

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

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЁТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №1.2**  
**дисциплины**  
**«Основы кроссплатформенного программирования»**

Выполнил:  
Костукайло Кирилл Николаевич  
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)

Проверил:  
Кафедры инфокоммуникаций, старший  
преподаватель  
Воронкин Р.А.

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2022 г.

## Тема: Исследование возможностей Git для работы с локальными репозиториями

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

Порядок выполнения работы:

1. Нужно создать новый репозиторий, в файле README.md указать информацию о обучающемся и скопировать репозиторий на рабочий стол.
2. Проработаем примеры в лабораторной работе.
3. Напишем небольшую программу на C++. Зафиксируем изменения при написании программы на локальном репозитории.

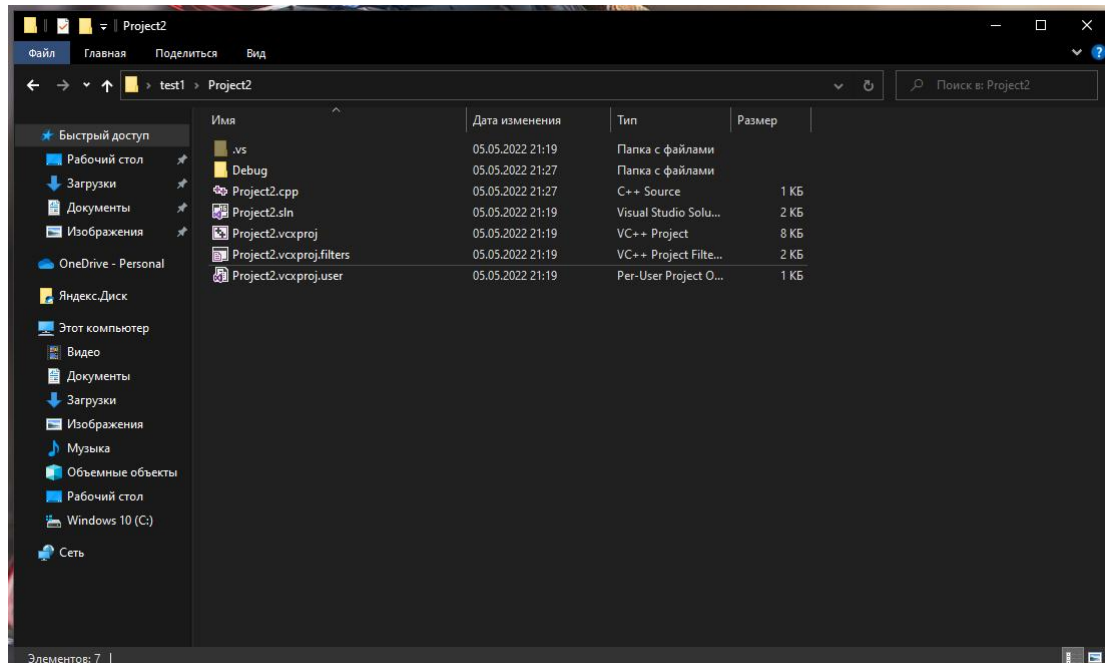


Рисунок 1. Программа на C++

```
MINGW64/c:/Users/User/Desktop/test1
$ git tag -a v0.1 -m "my version 0.1"

User@WIN-CD5F85VVD03 MINGW64 ~/Desktop/test1 (main)
$ git tag
v0.1

User@WIN-CD5F85VVD03 MINGW64 ~/Desktop/test1 (main)
$ git tag -a v0.3 -m "my version 0.3"

User@WIN-CD5F85VVD03 MINGW64 ~/Desktop/test1 (main)
$ git tag
v0.1
v0.3

User@WIN-CD5F85VVD03 MINGW64 ~/Desktop/test1 (main)
$ git tag -a v0.5 -m "my version 0.5"

User@WIN-CD5F85VVD03 MINGW64 ~/Desktop/test1 (main)
$ git tag
v0.1
v0.3
v0.5

User@WIN-CD5F85VVD03 MINGW64 ~/Desktop/test1 (main)
$
```

## Рисунок 2. Теги для программы

4. Просмотреть историю (журнал) хранилища командой `git log`. Например, с помощью команды `git log --graph --pretty=oneline --abbrev-commit`. Добавить скриншот консоли с выводом в отчет по лабораторной работе.

```
User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git log
commit f9d7c730ff9c7fade74d20ca862089d8577b3630 (HEAD -> main)
Author: MrGeleos <mr.geleos12@mail.ru>
Date: Thu May 5 21:31:21 2022 +0300

    Added a folder with the program

commit d9efeaaf17de90a04fede32bca451c446803366b6
Author: MrGeleos <mr.geleos12@mail.ru>
Date: Thu May 5 11:46:23 2022 +0300

    Added information about myself in the file README.md

commit 76c475ead6e1226104e5e64c5b99b7175d974efc (origin/main, origin/HEAD)
Author: Geleos Sapfirov <99472504+MrGeleos@users.noreply.github.com>
Date: Thu May 5 11:07:41 2022 +0300

    Initial commit

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git log --graph
* commit f9d7c730ff9c7fade74d20ca862089d8577b3630 (HEAD -> main)
  Author: MrGeleos <mr.geleos12@mail.ru>
  Date: Thu May 5 21:31:21 2022 +0300

    Added a folder with the program

* commit d9efeaaf17de90a04fede32bca451c446803366b6
  Author: MrGeleos <mr.geleos12@mail.ru>
  Date: Thu May 5 11:46:23 2022 +0300

    Added information about myself in the file README.md

* commit 76c475ead6e1226104e5e64c5b99b7175d974efc (origin/main, origin/HEAD)
  Author: Geleos Sapfirov <99472504+MrGeleos@users.noreply.github.com>
  Date: Thu May 5 11:07:41 2022 +0300

    Initial commit

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git log --pretty=oneline
fatal: invalid --pretty format: oneline

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git log --pretty=oneline
f9d7c730ff9c7fade74d20ca862089d8577b3630 (HEAD -> main) Added a folder with the program
d9efeaaf17de90a04fede32bca451c446803366b6 Added information about myself in the file README.md
76c475ead6e1226104e5e64c5b99b7175d974efc (origin/main, origin/HEAD) Initial commit

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git log --abbrev-commit
commit f9d7c73 (HEAD -> main)
Author: MrGeleos <mr.geleos12@mail.ru>
Date: Thu May 5 21:31:21 2022 +0300

    Added a folder with the program

commit d9efeaaf
Author: MrGeleos <mr.geleos12@mail.ru>
Date: Thu May 5 11:46:23 2022 +0300

    Added information about myself in the file README.md

commit 76c475e (origin/main, origin/HEAD)
Author: Geleos Sapfirov <99472504+MrGeleos@users.noreply.github.com>
Date: Thu May 5 11:07:41 2022 +0300

    Initial commit

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
```

## Рисунок 3. История хранилища

5. Просмотреть содержимое коммитов командой `git show <ref>`, где `<ref>` :  
HEAD : последний коммит;  
HEAD~1 : предпоследний коммит (и т. д.);  
b34a0e : коммит с указанным хэшем.  
Отобразите результаты работы этих команд в отчете.



6. Освойте возможность отката к заданной версии.

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

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

6.3. Повторите пункт 10.1 и сделайте коммит.

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

Сделайте выводы об изменении содержимого выбранного Вами файла программы после выполнения пунктов 6.1–6.4. Отрадите эти выводы в отчёте.

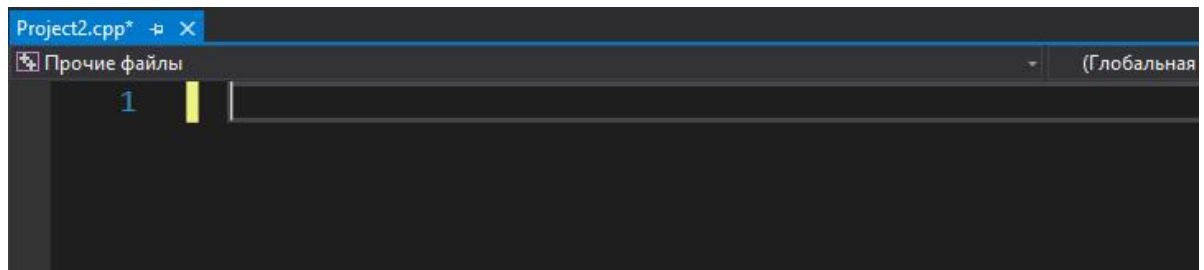


Рисунок 7.

```
User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git checkout -- Project2/Project2.cpp

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git add Project2/Project2.cpp

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git commit -m "Deleted all the code"
[main 84e1b3a] Deleted all the code
1 file changed, 1 insertion(+), 17 deletions(-)

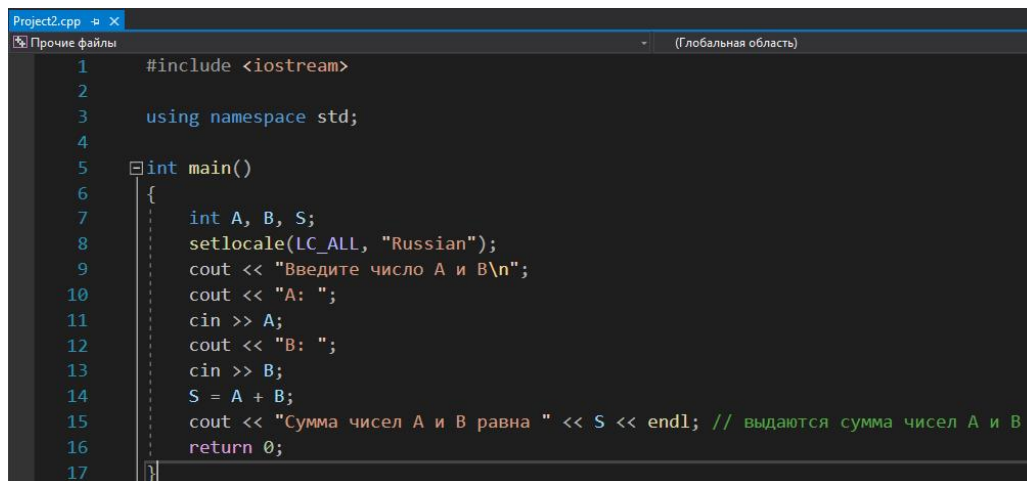
User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ |
```

Рисунок 8.

```
User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$ git reset --hard HEAD~1
HEAD is now at b686731 Add description

User@WIN-CD5F8SVVD03 MINGW64 ~/Desktop/test1 (main)
$
```

Рисунок 9.



```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      int A, B, S;
8      setlocale(LC_ALL, "Russian");
9      cout << "Введите число A и B\n";
10     cout << "A: ";
11     cin >> A;
12     cout << "B: ";
13     cin >> B;
14     S = A + B;
15     cout << "Сумма чисел A и B равна " << S << endl; // выдаются сумма чисел A и B
16     return 0;
17 }
```

Рисунок 10.

Выводы: освоил возможности отката к конкретным версиям программам.

Зафиксировал сделанные изменения. Добавил отчёт по лабораторной работе в формате PDF в папку doc репозитория. Отправил изменения из локального репозитория в удалённый репозиторий GitHub.

### 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 сокращённую 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), внесённую 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

online 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.

## 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` даёт 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.

#### 6. 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`

#### 7. Как добавить удалённый репозиторий для данного локального репозитория?

Для того, чтобы добавить удалённый репозиторий и присвоить ему имя (shortname), просто выполните команду `git remote add <shortname> <url>`.

#### 8. 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>`.

#### 9. Как выполнить просмотр удаленного репозитория?

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

#### 10. 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.

#### 11. 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`.

12. 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 удалённые ветки, у которых нет локального аналога.

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