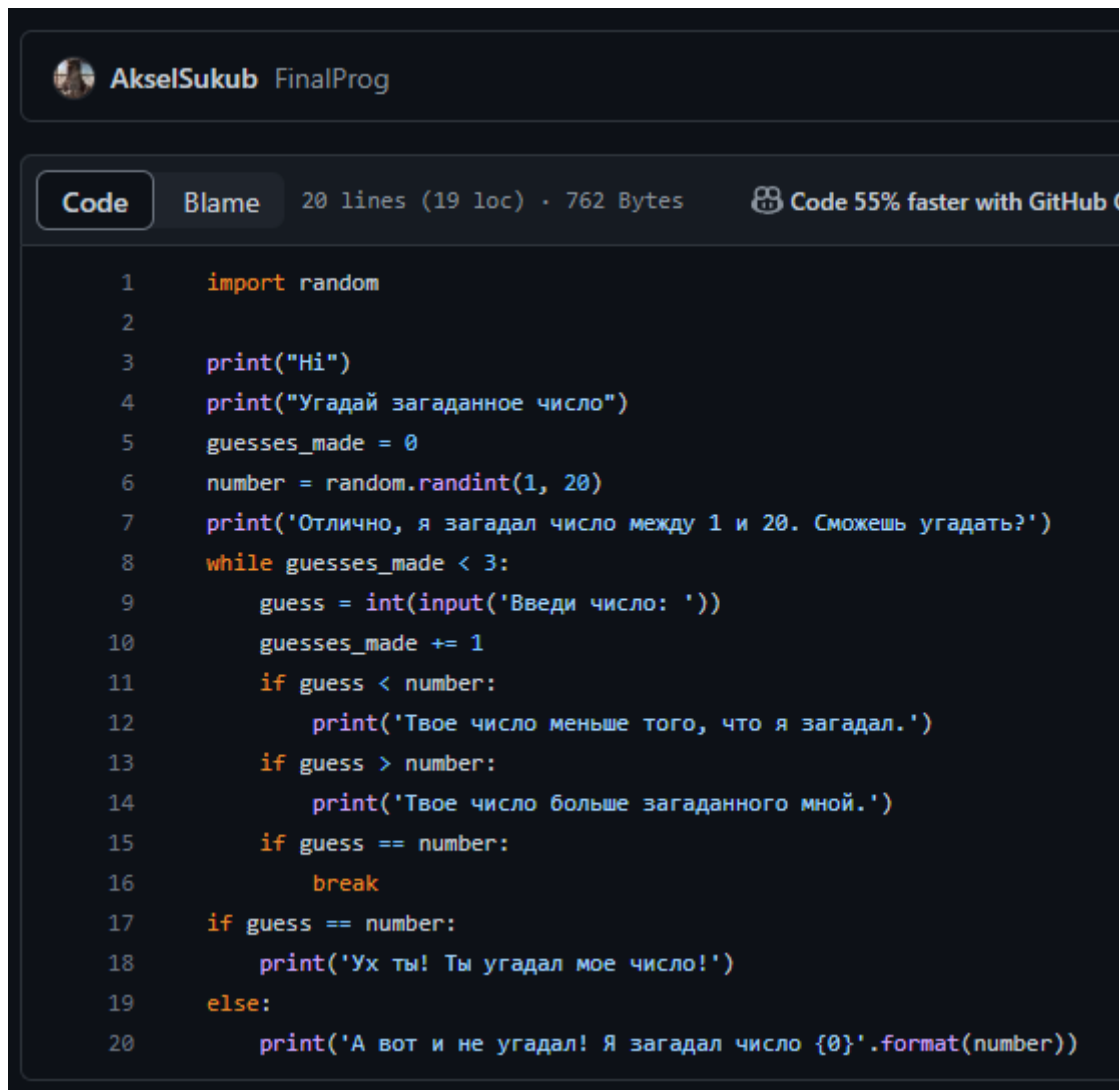


Рисунок 6. Все созданные коммиты

7. Завершил создание программы



```
1  import random
2
3  print("Hi")
4  print("Угадай загаданное число")
5  guesses_made = 0
6  number = random.randint(1, 20)
7  print('Отлично, я загадал число между 1 и 20. Сможешь угадать?')
8  while guesses_made < 3:
9      guess = int(input('Введи число: '))
10     guesses_made += 1
11     if guess < number:
12         print('Твое число меньше того, что я загадал.')
13     if guess > number:
14         print('Твое число больше загаданного мной.')
15     if guess == number:
16         break
17 if guess == number:
18     print('Ух ты! Ты угадал мое число!')
19 else:
20     print('А вот и не угадал! Я загадал число {0}'.format(number))
```

Рисунок 7. Созданная программа

Ответы на контрольные вопросы:

1. Что такое СКВ и каково ее назначение?

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

2. В чем недостатки локальных и централизованных СКВ?

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию. Такой подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели. Это недостаток локальной СКВ. Если сервер выйдет из строя на час,

то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Это недостаток централизованной СКВ.

3. К какой СКВ относится Git?

Git относится к распределённым системам контроля версий.

4. В чем концептуальное отличие Git от других СКВ?

Основное отличие Git от любой другой СКВ — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени. Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, поток снимков

5. Как обеспечивается целостность хранимых данных в Git?

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме. Это значит, что невозможно изменить содержимое файла или директории так, чтобы Git не узнал об этом. Данная функциональность встроена в Git на низком уровне и является неотъемлемой частью его основы.

Вы не потеряете информацию во время её передачи и не получите повреждённый файл без ведома Git.

6. В каких состояниях могут находиться файлы в Git?

Как связаны эти состояния? У Git есть три основных состояния, в которых могут находиться ваши файлы: зафиксированное (committed), изменённое (modified) и подготовленное (staged). Зафиксированный значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

8. Что такое профиль пользователя в GitHub?

Профиль - это ваша публичная страница на GitHub, как и в социальных сетях. 8. Какие бывают репозитории в GitHub? Репозитории бывают публичные и приватные. Публичные: любой желающий в Интернете может ознакомиться с этим хранилищем. Вы выбираете, кто может взять на себя обязательства. Приватные: вы выбираете, кто может просматривать этот репозиторий и фиксировать его.

9. Укажите основные этапы модели работы с GitHub.

- 1) создание репозитория;
- 2) клонирование репозитория.
- 3) Работа в локальном репозитории.
- 4) С помощью команд `git add`, `git commit`, `git push` сохранить изменения в исходном репозитории

10. Как осуществляется первоначальная настройка Git после установки?

Чтобы убедиться, что Git был успешно установлен используем команду: `git version`; Если она сработала, давайте добавим в настройки Git ваше имя, фамилию и адрес электронной почты, связанный с вашей учетной записью GitHub: `git config --global user.name` `git config --global user.email`

11. Опишите этапы создания репозитория в GitHub.

В правом верхнем углу, рядом с аватаром есть кнопка с плюсиком, нажимая которую мы переходим к созданию нового репозитория. В результате будет выполнен переход на страницу создания репозитория. Наиболее важными на ней являются следующие поля: Имя репозитория. Оно может быть любое, необязательно уникальное во всем github, потому что привязано к вашему аккаунту, но уникальное в рамках тех репозиторий, которые вы создавали. Описание (Description). Можно оставить пустым. Public/private. Выбираем открытый (Public), НЕ ставим галочку “Initialize this repository with a README” (В README потом будет лежать какая-то основная информация, что же такое ваш проект и как с ним работать). .gitignore и LICENSE можно сейчас не выбирать. После заполнения этих полей нажимаем кнопку Create repository.

12. Какие типы лицензий поддерживаются GitHub при создании репозитория?

Apache License 2.0, GNU General Public License v3.0, MIT License, BSD 2-Clause "Simplified" License, BSD 3-Clause "New" or "Revised" License, Boost Software License 1.0, Creative Commons Zero v1.0 Universal, Eclipse Public License 2.0, GNU Affero General Public License v3.0, GNU General Public License v2.0 ,GNU Lesser General Public License v2.1 ,Mozilla Public License e2 .Unlicense

13. Как осуществляется клонирование репозитория GitHub?

Зачем нужно клонировать репозиторий? После создания репозитория его необходимо клонировать на ваш компьютер. Для этого на странице репозитория необходимо найти кнопку Clone или Code и щелкнуть по ней, чтобы отобразить адрес репозитория для клонирования. Откройте командную строку или терминал и перейдите в каталог, куда вы хотите скопировать хранилище. Затем напишите `git clone` и введите адрес.

14. Как проверить состояние локального репозитория Git?

Локальный репозиторий включает в себя все те же файлы, ветки и историю коммитов, как и удаленный репозиторий. Введите эту команду, чтобы проверить состояние вашего репозитория: `git status`.

15. Как изменяется состояние локального репозитория Git после выполнения следующих операций: добавления/изменения файла в локальный репозиторий Git; добавления нового/измененного файла под версионный контроль с помощью команды `git add` ; фиксации (коммита) изменений с помощью команды `git commit` и отправки изменений на сервер с помощью команды `git push`?

Добавление/изменение файла в локальный репозиторий Git: Когда вы добавляете новый файл или вносите изменения в существующий файл, Git отслеживает эти изменения, но они ещё не зафиксированы. Добавление нового файла под версионный контроль с помощью команды `git add`: Команда `git add` добавляет файл(ы) в индекс (staging area), который является промежуточной областью между рабочей копией и коммитами. После выполнения этой команды, состояние индекса будет обновлено для отслеживания добавленных изменений. Фиксация (коммит) изменений с помощью команды `git commit`: Команда `git commit` создает коммит из всех файлов, находящихся в индексе на момент выполнения команды `git commit`. Этот коммит содержит фиксированный набор изменений и связывается с уникальным идентификатором (хэш-кодом). Состояние вашего локального репозитория теперь имеет зафиксированный (committed) код. Отправка изменений на сервер с помощью команды `git push`: Команда `git push` отправляет все зафиксированные коммиты из вашего локального репозитория на удаленный репозиторий. После успешной отправки изменений, состояние вашего локального репозитория и удаленного репозитория станут одинаковыми.

16. У Вас имеется репозиторий на GitHub и два рабочих компьютера, с помощью которых Вы можете осуществлять работу над некоторым проектом с использованием этого репозитория. Опишите последовательность команд, с

помощью которых оба локальных репозитория, связанных с репозиторием GitHub будут находиться в синхронизированном состоянии. Примечание: описание необходимо начать с команды `git clone`.

1) Клонировать репозиторий на каждый из компьютеров, используя команду `git clone` и ссылку.

2) При изменении локального репозитория используем последовательность команд: `git add`, `git commit`, `git push`.

3) Для синхронизации изменений, созданных другим рабочим компьютером используем команду `git pull`.

17. GitHub является не единственным сервисом, работающим с Git. Какие сервисы еще Вам известны?

Приведите сравнительный анализ одного из таких сервисов с GitHub. GitHub - это один из самых популярных сервисов для работы с Git. Однако, есть и другие сервисы, которые также предоставляют возможность работать с Git. Некоторые из них: GitLab, Bitbucket, SourceForge, GitKraken, и CodeCommit1. Сравнивая GitHub и GitLab, можно заметить, что оба сервиса имеют много общего. Они оба предоставляют возможность управления репозиториями Git и инструменты для управления проектами. Однако, GitLab является более гибким и настраиваемым сервисом. Он позволяет настроить свою собственную инфраструктуру для хранения кода и управления проектами. Кроме того, он предоставляет множество инструментов для управления проектами, включая возможность создания задач, управления релизами и т.д.

18. Интерфейс командной строки является не единственным и далеко не самым удобным способом работы с Git. Какие Вам известны программные средства с графическим интерфейсом пользователя для работы с Git?

Приведите как реализуются описанные в лабораторной работе операции Git с помощью одного из таких программных средств. Кроме интерфейса командной строки, существует множество программных средств с графическим интерфейсом пользователя для работы с Git. Некоторые из

них: GitHub Desktop, GitKraken, SourceTree, TortoiseGit, Git Extensions, и Magit. GitHub Desktop - это приложение, предоставляющее графический интерфейс пользователя для работы с Git на локальном компьютере. Оно обладает различными инструментами для управления репозиториями Git и выполнения операций над ними. С помощью GitHub Desktop можно выполнять все необходимые операции из описанной лабораторной работы, включая `git clone`, `git status`, `git add`, `git commit`, `git push` и `git pull`. Например, для выполнения операции `git clone` в GitHub Desktop необходимо выбрать опцию "Clone a repository" в меню "File" и указать URL репозитория. Для проверки статуса изменений файлов используется команда `git status`: после выбора соответствующего репозитория из списка доступных отображается информация о состоянии файлов в окне "Changes". Чтобы добавить файлы в индекс (`git add`), нужно выделить требуемые файлы и нажать кнопку "Stage all changes". Затем сообщение коммита пишется в поле соответствующего окна и происходит фиксация изменений с помощью кнопки "Commit to master". Для отправки изменений на удаленный сервер используется команда `git push`: опция "Push origin" выбирается из меню "Repository". А чтобы получить последние изменения с удаленного сервера (`git pull`), следует выбрать опцию "Pull origin" в меню "Repository".

Вывод: в результате выполнения работы я исследовал базовые возможности системы контроля версий Git и веб-сервиса для хостинга GitHub.