

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ
УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Исследование возможностей Git для работы с локальными
репозиториями»**

**Отчет по лабораторной работе № 1.2
по дисциплине «Основы кроссплатформенного
программирования»**

Выполнил студент группы ИВТ-б-о-21-1

Сумин Никита Сергеевич.

«25 » мая 2022г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

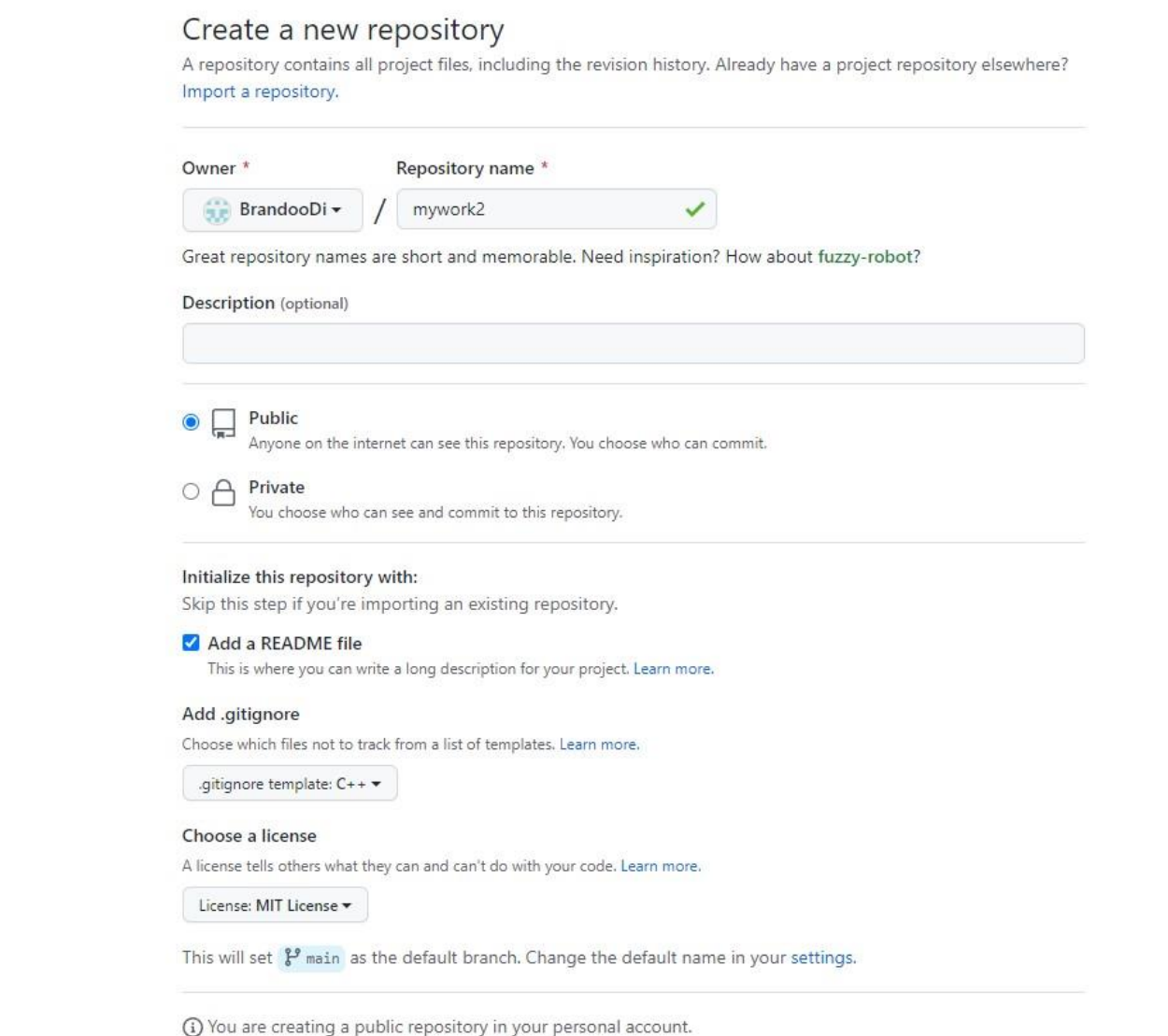
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2021

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

Порядок выполнения работы:


1. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования C++.



Create a new repository


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


Owner * Repository name *

 BrandooDi ▼ / mywork2 ✓

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

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:
Skip this step if you're importing an existing repository.

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

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

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

This will set  **main** as the default branch. Change the default name in your [settings](#).


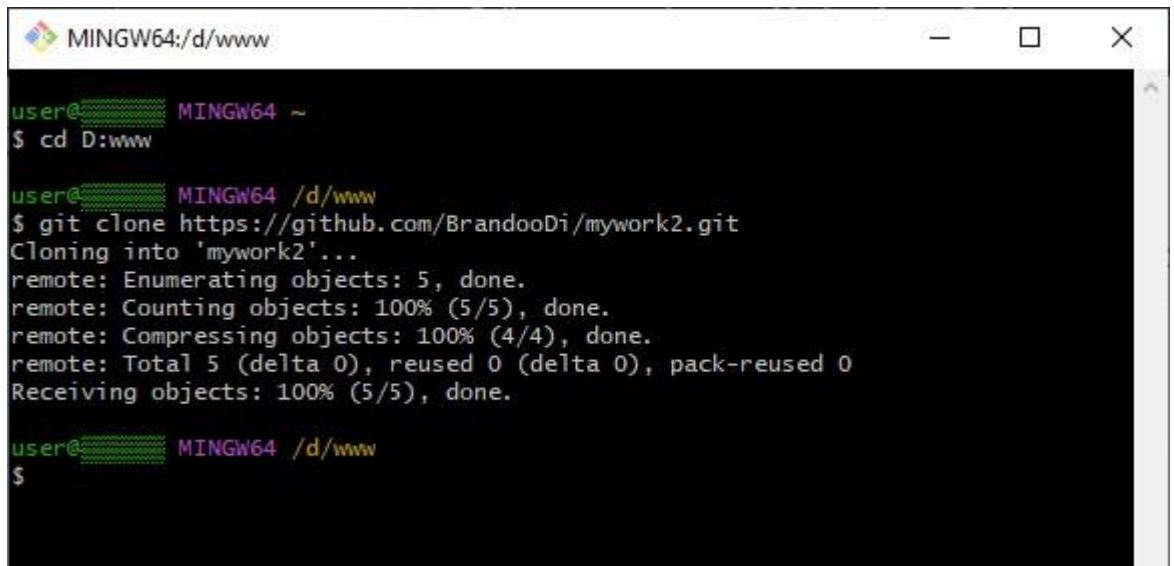
 You are creating a public repository in your personal account.

Рисунок 1 - Создание репозитория

2. Выполните клонирование созданного репозитория на рабочий компьютер.



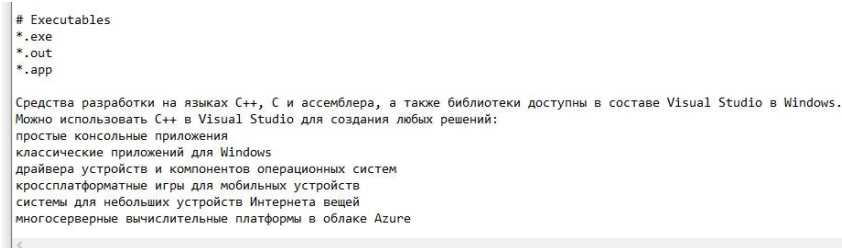
```
MINGW64:/d/www
user@MINGW64 ~
$ cd D:www

user@MINGW64 /d/www
$ git clone https://github.com/BrandooDi/mywork2.git
Cloning into 'mywork2'...
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
Receiving objects: 100% (5/5), done.

user@MINGW64 /d/www
$
```

Рисунок 2 - Клонирование репозитория

3. Дополнил файл .gitignore

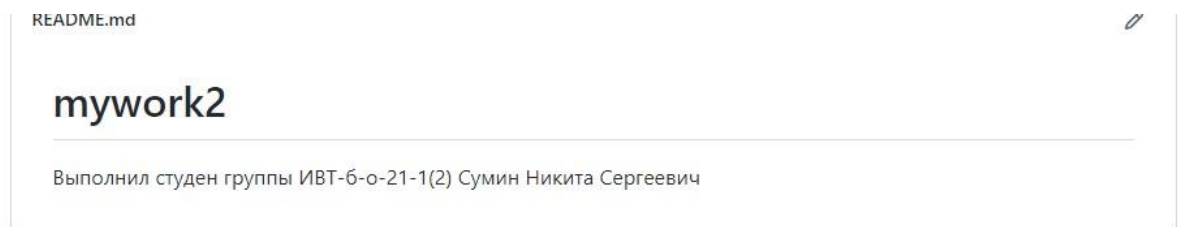


```
# Executables
*.exe
*.out
*.app

Средства разработки на языках C++, C и ассемблера, а также библиотеки доступны в составе Visual Studio в Windows.
Можно использовать C++ в Visual Studio для создания любых решений:
простые консольные приложения
классические приложения для Windows
драйвера устройств и компонентов операционных систем
кроссплатформатные игры для мобильных устройств
системы для небольших устройств Интернета вещей
многосерверные вычислительные платформы в облаке Azure
```

Рисунок 3 - Файл .gitignore

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



```
README.md

mywork2

Выполнил студент группы ИВТ-б-о-21-1(2) Сумин Никита Сергеевич
```

Рисунок 4 - Файл README.md

5. Напишите программу. Фиксировал изменения при написании программы в локальном репозитории. Сделано 7 коммитов, отмеченных 3 тэгами.

```

2 using namespace std;
3 int main()
4 {
5     setlocale(LC_ALL, "Russian");
6     const int n = 10;
7     int *A;
8     int *G;
9     int *F;
10    A = (int*)malloc(n * sizeof(int));
11    G = (int*)malloc(n * sizeof(int));
12    F = (int*)malloc(n * sizeof(int));
13    int i, suma = 0, sumg = 0, sumf = 0;
14    cout << "Введите оценки по алгебре:\n";
15    for (i = 0; i < n; i++)
16        cin >> A[i];
17    for (i = 0; i < n; i++) suma += A[i];
18    cout << "Средняя оценка по алгебре:" << suma / n;
19    cout << "Введите оценки по геометрии:\n";
20    for (i = 0; i < n; i++)
21        cin >> G[i];
22    for (i = 0; i < n; i++) sumg += G[i];
23    cout << "Средняя оценка по геометрии:" << sumg / n;
24    cout << "Введите оценки по физике:\n";
25    for (i = 0; i < n; i++)
26        cin >> F[i];
27    for (i = 0; i < n; i++) sumf += F[i];
28    cout << "Средняя оценка по физике:" << sumf / n;
29    if (suma / n > sumg / n)
30    {
31        if (suma / n > sumf / n)
32            cout << "Наилучший предмет: Алгебра";
33        else
34            cout << "Наилучший предмет: Физика";
35    }
36    else
37    {
38        if (suma / n < sumg / n)
39        {
40            if (sumg / n < sumf / n)

```

Рисунок 5 - Конечный вид программы

| | | |
|-------------------------|----------------------------------|---------------------|
| Commits on May 25, 2022 | | |
| 7 | BrandooDI committed 4 hours ago | 112bb11 <> |
| 6 | BrandooDI committed 4 hours ago | f264ad3 <> |
| 5 | BrandooDI committed 4 hours ago | c48d2dc <> |
| 4 | BrandooDI committed 4 hours ago | 2652688 <> |
| 3 | BrandooDI committed 4 hours ago | 07121b4 <> |
| 2 | BrandooDI committed 4 hours ago | 12de6fb <> |
| 1 | BrandooDI committed 4 hours ago | 920ba21 <> |
| 43434 | BrandooDI committed 5 hours ago | affa527 <> |
| Commits on May 24, 2022 | | |
| 4124 | BrandooDI committed 20 hours ago | dc9f8db <> |
| Commits on May 14, 2022 | | |
| 5345 | BrandooDI committed 11 days ago | c9dc5ab <> |
| Initial commit | BrandooDI committed 11 days ago | Verified 7858832 <> |

Рисунок 6 – Коммиты

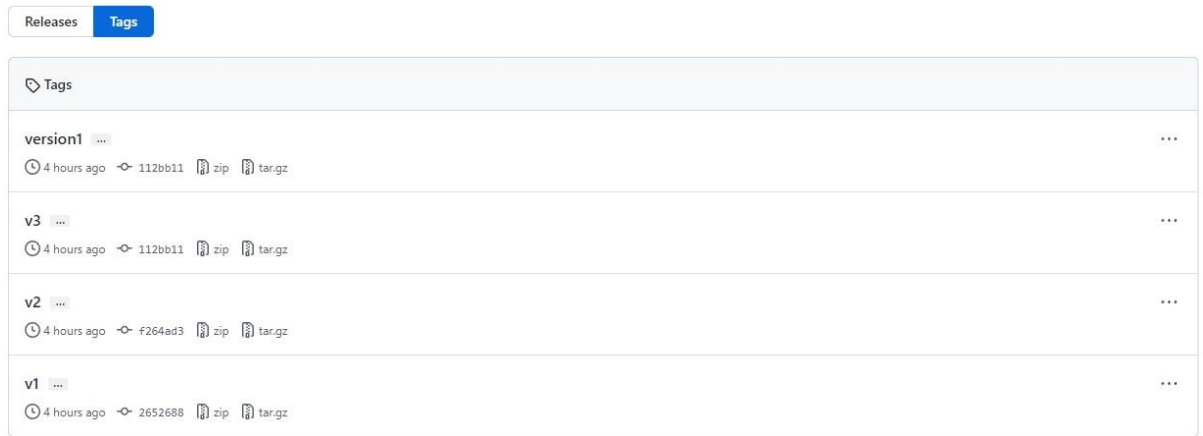


Рисунок 7 - Теги

6. Просмотрел историю (журнал) хранилища командой `git log -graph --pretty=oneline --abbrev-commit`.

```
MINGW64:/d/www/mywork2
* [new tag]          v2 -> v2
* [new tag]          v3 -> v3
* [new tag]          version1 -> version1

user@MINGW64 /d/www/mywork2 (main)
$ git log -graph --pretty=oneline --abbrev-commit
fatal: unrecognized argument: -graph

user@MINGW64 /d/www/mywork2 (main)
$ git log --graph --pretty=oneline --abbrev-commit
* 112bb11 (HEAD -> main, tag: version1, tag: v3, origin/main, origin/HEAD) 7
* f264ad3 (tag: v2) 6
* c48d2dc 5
* 2652688 (tag: v1) 4
* 07121b4 3
* 12de6fb 2
* 920ba21 1
* affa527 43434
* dc9f0db 4124
* c9dc5ab 5345
* 7858832 Initial commit

user@MINGW64 /d/www/mywork2 (main)
$
```

Рисунок 8 - История коммитов

7. Просмотрел содержимое коммитов командой `z, git show HEAD~`, `git show 112bb11dc7d5e4885ead4a6c35853b363cd550c9`:

```

user@MINGW64 ~
$ cd D:\www\mywork2

user@MINGW64 /d/www/mywork2 (main)
$ git show HEAD
commit 112bb11dc7d5e4885ead4a6c35853b363cd550c9 (HEAD -> main, tag: version1, tag: v3, origin/main, origin/HEAD)
Author: BrandooDi <nikita1231w@gmail.com>
Date:   Wed May 25 15:50:12 2022 +0300

    7

diff --git a/2/main.cpp b/2/main.cpp
index bd57ef8..a66befd 100644
--- a/2/main.cpp
+++ b/2/main.cpp
@@ -38,3 +38,10 @@ cout << "

```

Рисунок 9 - git show HEAD

```

user@MINGW64 ~
$ git show HEAD~
fatal: not a git repository (or any of the parent directories): .git

user@MINGW64 ~
$ cd D:\www\mywork2

user@MINGW64 /d/www/mywork2 (main)
$ git show HEAD~
commit f264ad36b0ba96d1b8c2ae71d6fb3310ad0716ce (tag: v2)
Author: BrandooDi <nikita1231w@gmail.com>
Date:   Wed May 25 15:49:39 2022 +0300

    6

diff --git a/2/main.cpp b/2/main.cpp
index 46a8bcd..bd57ef8 100644
--- a/2/main.cpp
+++ b/2/main.cpp
@@ -29,3 +29,12 @@ for (i = 0; i < n; i++) sumf += F[i];
     if (suma / n > sumg / n)
     {
         if (suma / n > sumf / n)

```

Рисунок 10 - git show HEAD~

```

MINGW64:/d/www/mywork2
user@MINGW64 /d/www/mywork2 (main)
$ git show ^M 6
fatal: ambiguous argument '6': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

user@MINGW64 /d/www/mywork2 (main)
$ git show 112bb11dc7d5e4885ead4a6c35853b363cd550c9
commit 112bb11dc7d5e4885ead4a6c35853b363cd550c9 (HEAD -> main, tag: version1, ta
g: v3, origin/main, origin/HEAD)
Author: BrandooDi <nikita1231w@gmail.com>
Date: Wed May 25 15:50:12 2022 +0300

7

diff --git a/2/main.cpp b/2/main.cpp
index bd57ef8..a66befd 100644
--- a/2/main.cpp
+++ b/2/main.cpp
@@ -38,3 +38,10 @@ cout << "
{
    if (sumg / n < sumf / n)
        cout << "<CD><E0><E8><EB><F3><F7><F8><E8><E9> <EF><F0><E5><E4><

```

Рисунок 11 - git show 112bb11dc7d5e4885ead4a6c35853b363cd550c9

8. Удалил весь код в файле `Untitled-1.cpp` и сохранил его, затем удалил все несохраненные изменения командой, после этого еще раз удалил весь код в файле и сделал коммит, после чего откатил состояние файла к предыдущей версии.

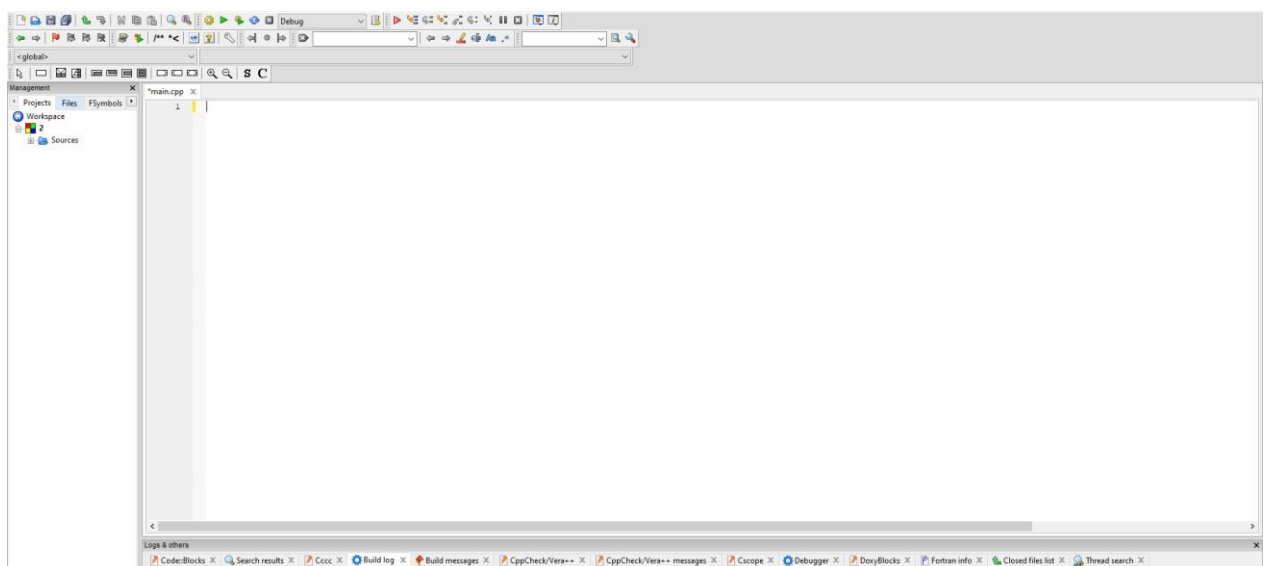


Рисунок 12 - Удаление кода из файла pr2.cpp


```
user@MINGW64 /d/www/mywork2 (main)
$ git checkout -- 2/main.cpp

user@MINGW64 /d/www/mywork2 (main)
$ |
```

Рисунок 13 - Удаление несохраненных изменений при помощи команды git checkout

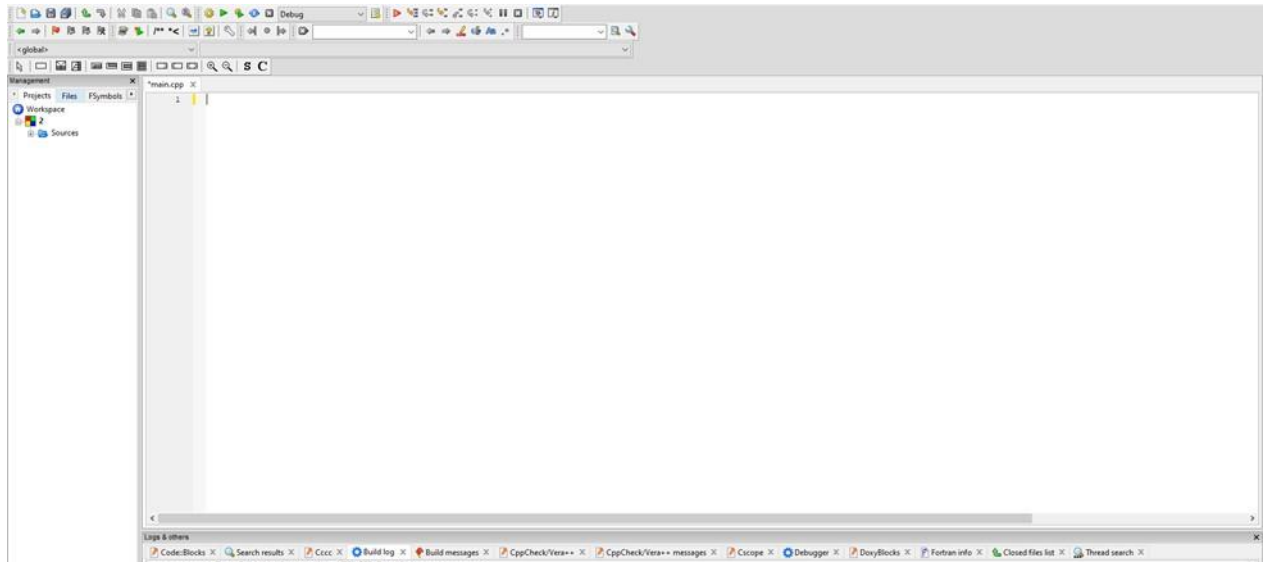


Рисунок 14 - Удаление кода

```
MINGW64:/d/www/mywork2

user@MINGW64 /d/www/mywork2 (main)
$ git commit -m "delete code"
[main 137c912] delete code
2 files changed, 4 insertions(+), 47 deletions(-)
create mode 100644 2/2.depend
rewrite 2/main.cpp (100%)

user@MINGW64 /d/www/mywork2 (main)
$ git reset --hard HEAD~1
fatal: ambiguous argument '--hard': unknown revision or path not in the working tree.
Use '--' to separate paths from revisions, like this:
'git <command> [<revision>...] -- [<file>...]'

user@MINGW64 /d/www/mywork2 (main)
$ git reset -hard HEAD~1
error: did you mean '--hard' (with two dashes)?

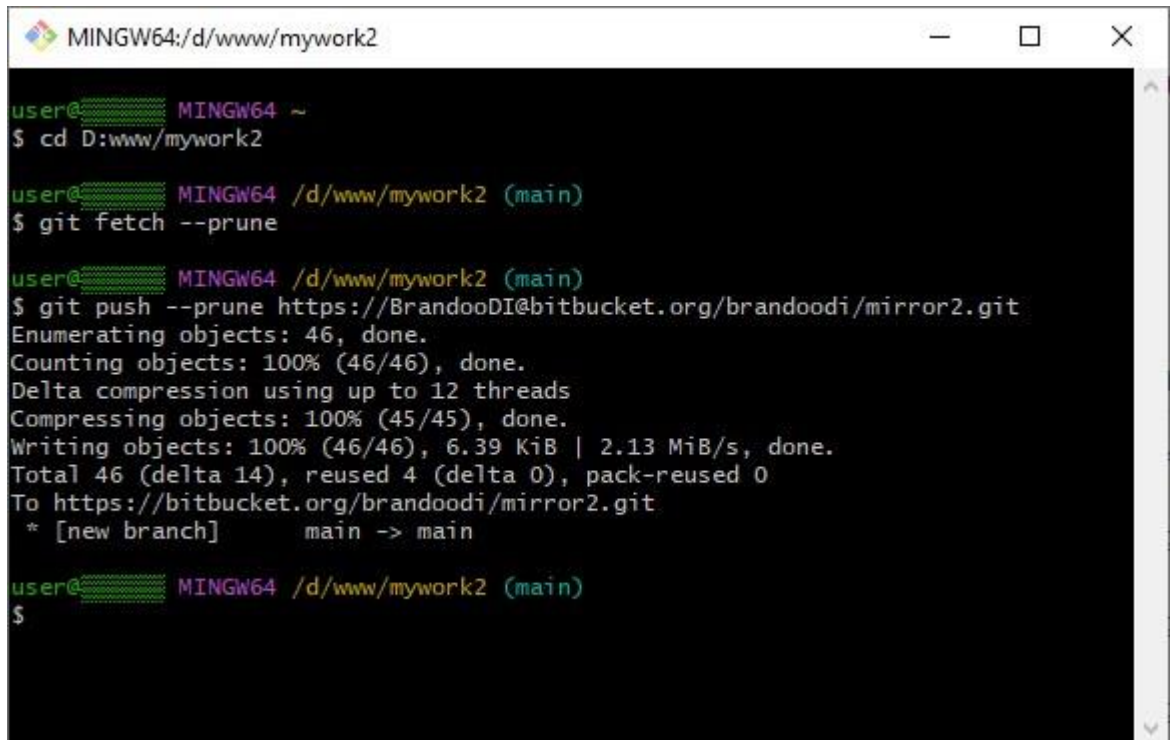
user@MINGW64 /d/www/mywork2 (main)
$ git reset --hard HEAD~1
HEAD is now at 112bb11 7

user@MINGW64 /d/www/mywork2 (main)
$
```

Рисунок 15 - Коммит и команда git reset --hard HEAD~1 команда git --checkout удаляет изменения произошедшие с файлом в

репозитории до коммита.

9. Создал зеркало репозитория на BitBucket.



```
MINGW64:/d/www/mywork2

user@MINGW64 ~
$ cd D:/www/mywork2

user@MINGW64 /d/www/mywork2 (main)
$ git fetch --prune

user@MINGW64 /d/www/mywork2 (main)
$ git push --prune https://Brandoodi@bitbucket.org/brandoodi/mirror2.git
Enumerating objects: 46, done.
Counting objects: 100% (46/46), done.
Delta compression using up to 12 threads
Compressing objects: 100% (45/45), done.
Writing objects: 100% (46/46), 6.39 KiB | 2.13 MiB/s, done.
Total 46 (delta 14), reused 4 (delta 0), pack-reused 0
To https://bitbucket.org/brandoodi/mirror2.git
 * [new branch]      main -> main

user@MINGW64 /d/www/mywork2 (main)
$
```

Рисунок 16 - Команды git fetch --prune и git push --prune

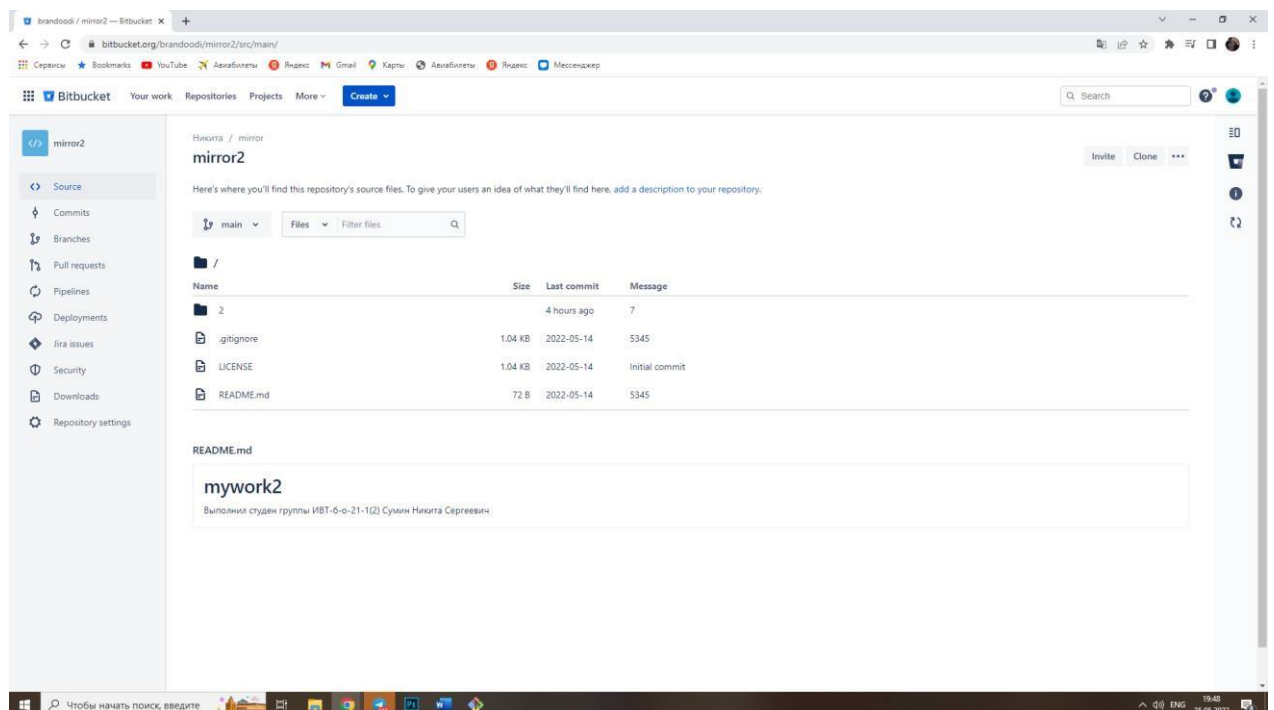


Рисунок 17 - Репозиторий на BitBucket Адрес

репозитория: <https://bitbucket.org/brandoodi/mirror2/src/main/>

Ответы на 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 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 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`. Эта 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`.

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` для исключения из индекса.

5. Как отменить изменения в файле?

С помощью команды `git checkout --` .

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 репозиториях данного локального репозитория?

Для того, чтобы просмотреть список настроенных удалённых репозиториях, 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` .

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` . 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` .

10. Как выполнить просмотр удаленного репозитория?

Для просмотра удалённого репозитория, можно использовать ко-манду `git remote show` .

11. Каково назначение тэгов Git?

Теги - это ссылки указывающие на определённые версии ко да/написанной программы. Они удобно 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` . Для отправки всех тегов можно использовать команду `git push origin tags`.

Для удаления тега в локальном репозитории достаточно выполнить команду `git tag -d` . Например, удалить созданный ранее легко весный тег

можно следующим образом: `git tag -d v1.4-lw` Для удаления тега из внешнего репозитория используется команда `git push origin --delete` .

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.

`Git push --prune` позволяет выполнить отправку содержимого локального репозитория в репозиторий GitLab или BitBucket.

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