

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 ЛАБОРАТОРНОЙ РАБОТЕ №2**

Выполнила:
Кузнецова Алена Валерьевна
1 курс, группа ИВТ-б-о-21-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем», очная
форма обучения

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

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

Выполнение работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и выбранный язык программирования.

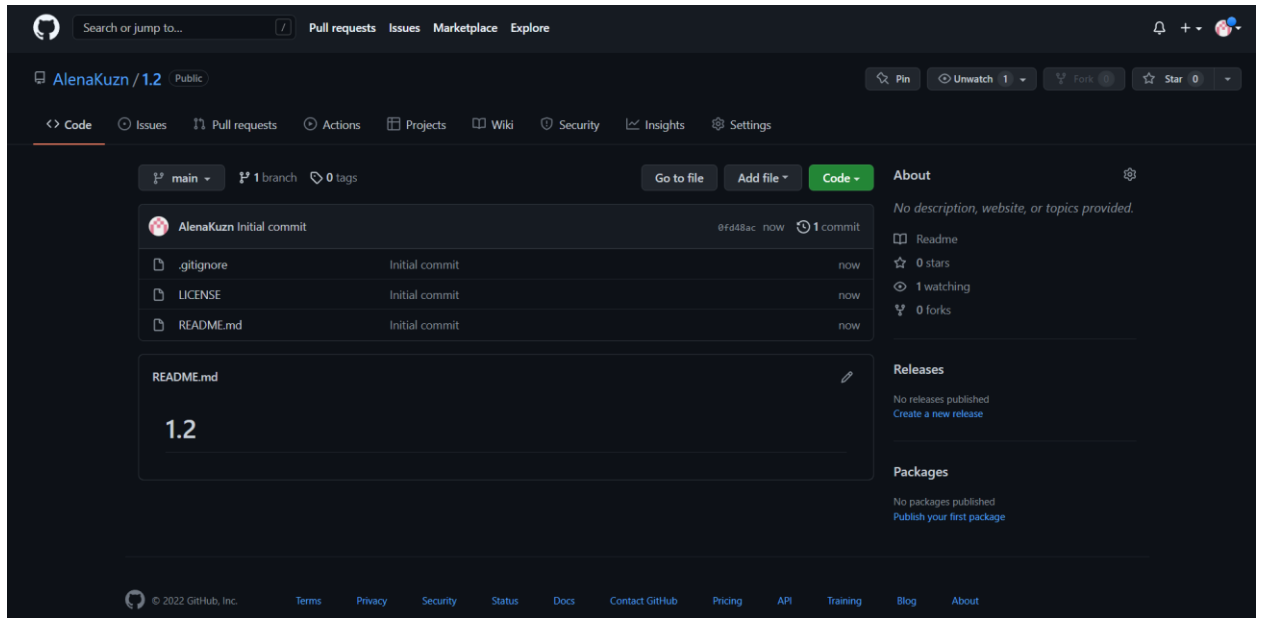


Рисунок 1 – Создание репозитория

2. Выполните клонирование созданного репозитория на рабочий компьютер.

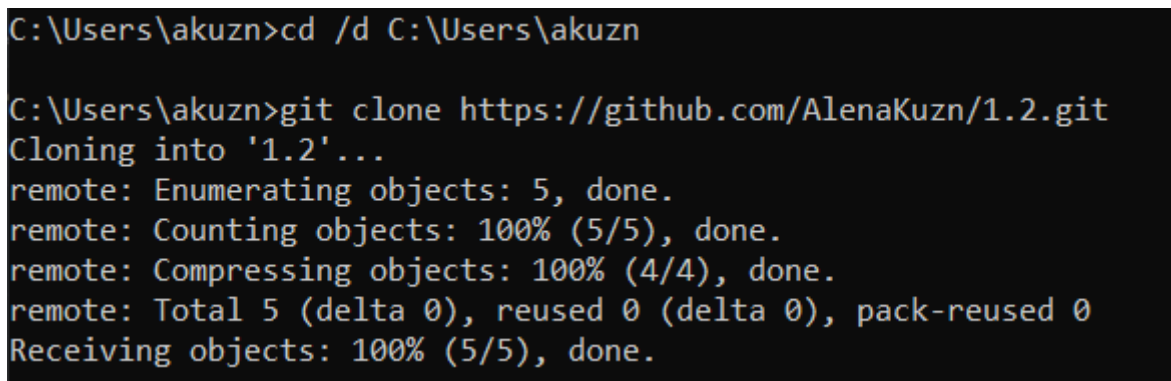
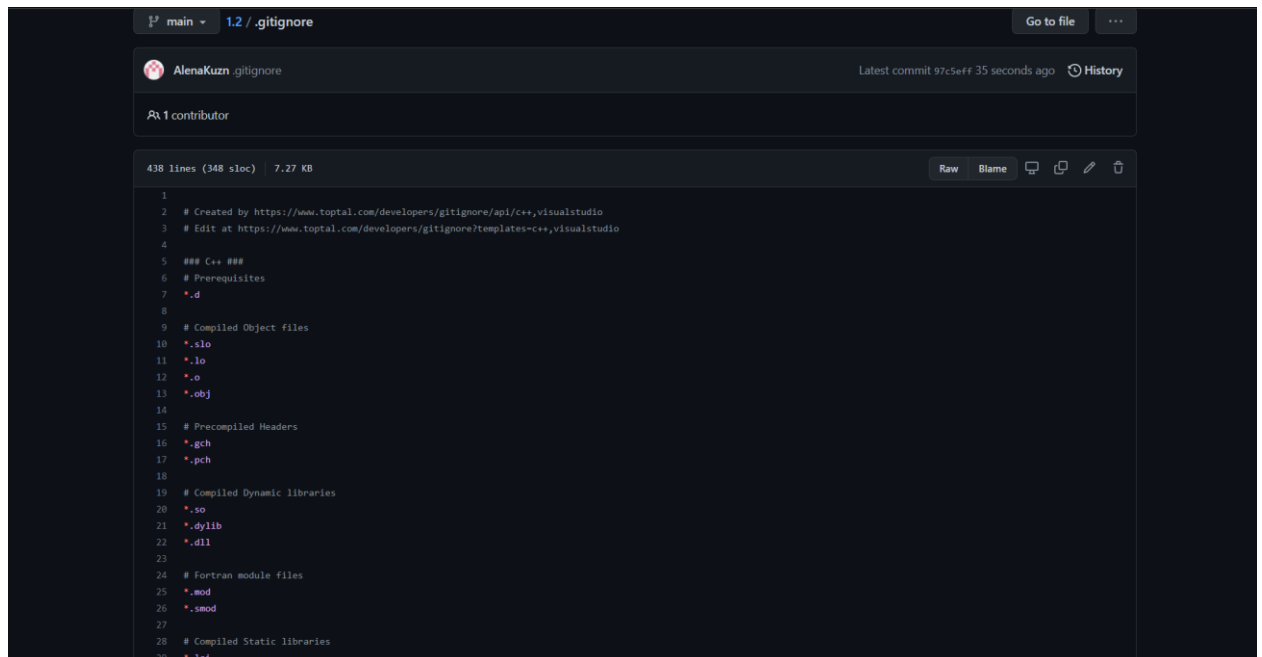


Рисунок 2 – Клонирование репозитория

3. Дополните файл .gitignore необходимыми правилами для выбранного языка программирования и интегрированной среды разработки.



The screenshot shows a GitHub repository for 'AlenaKuzn' with the file '.gitignore' selected. The file content is as follows:

```
1
2 # Created by https://www.toptal.com/developers/gitignore/api/c++,visualstudio
3 # Edit at https://www.toptal.com/developers/gitignore/templates-c++,visualstudio
4
5 ### C++ ###
6 # Prerequisites
7 *.d
8
9 # Compiled Object files
10 *.slo
11 *.lo
12 *.o
13 *.obj
14
15 # Precompiled Headers
16 *.gch
17 *.pch
18
19 # Compiled Dynamic libraries
20 *.so
21 *.dylib
22 *.dll
23
24 # Fortran module files
25 *.mod
26 *.smod
27
28 # Compiled Static libraries
29 *.lal
```

Рисунок 3 – Изменения в файле .gitignore

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

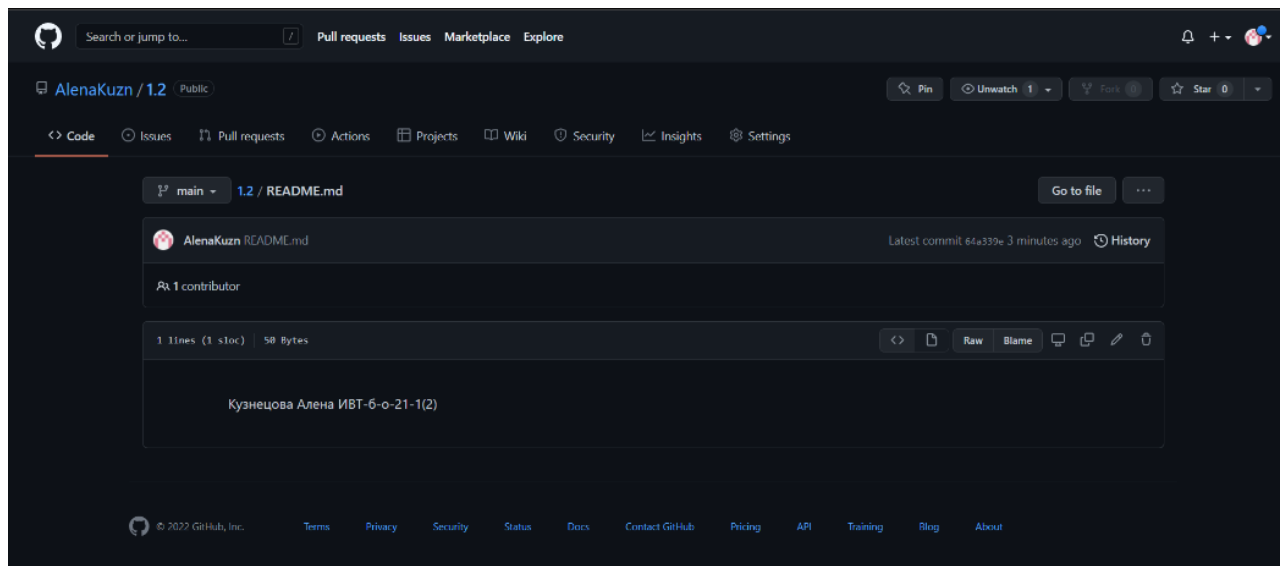


Рисунок 4 – Изменения в файле README.md

5. Напишите небольшую программу на выбранном Вами языке программирования. Фиксируйте изменения при написании программы в локальном репозитории. Должно быть сделано не менее 7 коммитов.

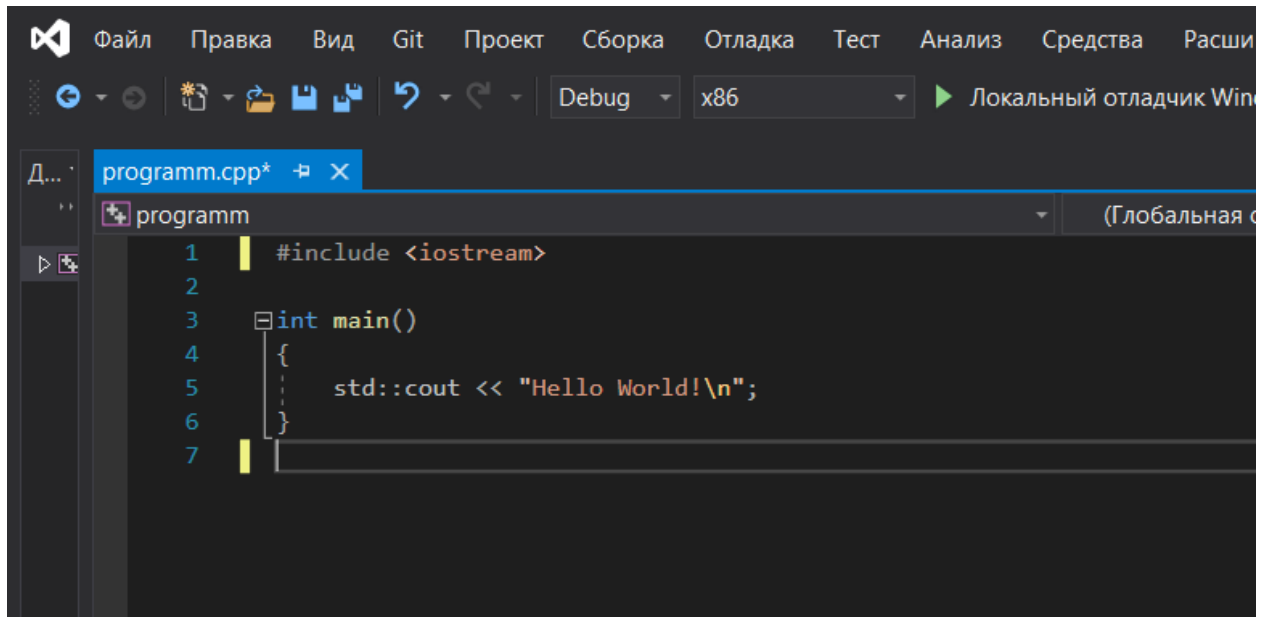


Рисунок 5 – Создание программы на языке C++

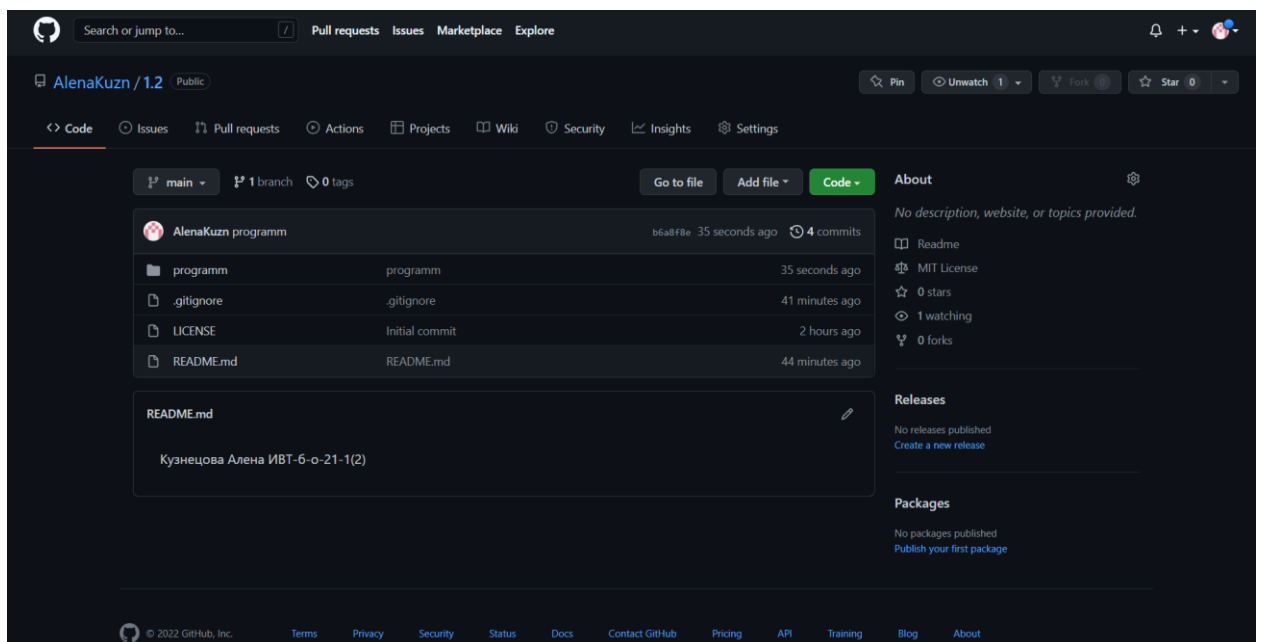


Рисунок 6 – Добавление файла с программой на удаленный репозиторий

```

C:\Users\akuzn\1.2>git commit -am"5"
[main 7b26a72] 5
1 file changed, 1 insertion(+)

C:\Users\akuzn\1.2>git commit -am"6"
[main 555ad85] 6
1 file changed, 1 insertion(+)

C:\Users\akuzn\1.2>git commit -am"7"
[main 48c4311] 7
1 file changed, 1 insertion(+)

C:\Users\akuzn\1.2>git push
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 16 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 1016 bytes | 1016.00 KiB/s, done.
Total 12 (delta 8), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (8/8), completed with 2 local objects.
To https://github.com/AlenaKuzn/1.2.git
8cdbc45..48c4311 main -> main

```

Рисунок 7 – Создание коммитов

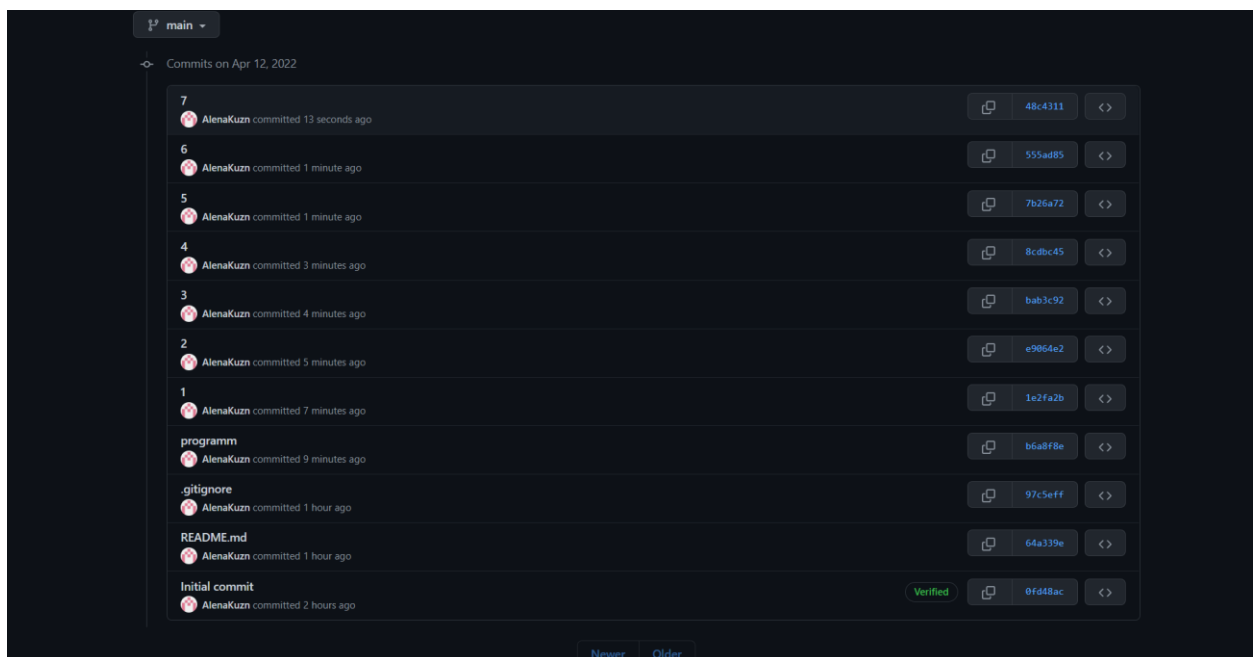


Рисунок 8 – История коммитов на удаленном сервере

6. Просмотреть историю (журнал) хранилища командой `git log`. Например, с помощью команды `git log --graph --pretty=oneline --abbrev-commit`.

```
C:\Users\akuzn\1.2>git log --graph --pretty=oneline --abbrev-commit
* 48c4311 (HEAD -> main, origin/main, origin/HEAD) 7
* 555ad85 6
* 7b26a72 5
* 8cdb45 4
* bab3c92 3
* e9064e2 2
* 1e2fa2b 1
* b6a8f8e programm
* 97c5eff .gitignore
* 64a339e README.md
* 0fd48ac Initial commit
```

Рисунок 9 – Просмотр журнала хранилища коммитов

7. Просмотреть содержимое коммитов командой `git show <ref>`, где `<ref>`:

- HEAD: последний коммит;
- HEAD~1: предпоследний коммит (и т. д.);
- b34a0e: коммит с указанным хэшем.

```
C:\Users\akuzn\1.2>git show HEAD
commit 48c4311be6c1cbd44990116e0dca9c25e98beb30 (HEAD -> main, origin/main, origin/HEAD)
Author: AlenaKuzn <99790256+AlenaKuzn@users.noreply.github.com>
Date: Tue Apr 12 19:26:15 2022 +0300

    7

diff --git a/programm/programm.cpp b/programm/programm.cpp
index d3591a1..802fc91 100644
--- a/programm/programm.cpp
+++ b/programm/programm.cpp
@@ -6,4 +6,5 @@ int main()
     int a = 5;
     int b = 54;
     int c = a;
+    float l = 6.7;
 }
```

Рисунок 10 – Содержание последнего коммита

```

C:\Users\akuzn\1.2>git show HEAD~1
commit 555ad85389abe80c961c7979ac41d1f79e763bfd
Author: AlenaKuzn <99790256+AlenaKuzn@users.noreply.github.com>
Date: Tue Apr 12 19:25:38 2022 +0300

    6

diff --git a/programm/programm.cpp b/programm/programm.cpp
index 3e10ec5..d3591a1 100644
--- a/programm/programm.cpp
+++ b/programm/programm.cpp
@@ -5,4 +5,5 @@ int main()
     std::cout << "Hello";
     int a = 5;
     int b = 54;
+    int c = a;
 }

```

Рисунок 11 – Содержание предпоследнего коммита

```

C:\Users\akuzn\1.2>git show 8cdbc45850f2bfc1bb2f8997e14f5ae5e99dd992
commit 8cdbc45850f2bfc1bb2f8997e14f5ae5e99dd992
Author: AlenaKuzn <99790256+AlenaKuzn@users.noreply.github.com>
Date: Tue Apr 12 19:23:54 2022 +0300

    4

diff --git a/programm/programm.cpp b/programm/programm.cpp
index aedab2e..4d03b6d 100644
--- a/programm/programm.cpp
+++ b/programm/programm.cpp
@@ -2,5 +2,6 @@

int main()
{
-    std::cout << "Hello, Alena";
+    std::cout << "Hello";
+    int a = 5;
}

C:\Users\akuzn\1.2>

```

Рисунок 12 – Содержание выбранного коммита

8. Освойте возможность отката к заданной версии.

8.1. Удалите весь код из одного из файлов программы репозитория, например `main.cpp`, и сохраните этот файл.

8.2. Удалите все несохраненные изменения в файле командой: `git checkout --<имя_файла>`, например `git checkout -- main.cpp`.

8.3. Повторите пункт 10.1 и сделайте коммит.

8.4. Откатить состояние хранилища к предыдущей версии командой: `git reset --hard HEAD~1`.

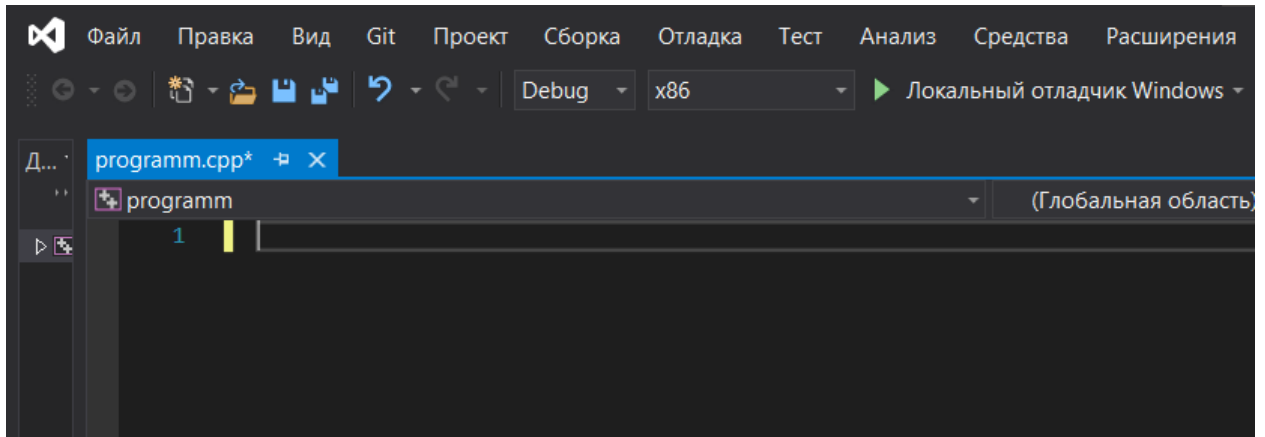


Рисунок 13 – Удаление программы

```
C:\Users\akuzn\1.2\programm>git checkout -- programm.cpp  
C:\Users\akuzn\1.2\programm>
```

Рисунок 14 – Изменение файла с помощью checkout

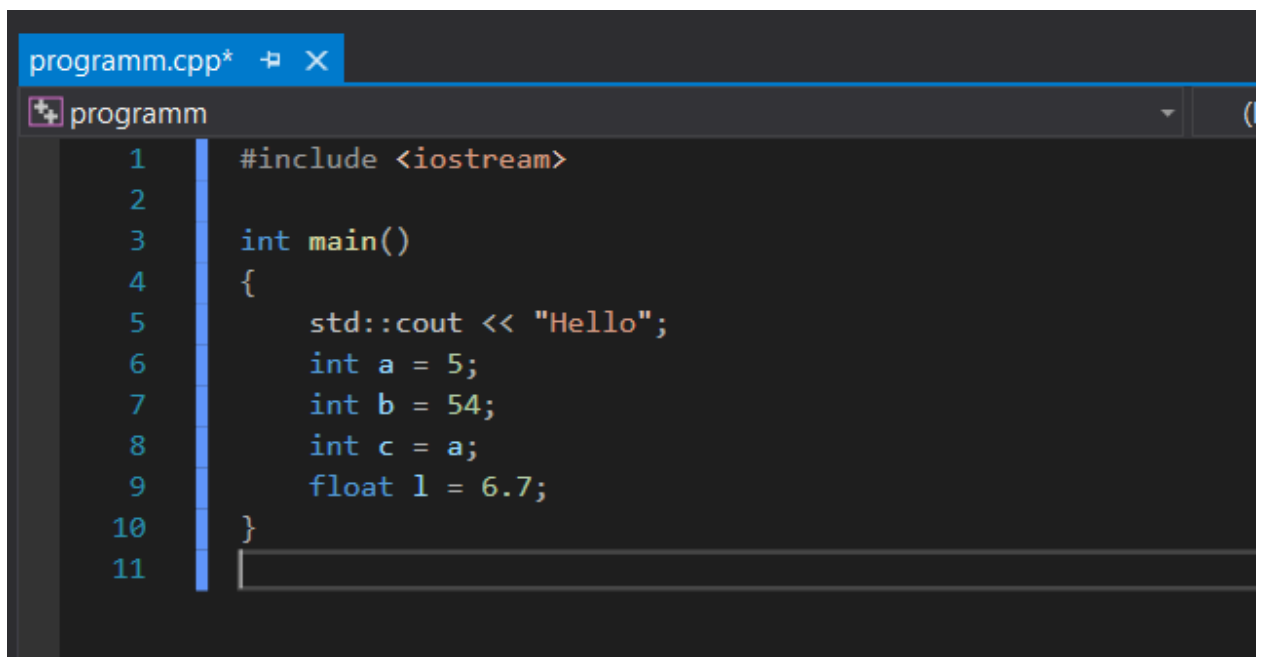


Рисунок 15 – Изменения после checkout

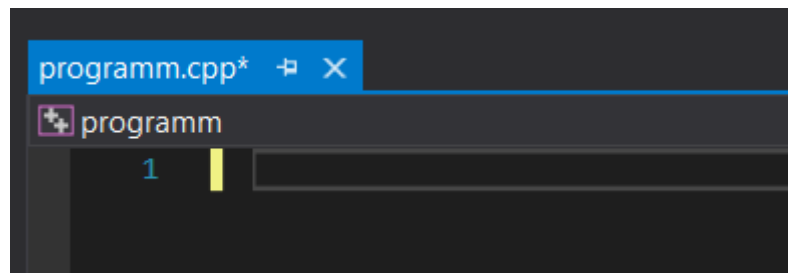


Рисунок 16 – Удаление кода

```
C:\Users\akuzn\1.2>git commit -am"Delet"
[main d2c6da1] Delet
1 file changed, 1 insertion(+), 10 deletions(-)
```

Рисунок 17 – Коммит изменений

Вывод: команда `git checkout --<имя_файла>` удаляет изменения файла в репозитории.

Вывод: мы исследовали базовые возможности системы контроля версий Git для работы с локальными репозиториями.

Контрольные вопросы:

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 ввыводе 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

удобным 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 команду `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` напомним 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 репозитория данного локального репозитория?

Для того, чтобы просмотреть список настроенных удалённых репозитория, 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 немодифицирует 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 удобны чтобы в случае чего вернуться 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.