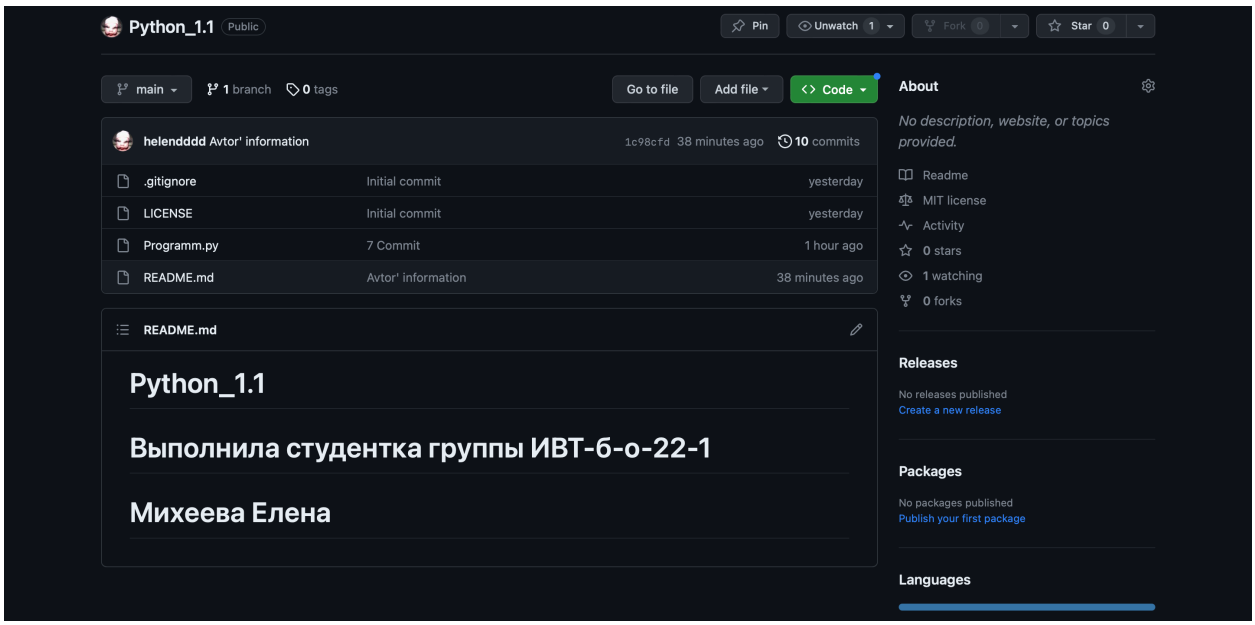
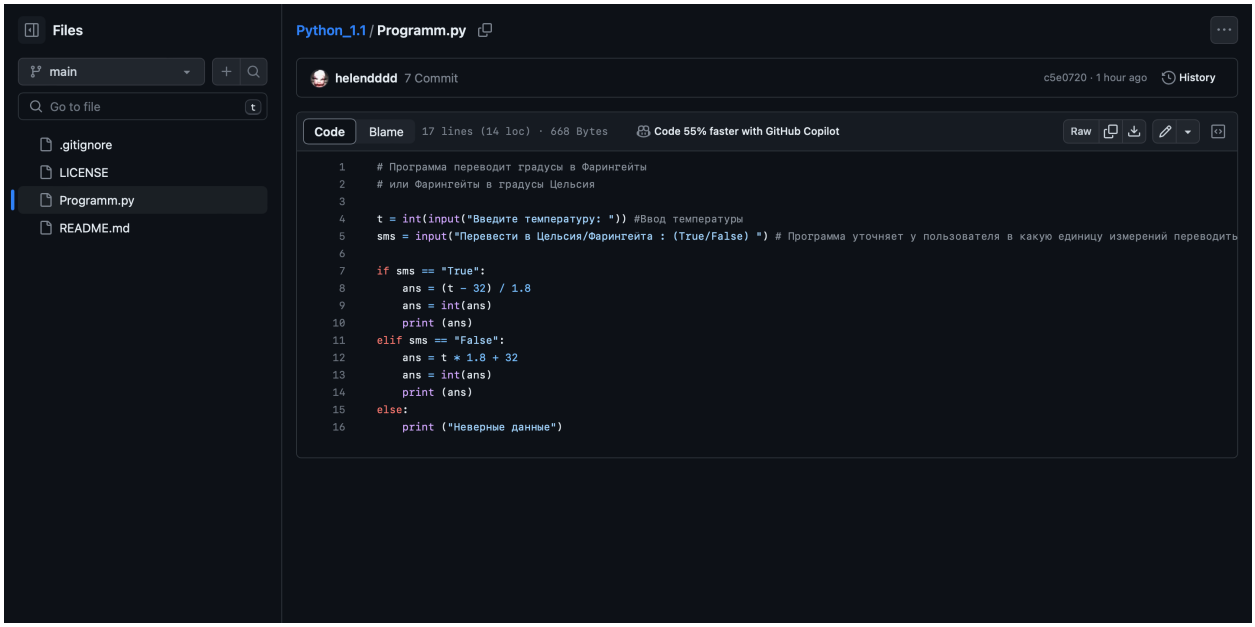
```
MacBook-Pro-Elena:~ mikheeva$ git config --global core.autocrlf input
MacBook-Pro-Elena:~ mikheeva$ git config --global core.safecrlf warn
MacBook-Pro-Elena:~ mikheeva$ git config --global core.quotepath off
MacBook-Pro-Elena:~ mikheeva$ git clone https://github.com/helendddd/Python_1.1.git
Клонирование в «Python_1.1»...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Получение объектов: 100% (5/5), готово.
MacBook-Pro-Elena:~ mikheeva$ cd python_1.1
MacBook-Pro-Elena:python_1.1 mikheeva$ git init
Переинициализирован существующий репозиторий Git в /Users/mikheeva/Python_1.1/.git/
MacBook-Pro-Elena:python_1.1 mikheeva$ git status
Текущая ветка: main
Эта ветка соответствует «origin/main».

нечего коммитить, нет изменений в рабочем каталоге
MacBook-Pro-Elena:python_1.1 mikheeva$ git add Programm.py
MacBook-Pro-Elena:python_1.1 mikheeva$ git add Programm.py
MacBook-Pro-Elena:python_1.1 mikheeva$ git status
Текущая ветка: main
Ваша ветка опережает «origin/main» на 1 коммит.
  (используйте «git push», чтобы опубликовать ваши локальные коммиты)

Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
        изменено:      Programm.py

MacBook-Pro-Elena:python_1.1 mikheeva$ git commit -m "First Commit"
[main 479c5ee] First Commit
 1 file changed, 1 insertion(+)
MacBook-Pro-Elena:python_1.1 mikheeva$ git add Programm.py
MacBook-Pro-Elena:python_1.1 mikheeva$ git commit -m "Second Commit"
[main 6cfe517] Second Commit
 1 file changed, 1 insertion(+)
```

```
MacBook-Pro-Elena:python_1.1 mikheeva$ git add README.md
MacBook-Pro-Elena:python_1.1 mikheeva$ git status
Текущая ветка: main
Ваша ветка опережает «origin/main» на 8 коммитов.
  (используйте «git push», чтобы опубликовать ваши локальные коммиты)

Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
        изменено:      README.md
```

```
MacBook-Pro-Elena:python_1.1 mikheeva$ git push
Username for 'https://github.com': helendddd
Password for 'https://helendddd@github.com':
Перечисление объектов: 25, готово.
Подсчет объектов: 100% (25/25), готово.
При сжатии изменений используется до 8 потоков
Сжатие объектов: 100% (24/24), готово.
Запись объектов: 100% (24/24), 2.17 КиБ | 742.00 КиБ/с, готово.
Всего 24 (изменений 14), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (14/14), done.
To https://github.com/helendddd/Python_1.1.git
   e2977d1..c5e0720  main -> main
```





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?

Недостатки централизированных СКВ:

1) Единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений.

Недостатки локальных СКВ:

1) Отсутствие централизованного хранилища: Локальные системы контроля версий не предоставляют централизованного сервера, где хранятся все версии проекта. Это может сделать сложным совместное сотрудничество и обмен изменениями между разработчиками.

2) Ограниченные средства управления доступом: Управление доступом к репозиторию более сложно в локальных системах контроля версий по сравнению с централизованными системами, что может привести к проблемам с безопасностью и конфиденциальностью данных.

3. К какой СКВ относится Git?

Централизированной.

4. В чем концептуальное отличие Git от других СКВ?

Большинство других систем хранят информацию в виде списка изменений в файлах, Git для увеличения эффективности, если файлы не были изменены, не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как поток снимков.

5. Как обеспечивается целостность хранимых данных в Git?

Целостность хранимых данных в Git обеспечивается с помощью криптографических хеш-функций, которые генерируют уникальные хеш-коды для каждого объекта в репозитории. Эти хеш-коды зависят от содержания объектов и используются для проверки их целостности при каждой операции. Кроме того, Git использует проверку хеш-сумм и поддерживает распределенное хранение данных для обеспечения надежности и целостности информаци**и.**

6. В каких состояниях могут находится файлы в Git? Как связаны эти состояния?

У файлов в git может быть 3 состояния: зафиксированное (committed), изменённое (modified) и подготовленное (staged). Если определённая версия файла есть в Git-директории, эта версия считается зафиксированной. Если

версия файла изменена и добавлена в индекс, значит, она подготовлена. И если файл был изменён с момента последнего распаковывания из репозитория, но не был добавлен в индекс, он считается изменённым.

7. Что такое профиль пользователя в GitHub?

Профиль пользователя GitHub - это виртуальная страница, которая представляет конкретного пользователя на платформе GitHub. Профиль содержит информацию о пользователе, его активности на GitHub и его участии в различных проектах.

8. Какие бывают репозитории в GitHub?

Приватные и публичные.

9. Укажите основные этапы модели работы с GitHub.

- Создание репозитория: Пользователь создает новый репозиторий, который может быть публичным (доступным для всех) или частным (доступным только для определенных пользователей или организаций). В репозитории хранится код, документация и другие файлы проекта.
- Работа с локальным репозиторием: Пользователь клонирует репозиторий с GitHub на свой локальный компьютер. Здесь он вносит изменения в код и файлы проекта с помощью команд Git.
- Коммиты: Пользователь делает коммиты, сохраняя свои изменения с сообщением описания.
- Синхронизация с GitHub: Пользователь загружает свои изменения на GitHub, отправляя коммиты на сервер. Это обновляет репозиторий на GitHub и делает изменения доступными для других разработчиков.

10. Как осуществляется первоначальная настройка Git после установки?

После установки Git необходимо настроить имя пользователя и адрес электронной почты. Для этого ввести следующие команды:

git config --global user.name "Ваше имя"

git config --global user.email "ваш@адрес.почты"

Эта информация будет связана с коммитами,

11. Опишите этапы создания репозитория в GitHub.

- Создать новый репозиторий с главной страницы GitHub
- Заполнить данные о репозитории: ввести название репозитория, его описание, определить вид доступности (Privat or Public), добавить файл READ.me, .gitignore или выбрать тип лицензии.
- Нажать на кнопку "Create repository" (Создать репозиторий), чтобы завершить процесс.
- После создания репозитория на GitHub, можно начать работу с ним: клонировать на компьютер и добавить код, файлы и другие ресурсы.

12. Какие типы лицензий поддерживаются GitHub при создании репозитория?

MIT License, GNU General Public License (GPL), Apache License, Creative Commons лицензии, BSD лицензии, Mozilla Public License, The Unlicense и тд.

13. Как осуществляется клонирование репозитория GitHub? Зачем нужно клонировать репозиторий?

Клонирование репозитория GitHub — это процесс создания локальной копии удаленного репозитория на компьютере. Клонирование репозитория позволяет работать над проектом локально, таким образом каждый разработчик может изолировать свои изменения от других членов команды.

Так же клонирование репозитория включает в себя всю историю изменения, что позволяет быстро перемещаться по истории коммитов и восстанавливать предыдущие версии проекта.

14. Как проверить состояние локального репозитория Git?

Для проверки состояния локального репозитория Git используют команду «git status». Она предоставляет информацию о текущем состоянии репозитория, количестве изменений и коммитов.

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 ?

После добавления или изменения файлов в локальном репозитории Git, они переходят в состояние "Modified". Команда git add переводит выбранные файлы в состояние "Staged", подготавливая их к коммиту. Завершив коммит с помощью команды git commit, изменения фиксируются в локальном репозитории, а команда git push отправляет их на удаленный сервер, обновляя состояние и синхронизируя локальный и удаленный репозитории.

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) Клонируем репозитория с GitHub:

git clone URL_РЕПОЗИТОРИЯ

2) Переходим в каталог, созданный при клонировании, с помощью команды «cd».

3) Перед началом работы получаем актуальную версию проекта из удаленного репозитория с помощью команды «git pull».

4) После работы над проектом добавляем и фиксируем изменения командами «git add» и «git commit».

5) Отправляем изменения на GitHub с помощью команды «git push».

17. GitHub является не единственным сервисом, работающим с Git. Какие сервисы еще Вам известны? Приведите сравнительный анализ одного из таких сервисов с GitHub.

GitLab — это платформа для управления кодом с функциональностью, аналогичной GitHub. Он предоставляет возможность создавать репозитории, отслеживать задачи, выполнять непрерывную интеграцию и развертывание, а

также управлять проектами. Один из основных отличительных моментов GitLab - это возможность самостоятельно развернуть его на вашем сервере.

GitHub и GitLab - оба сервиса для управления кодом с использованием Git. GitHub широко используется в сообществе с открытым исходным кодом и предоставляет бесплатные аккаунты для открытых проектов. GitLab, наоборот, предоставляет бесплатные аккаунты и более доступные коммерческие варианты, а также позволяет развертывать его на собственных серверах. Оба сервиса обеспечивают управление репозиториями и интеграцию с CI/CD, но GitLab также включает управление задачами и проектами. GitHub обладает более широкой экосистемой интеграций с различными инструментами разработки, такими как Slack и JIRA. GitLab также предоставляет интеграции, но их количество и разнообразие меньше.

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.

GitHub Desktop — это официальное приложение от GitHub, которое обеспечивает простой и интуитивно понятный интерфейс для работы с Git. С его помощью можно выполнять операции, такие как клонирование репозиториев, создание веток, фиксацию изменений и push в удаленный репозиторий.

Так же существуют такие программные средства с как GitKraken, Sourcetree, TortoiseGit и тд. Эти графические клиенты облегчают работу с Git для пользователей, предпочитающих интерфейсс визуальными элементами.

Пример выполнения операций Git с использованием GitHub Desktop:

- Чтобы клонировать репозиторий, необходимо открыть GitHub Desktop, выбрать опцию "File" -> "Clone Repository" и указать URL репозитория, а затем выбрать локальное местоположение.

- Для фиксации изменений, выделить файлы, которые необходимо зафиксировать, в окне GitHub Desktop, ввести сообщение коммита и нажать кнопку "Commit to <branch name>".
- Для отправки изменений на удаленный репозиторий, необходимо нажать кнопку "Push origin" на верхней панели в GitHub Desktop.