МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра

инфокоммуникаций

Институт цифрового

развития

ОТЧЁТ

по лабораторной работе №1.2

Дисциплина: «Основы программной инженерии»

Тема: «Исследование возможностей Git для работы с локальными репозиториями»

Выполнила:

студентка 2 курса

группы Пиж-б-о-21-1

Джолдошова Мээрим

Бекболотовна

Ставрополь 2022

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

Выполнение работы

1. Был создан общедоступный репозиторий lab_1.2 на GitHub в котором будет использована лицензия MIT и язык программирования C++

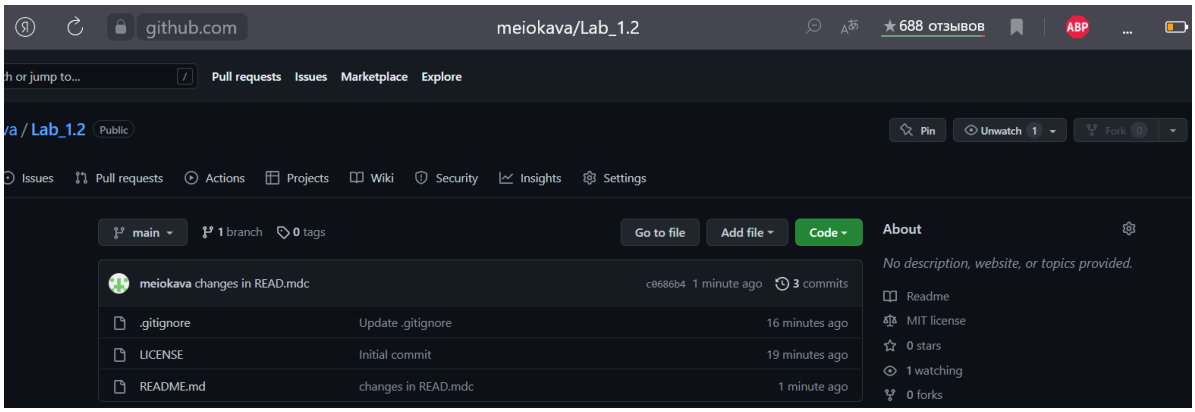

Рисунок 1 – Созданный репозиторий

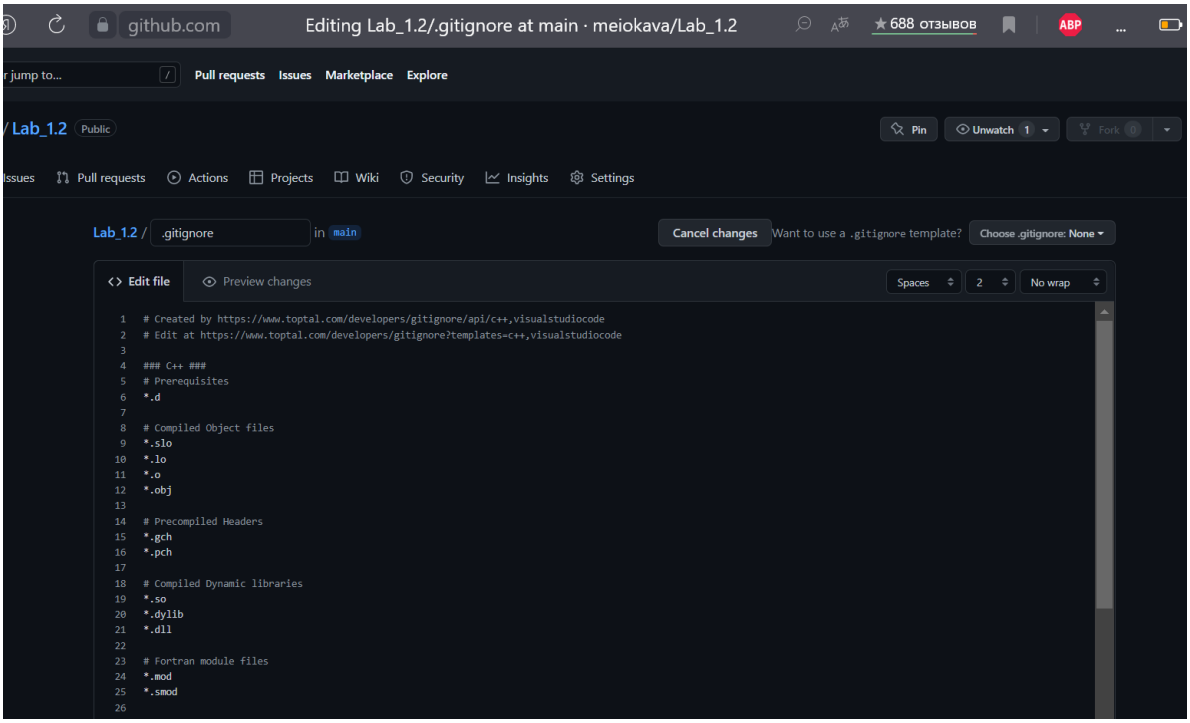

Рисунок 2 – Изменения в файле gitignor для выбранного мной языка программирования

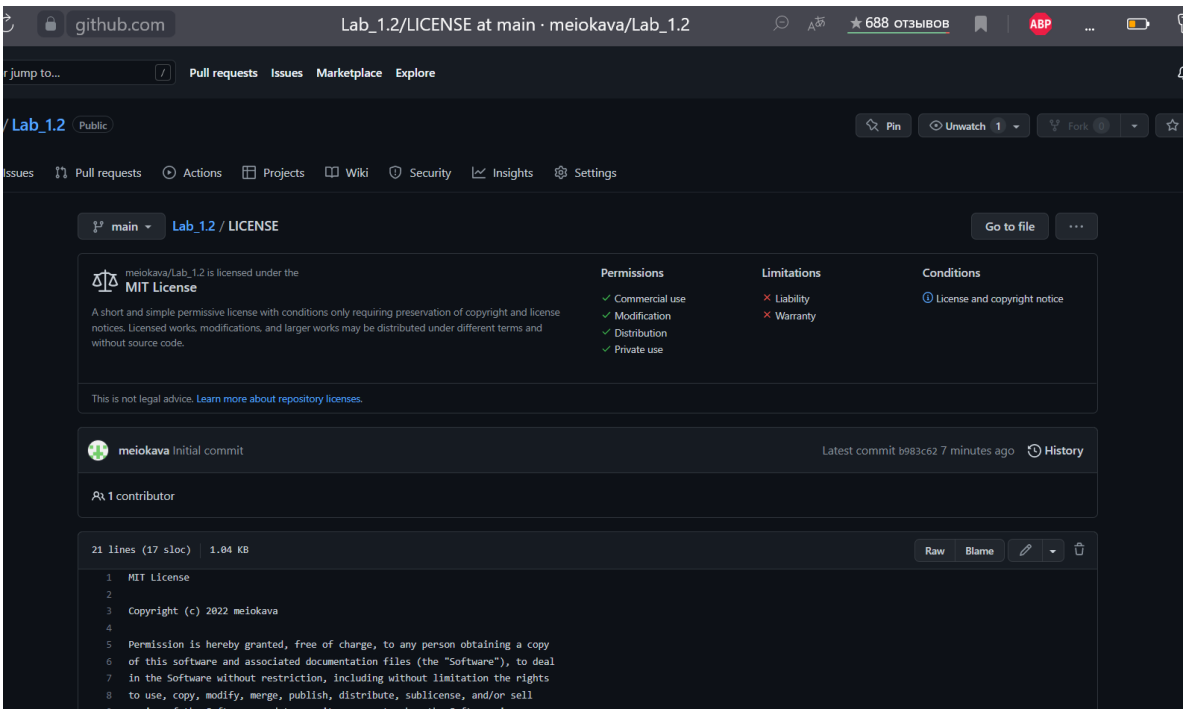

Рисунок 3 – Выбранная лицензия MIT

2. Был клонирован репозиторий на рабочий компьютер

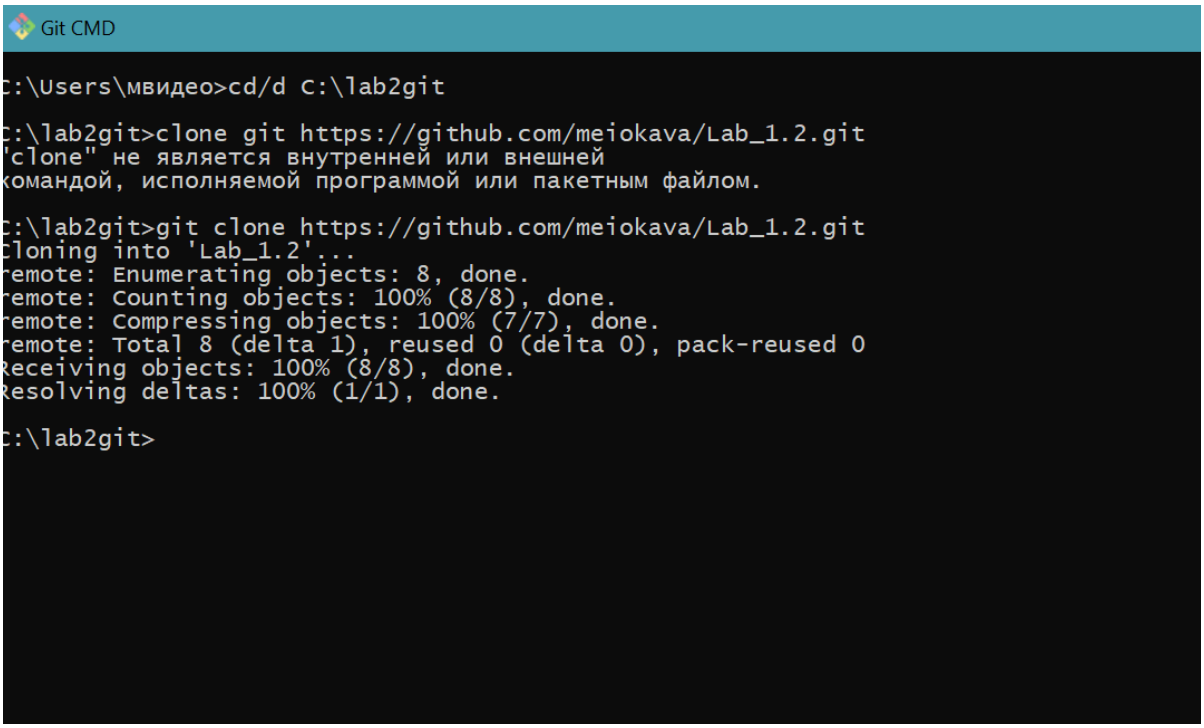

Рисунок 4 – Клонирование репозитория

3. Добавление информации в README

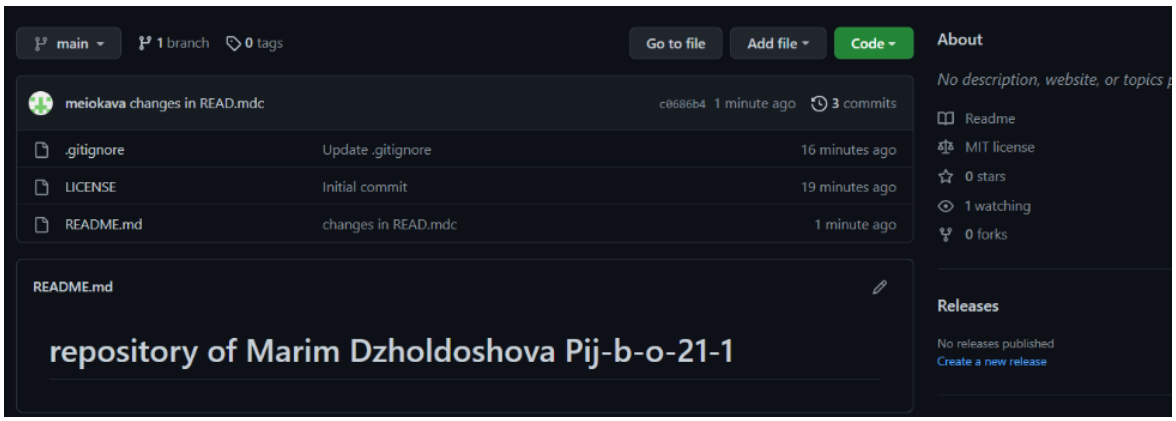

Рисунок 5 – Информация о себе в файле README

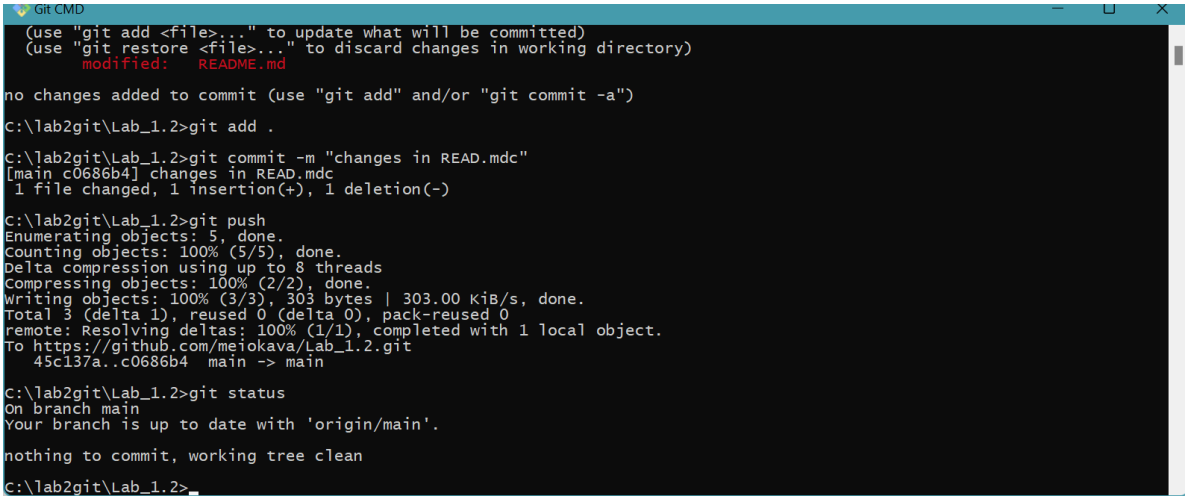

Рисунок 6 – Коммит файла README

4. Было сделано 8 коммитов и добавлено 3 тега

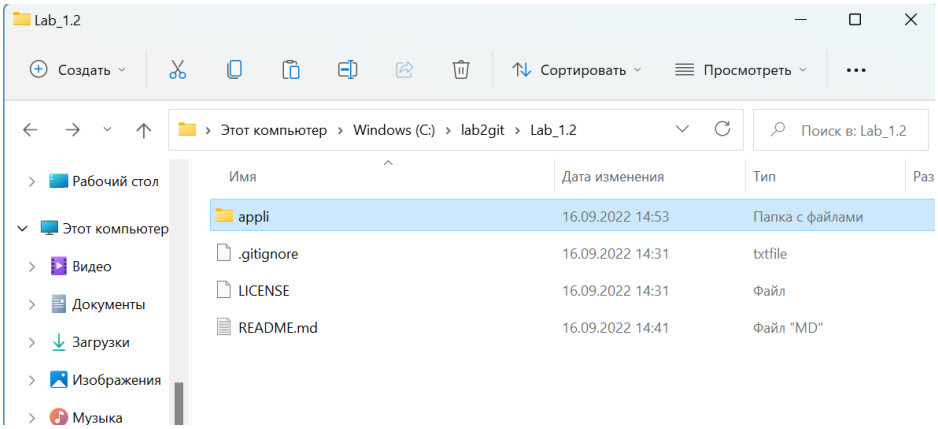

Рисунок 7 – Была создана папка для хранения проекта

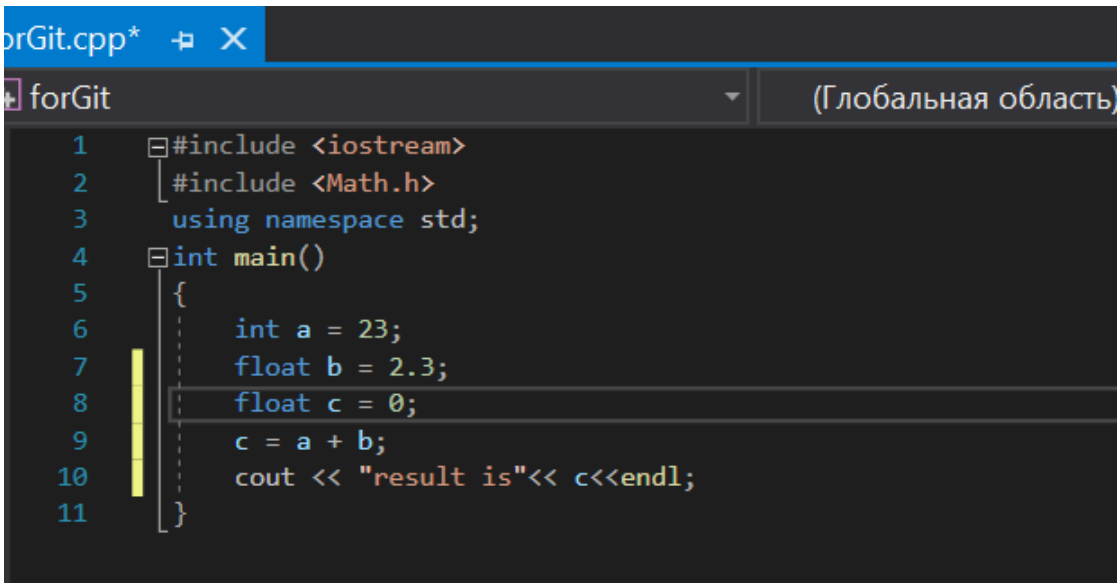

Рисунок 8 – Код программы

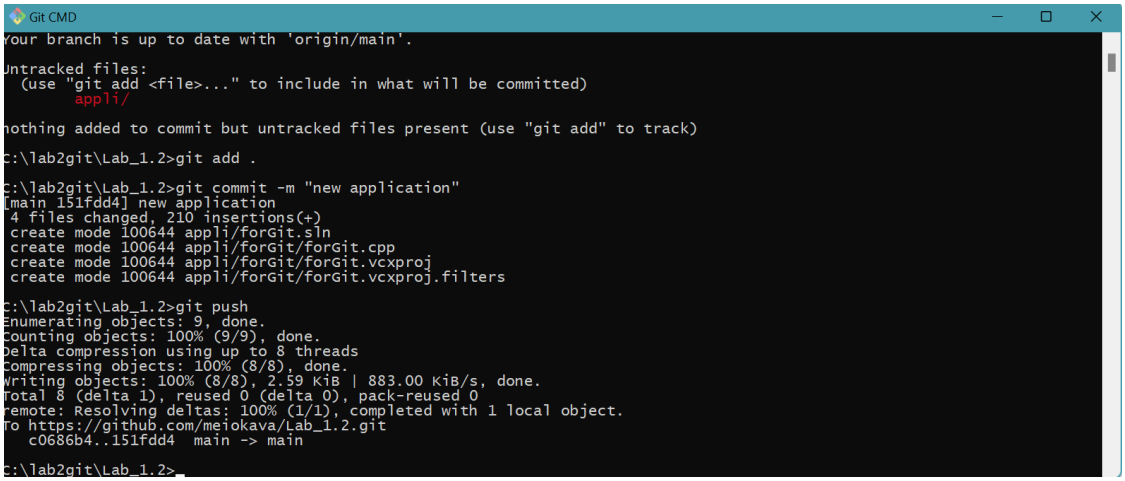

Рисунок 9 – Коммит и пуш программы на удаленный сервис

Рисунок 10 – Изменения на удаленном сервере

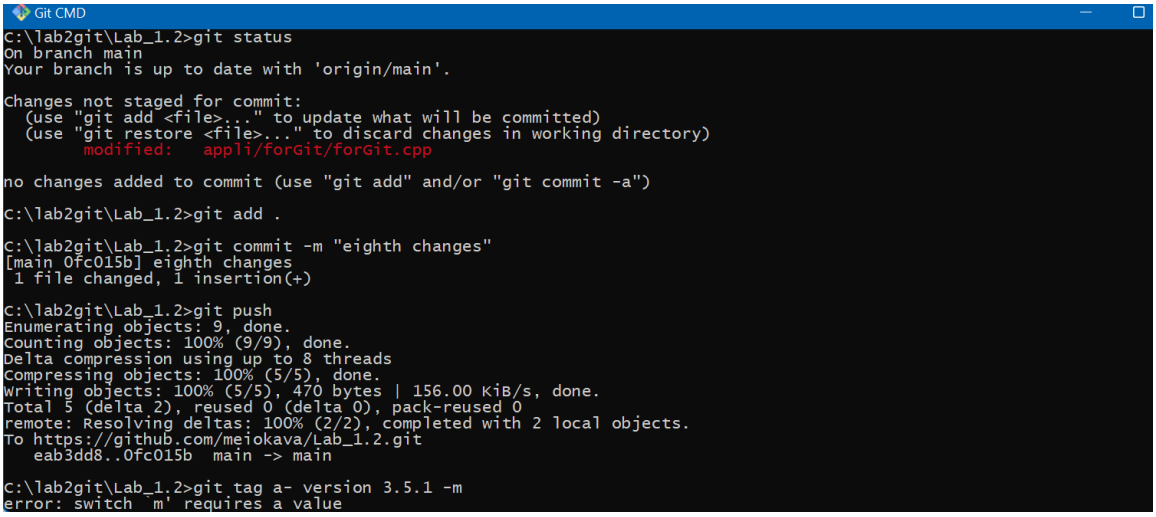

Рисунок 11 – Было сделано 8 коммитов

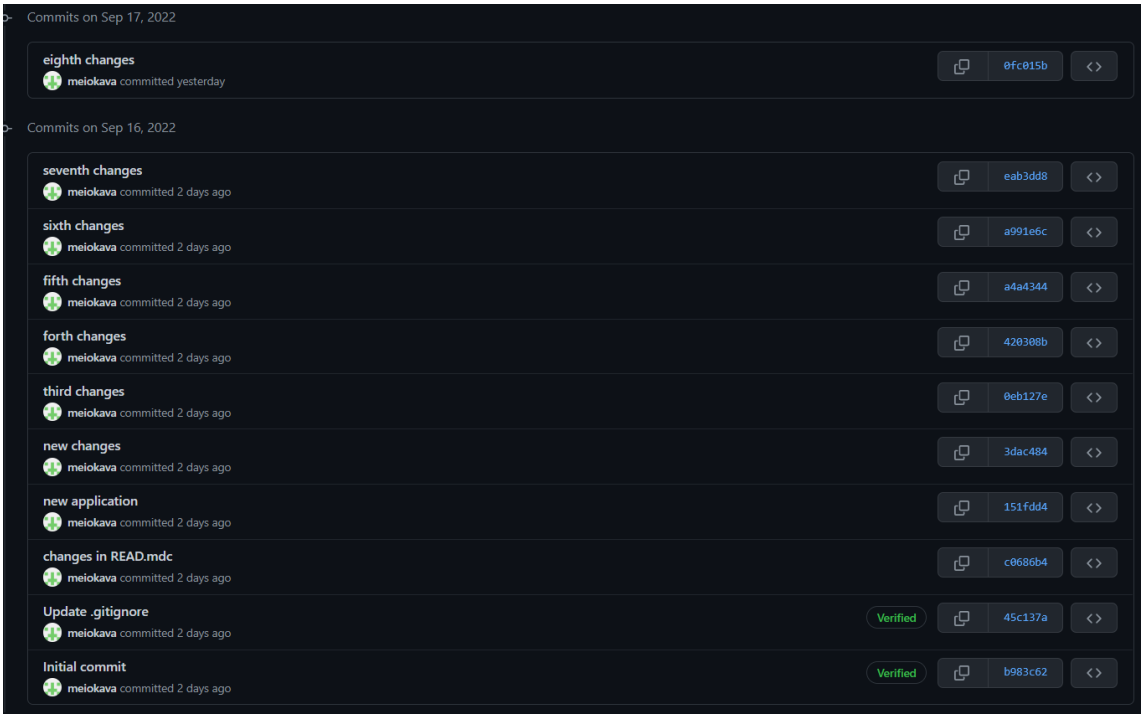

Рисунок 12 – История коммитов на удаленном сервере

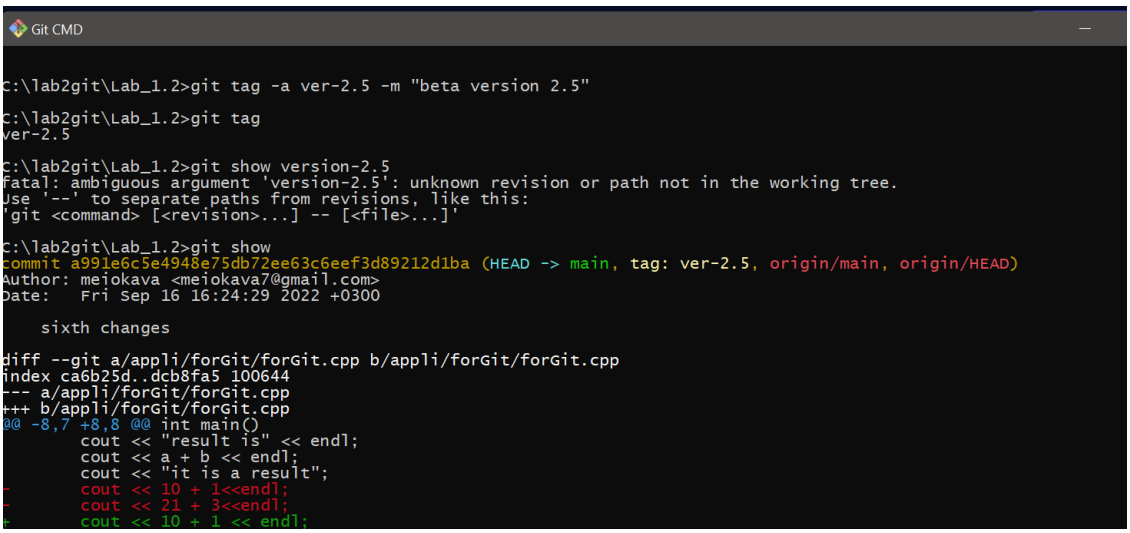

Рисунок 13 – Создание аннотированного тега

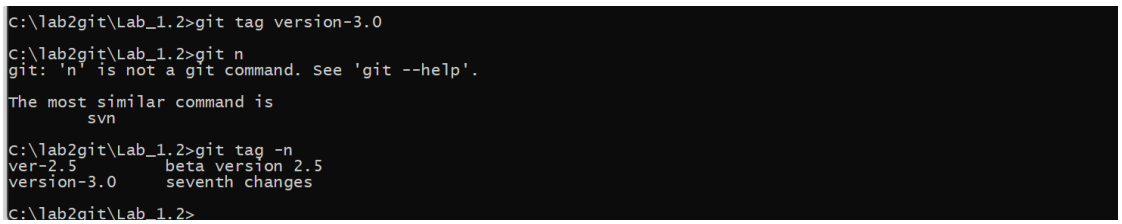

Рисунок 14 – Создание легковесного тега

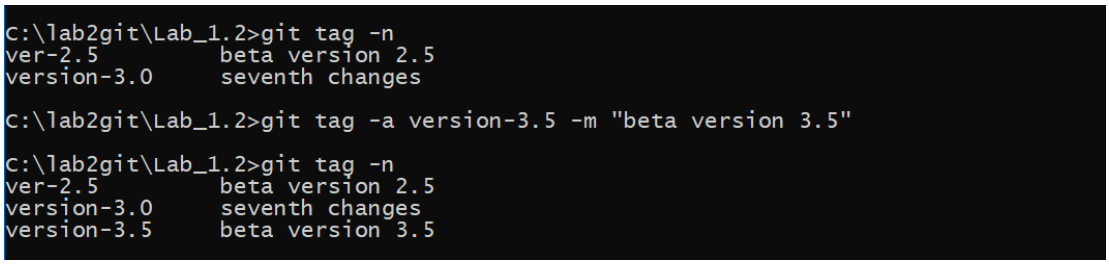

Рисунок 15 – Создание третьего тега

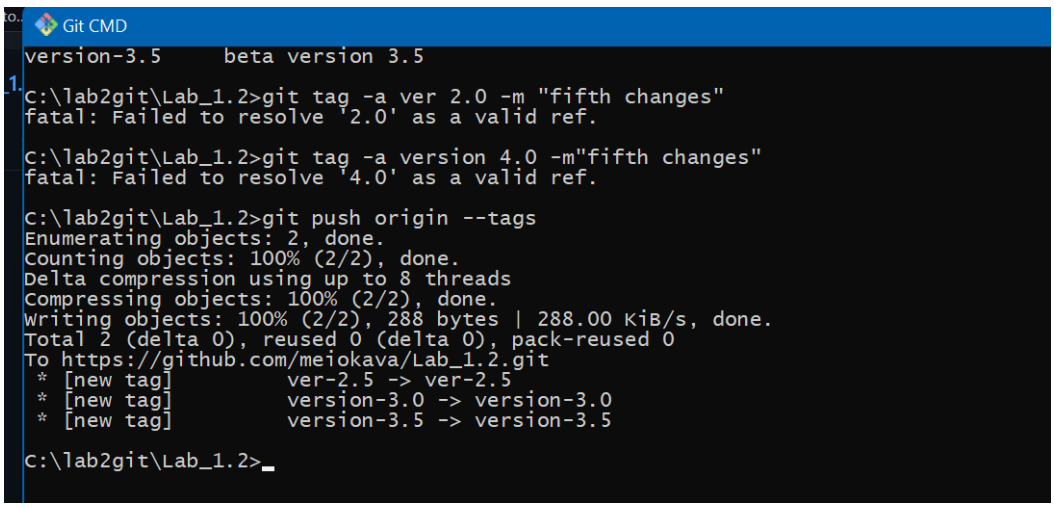

Рисунок 16 – Отправка тегов на удаленный сервер

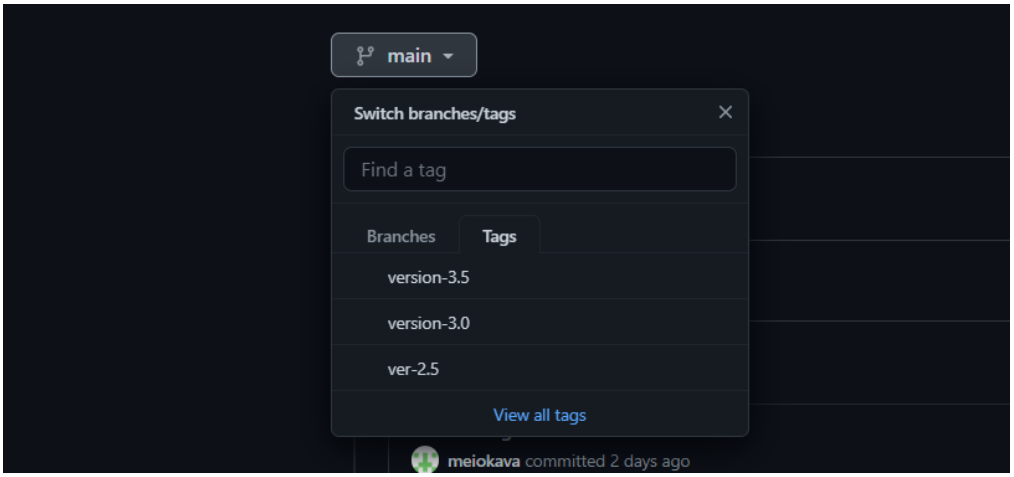

Рисунок 17 – История тегов на удаленном сервере

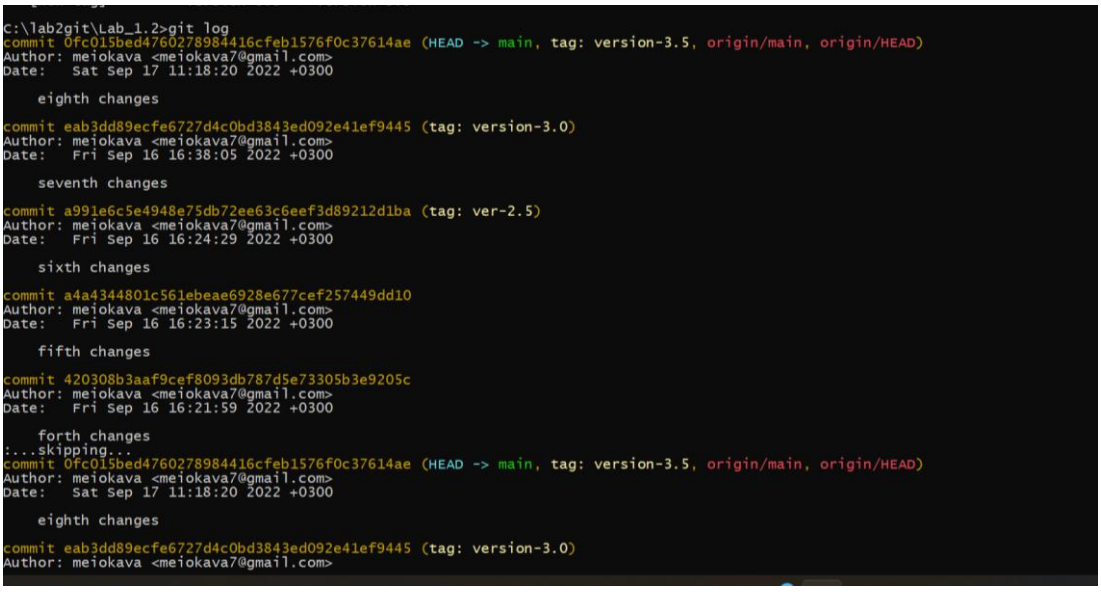

Рисунок 18 – История коммитов при помощи команды git log

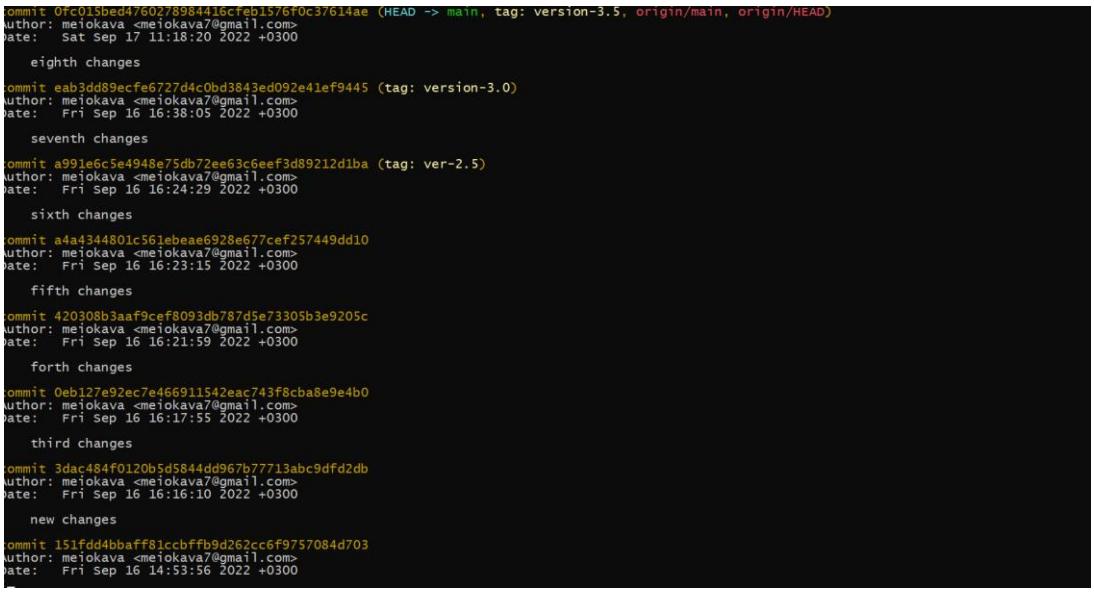

Рисунок 19 – Продолжение истории коммитов

Рисунок 20 – Просмотр коммитов командой git graph

5. Просмотрел содержимое коммитов командой git show HEAD, git show HEAD~1, git show a991e6c:

```
C:\lab2git\Lab_1.2>git show head
commit 0fc015bed4760278984416cfeb1576f0c37614ae (HEAD -> main, tag: ver
sion-3.5, origin/main, origin/HEAD)
Author: meiokava <meiokava7@gmail.com>
Date:   Sat Sep 17 11:18:20 2022 +0300

    eighth changes

diff --git a/appli/forGit/forGit.cpp b/appli/forGit/forGit.cpp
index 32fd72b..84216cf 100644
--- a/appli/forGit/forGit.cpp
+++ b/appli/forGit/forGit.cpp
@@ -13,4 +13,5 @@ int main()
        cout << "it is a result";
        cout << "it is a result";
        cout << "it is rainy day";
+       cout << "it is rainy day";
 }
\ No newline at end of file

C:\lab2git\Lab_1.2>_
```

Рисунок 21 – Просмотр содержимого последнего коммита

```
C:\lab2git\Lab_1.2>git show head~1
commit eab3dd89ecfe6727d4c0bd3843ed092e41ef9445 (tag: version-3.0)
Author: meiokava <meiokava7@gmail.com>
Date:   Fri Sep 16 16:38:05 2022 +0300

    seventh changes

diff --git a/appli/forGit/forGit.cpp b/appli/forGit/forGit.cpp
index dcb8fa5..32fd72b 100644
--- a/appli/forGit/forGit.cpp
+++ b/appli/forGit/forGit.cpp
@@ -12,4 +12,5 @@ int main()
        cout << 21 + 3 << endl;
        cout << "it is a result";
        cout << "it is a result";
+       cout << "it is rainy day";
 }
\ No newline at end of file

C:\lab2git\Lab_1.2>_
```

Рисунок 22 – Просмотр предпоследнего коммита

```
C:\lab2git\Lab_1.2>git show a991e6c
commit a991e6c5e4948e75db72ee63c6eef3d89212d1ba (tag: ver-2.5)
Author: meiokava <meiokava7@gmail.com>
Date:   Fri Sep 16 16:24:29 2022 +0300

    sixth changes

diff --git a/appli/forGit/forGit.cpp b/appli/forGit/forGit.cpp
index ca6b25d..dcb8fa5 100644
--- a/appli/forGit/forGit.cpp
+++ b/appli/forGit/forGit.cpp
@@ -8,7 +8,8 @@ int main()
        cout << "result is" << endl;
        cout << a + b << endl;
        cout << "it is a result";
-       cout << 10 + 1<<endl;
-       cout << 21 + 3<<endl;
+       cout << 10 + 1 << endl;
+       cout << 21 + 3 << endl;
+       cout << "it is a result";
        cout << "it is a result";
 }
\ No newline at end of file

C:\lab2git\Lab_1.2>
```

Рисунок 23 – Просмотр коммита с указанным хэшем

6. Возможность отката к заданной версии

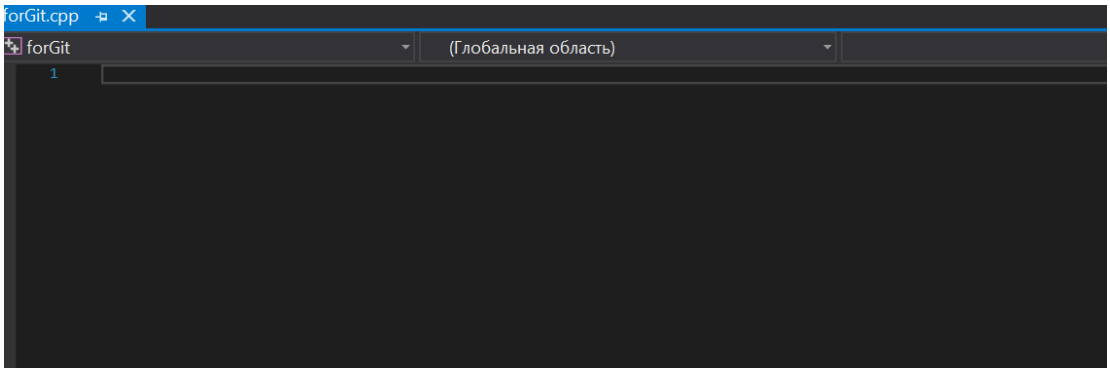

Рисунок 24 – Весь код был удален, а изменения сохранены

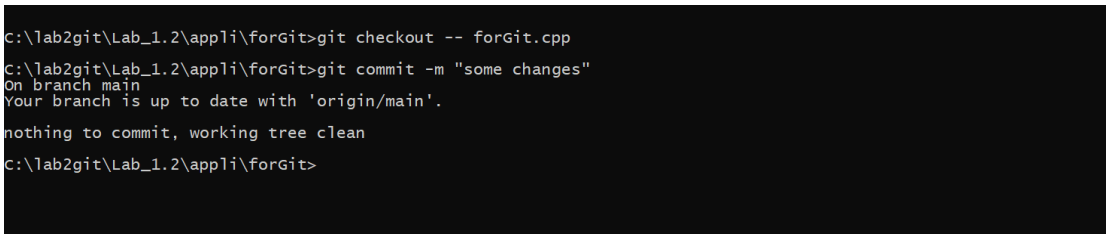

Рисунок 25 – Удаление всех несохраненных изменений командой checkout

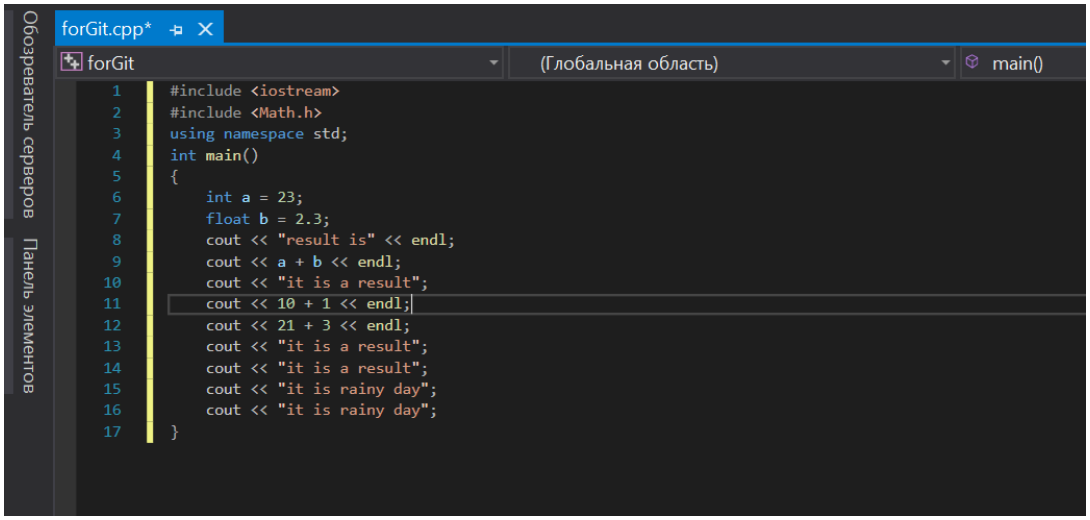

Рисунок 26 – Изменения программы после команды checkout

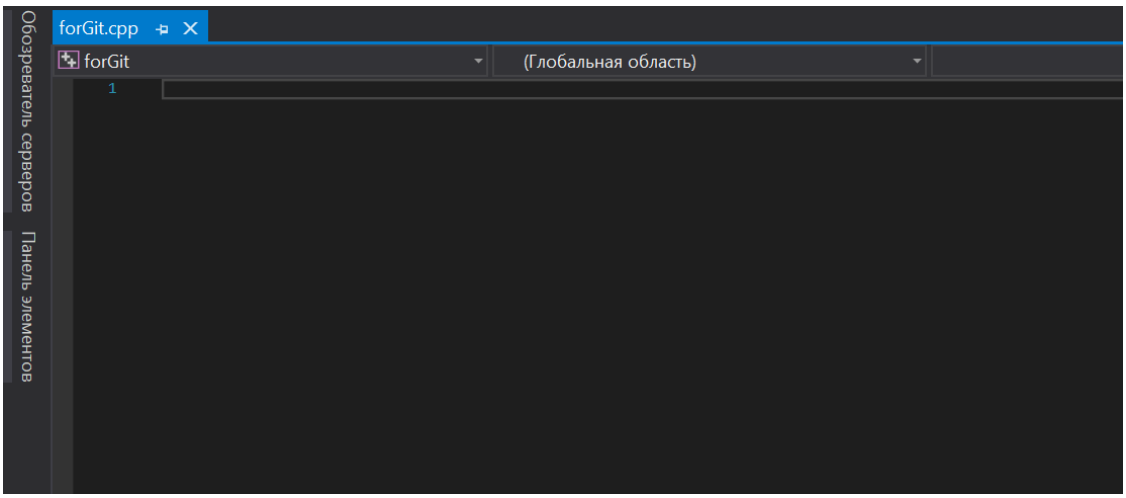

Рисунок 27 – Удаление кода в Visual studio

Рисунок 28 – Коммит изменений

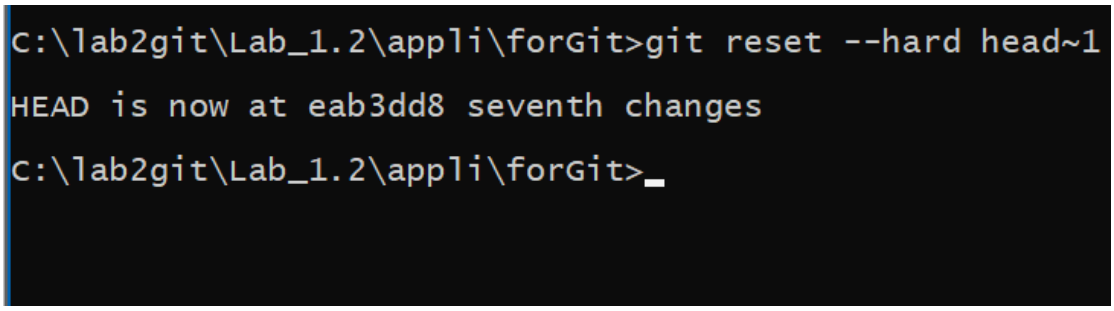

Рисунок 29 – Откат состояния хранилища к предыдущей версии

Вывод: в ходе работы были изучены базовые возможности системы контроля версий Git для работы с локальными репозиториями

Контрольные вопросы и ответы на них:

Вопросы для защиты работы.

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

Наиболее простой и в то же время мощный инструмент для этого — команда git log. По умолчанию, без аргументов, git log выводит список

коммитов созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми. Одна из опций, когда вы хотите увидеть сокращенную статистику для каждого коммита, вы можете использовать опцию –stat.

Вторая опция (одна 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 (пример команды git log –p -2).

Третья действительно полезная опция это --pretty. Она меняет формат вывода. Существует несколько встроенных вариантов отображения. Опция oneline выводит каждый коммит в одну строку, что может быть очень удоб ным если вы просматриваете большое количество коммитов. К тому же, опции short, full и fuller делают вывод приблизительно в том же формате, но с меньшим или большим количеством информации соответственно. Наиболее интересной опцией является format, которая позволяет указать формат для вывода информации. Особенно это может быть полезным, когда вы хотите сгенерировать вывод для автоматического анализа — так как вы указываете формат явно, он не будет изменен даже после обновления Git. Для опции git log --pretty=format существуют различного рода опции для изменения формата отображения.

2. **Как ограничить вывод при просмотре истории коммитов?**

Для ограничения может использоваться функция git log <n>, где n число записей. Также, существуют опции для ограничения вывода по времени, такие как --since и --until, они являются очень удобными. Например, следующая команда покажет список коммитов, сделанных за последние две недели: git log --since=2. weeks Это команда работает с большим количеством форматов — вы можете указать определенную дату вида 2008-01-15 или же относительную дату, например 2 years 1 day 3 minutes ago.

Также вы можете фильтровать список коммитов по заданным

параметрам. Опция --author дает возможность фильтровать по автору коммита, а опция --grep (показывает только коммиты, сообщение которых содержит указанную строку) искать по ключевым словам в сообщении коммита.

Функция –S показывает только коммиты, в которых изменение в коде повлекло за собой добавление или удаление указанной строки.

3. **Как внести изменения в уже сделанный коммит?**

Внести изменения можно с помощью команды git commit –amend Эта команда берёт индекс и применяет его к последнему коммиту. Если после последнего коммита не было никаких проиндексированных изменений (например, вы запустили приведённую команду сразу после предыдущего коммита), то состояние проекта будет абсолютно таким же и всё, что мы изменим, это комментарий к коммиту.

Для того, чтобы внести необходимые изменения - нам нужно проиндексировать их и выполнить комманду git commit --amend. git commit -m 'initial commit' git add forgotten_file git commit --amend

Эффект от выполнения этой команды такой, как будто мы не выполнили предыдущий коммит, а еще раз выполнили команду git add и выполнили коммит.

4**. Как отменить индексацию файла в Git?**

Например, вы изменили два файла и хотите добавить их в разные коммиты, но случайно выполнили команду git add * и добавили в индекс оба. Как исключить из индекса один из них? Команда git status напомнит вам: прямо под текстом «Changes to be committed» говорится: используйте git reset HEAD <file> для исключения из индекса.

5. Как отменить изменения в файле? С помощью команды git checkout -- <file>.

6. Что такое удаленный репозиторий Git? Удалённый репозиторий — это своего рода наше облако, в которое мы сохраняем те или иные изменения в нашей программе/коде/файлах.

7. Как выполнить просмотр удаленных репозиториев данного

локального репозитория?

Для того, чтобы просмотреть список настроенных удалённых репозиториев, необходимо запустить команду git remote. Также можно указать ключ -v, чтобы просмотреть адреса для чтения и записи, привязанные к репозиторию. Пример: git remote -v

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?**

Если необходимо получить изменения, которые есть у Пола, но нету у вас, вы можете выполнить команду git fetch <Название репозитория>. Важно отметить, что команда git fetch забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы. Если ветка настроена на отслеживание удалённой ветки, то вы можете использовать команду git pull чтобы автоматически получить изменения из удалённой ветки и слить их со своей текущей. Выполнение git pull, как правило, извлекает (fetch) данные с сервера, с которого вы изначально клонировали, и автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете. Чтобы отправить изменения на удалённый репозиторий необходимо отправить их в удалённый репозиторий. Команда для этого действия простая: git push <remote-name> <branch-name>.

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

Для просмотра удалённого репозитория, можно использовать команду git remote show <remote>.

11**. Каково назначение тэгов Git?**

Теги — это ссылки, указывающие на определённые версии

кода/написанной программы. Они удобно чтобы в случае чего вернутся к нужному моменту. Также при помощи тегов можно помечать важные моменты.

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

Просмотреть наличие тегов можно с помощью команды: git tag. А назначить (указать, добавить тег) можно с помощью команды git tag - a v1.4(версия изначальная) -m "Название". С помощью команды git show вы можете посмотреть данные тега вместе с коммитом: git show v1.4. Отправка тегов, по умолчанию, команда git push не отправляет теги на удалённые сервера. После создания теги нужно отправлять явно на удалённый сервер. Процесс аналогичен отправке веток — достаточно выполнить команду git push origin <tagname>. Для отправки всех тегов можно использовать команду git push origin tags. Для удаления тега в локальном репозитории достаточно выполнить команду git tag –d <tagname>. Например, удалить созданный ранее легковесный тег можно следующим образом: git tag -d v1.4-lw Для удаления тега из внешнего репозитория используется команда git push origin --delete <tagname>. Если вы хотите получить версии файлов, на которые указывает тег, то вы можете сделать git checkout для тега пример: git checkout -b version2 v2.0.0.

13**. Самостоятельно изучите назначение флага** --prune в командах git fetch и git push. Каково назначение этого флага? Git fetch --prune команда получения всех изменений с репозитория GitHub. В команде git push –prune удаляет удаленные ветки, у которых нет локального аналога.