**МИНИCTEPCTBO 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 кроссплатформенного программирования»

Тема: «Исследование возможностей Git для работы с локальным репозиторием»

Выполнил: студент 1 курса

группы ИВТ-б-о-21-1

Толубаев Рамиль Ахметович

Ставрополь 2022

Ход работы:

1. Создал новый репозиторий со всеми дополнениями:

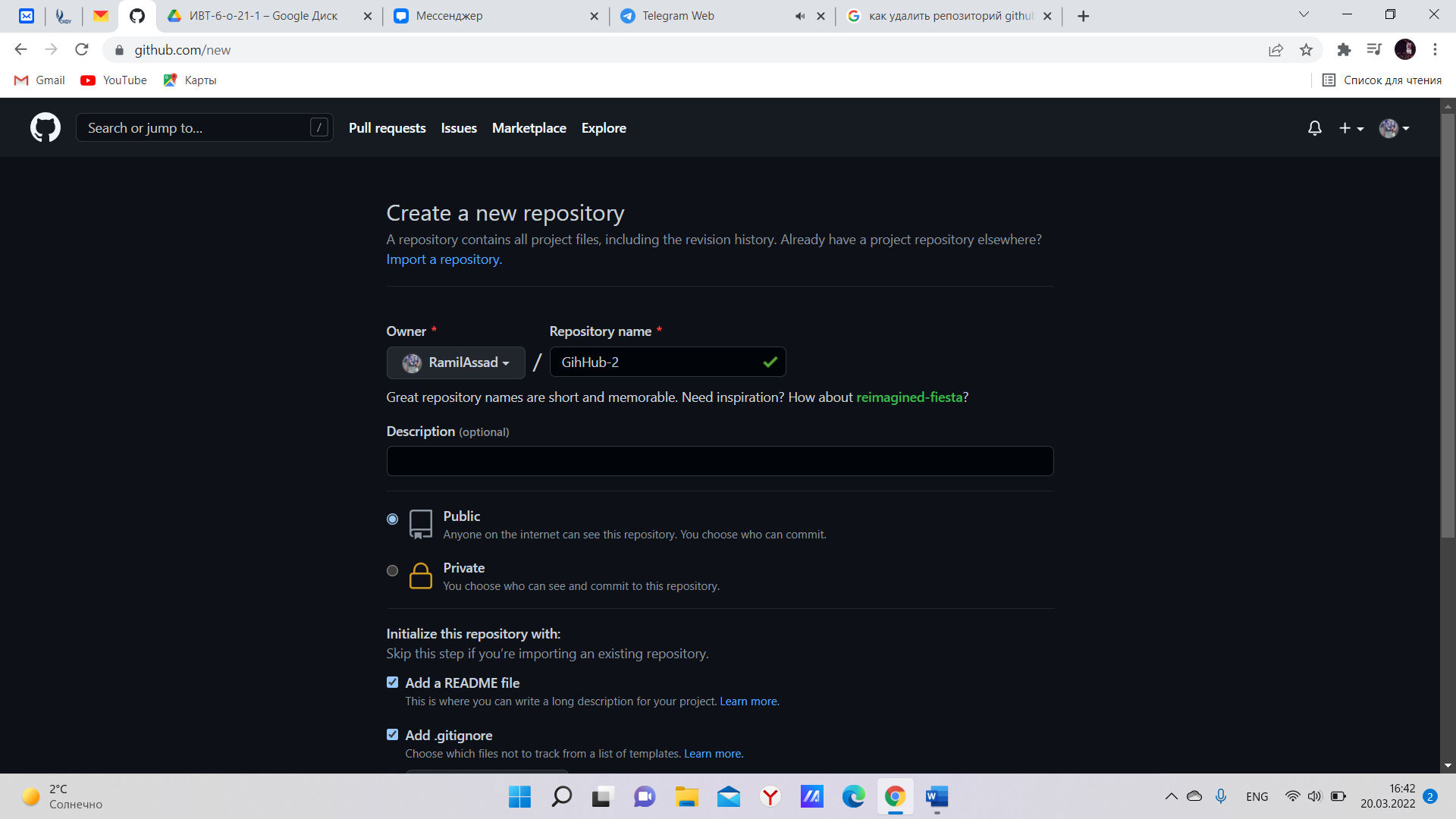


Рисунок 1. Новый репозиторий

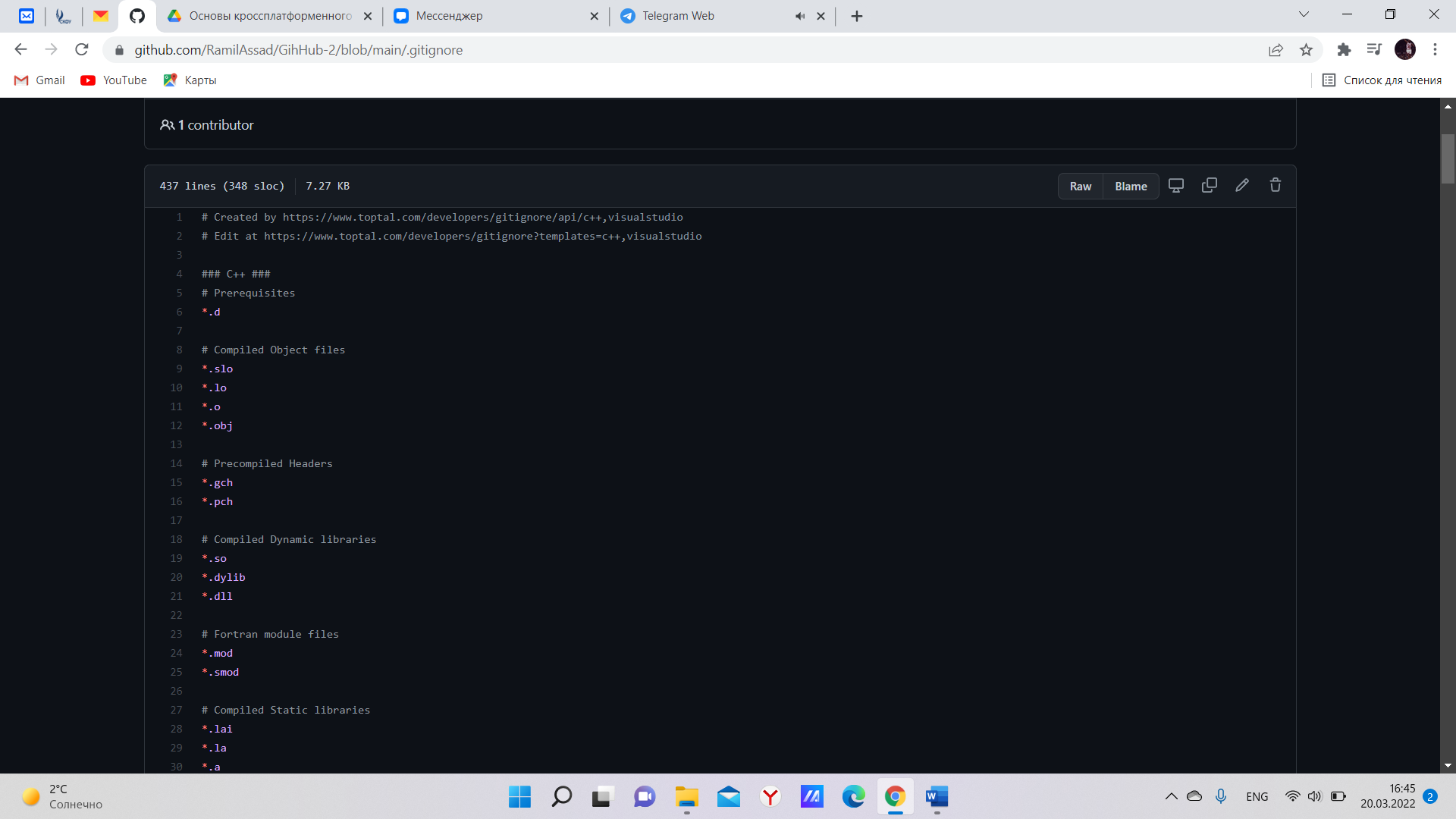


Рисунок 2. Gitignore

1. Клонировал репозитории на рабочий компьютер

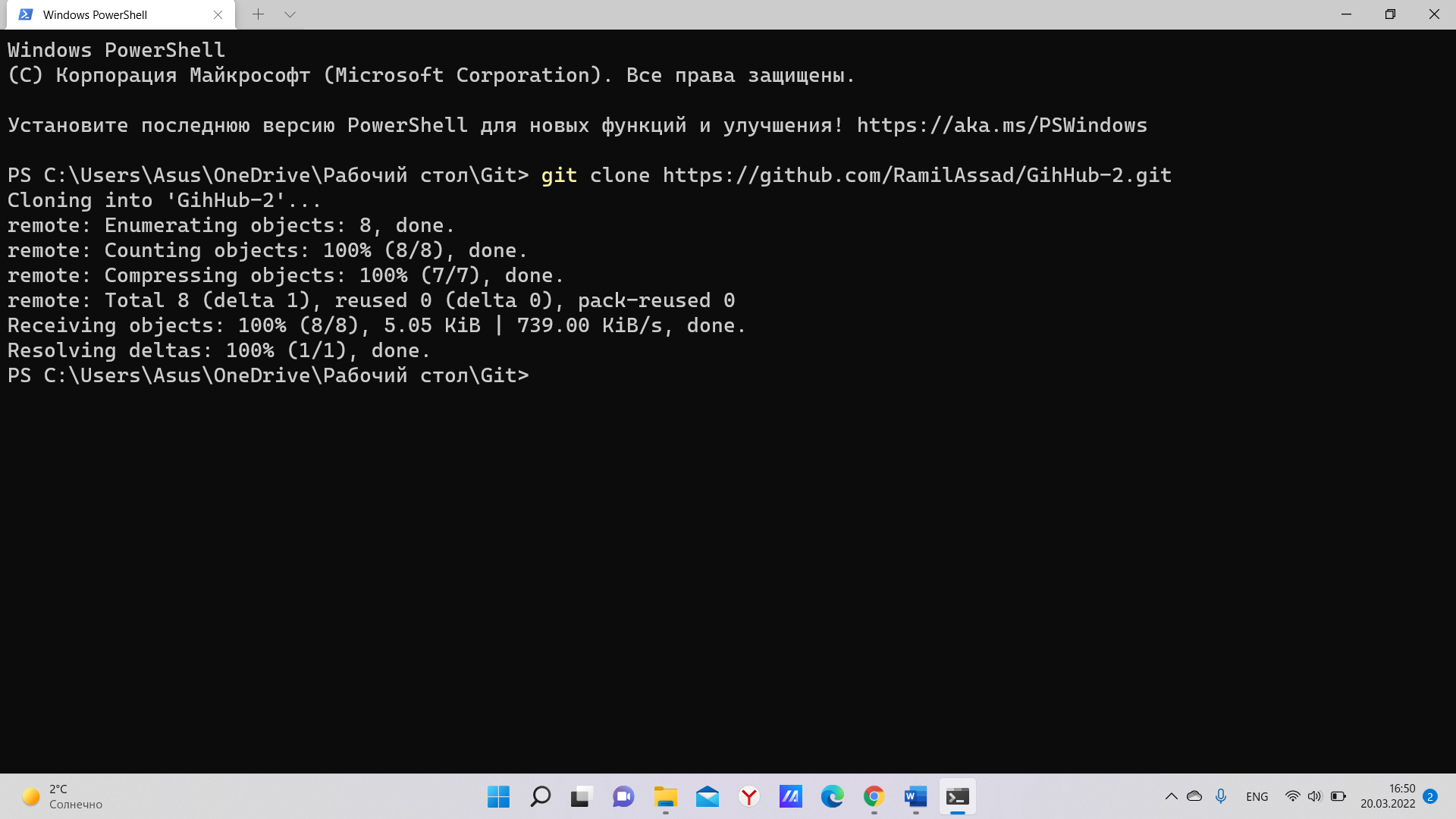


Рисунок 3. Работа в консоли

1. Изменил информацию в README, сделал коммит и запушил на удаленный репозитории

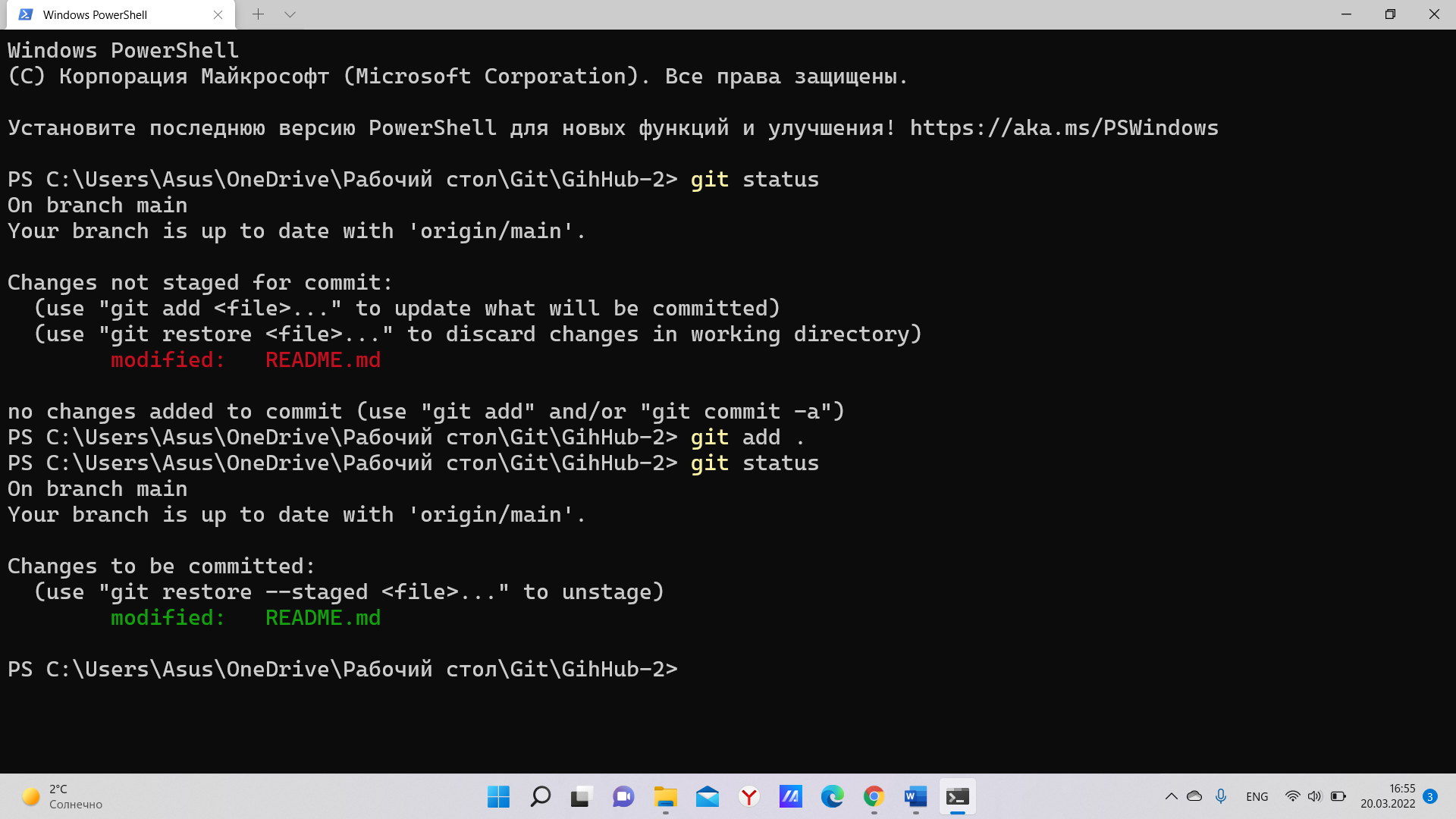


Рисунок 4. Изменение в README

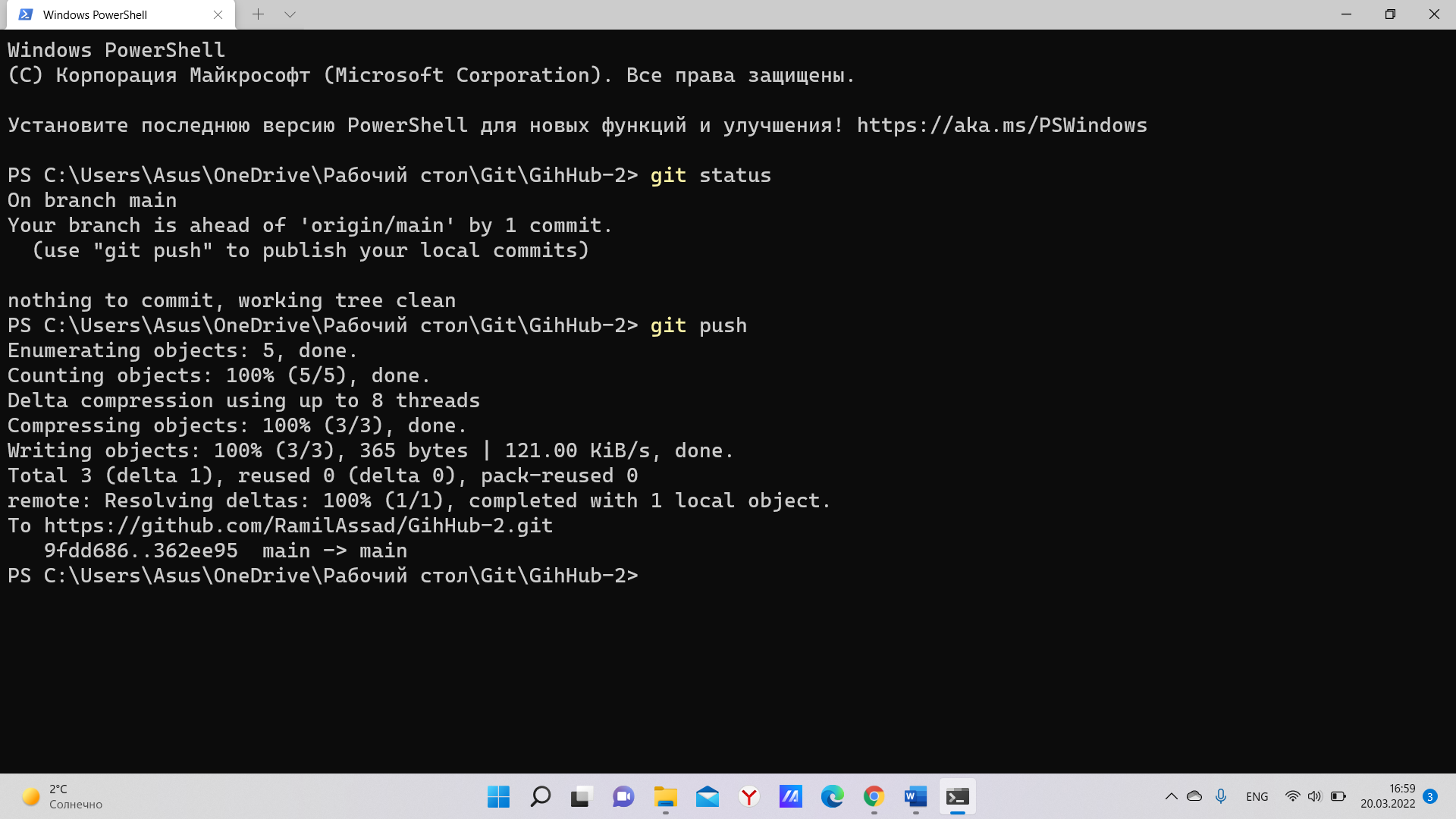


Рисунок 5. Пуш изменения

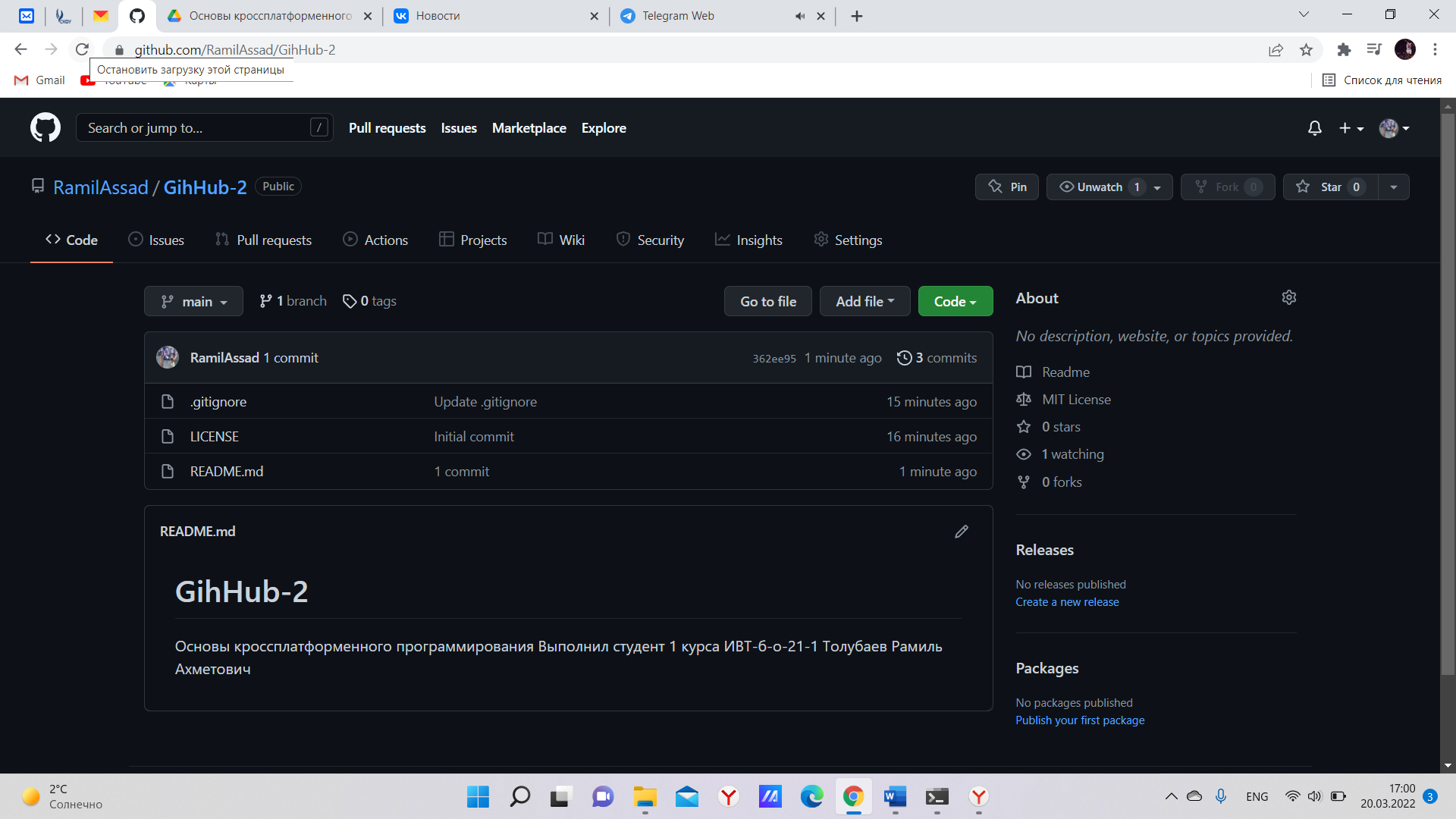


Рисунок 6. Удаленный репозитории

1. Написал программу в репозитории, закоммитил и сделал пуш

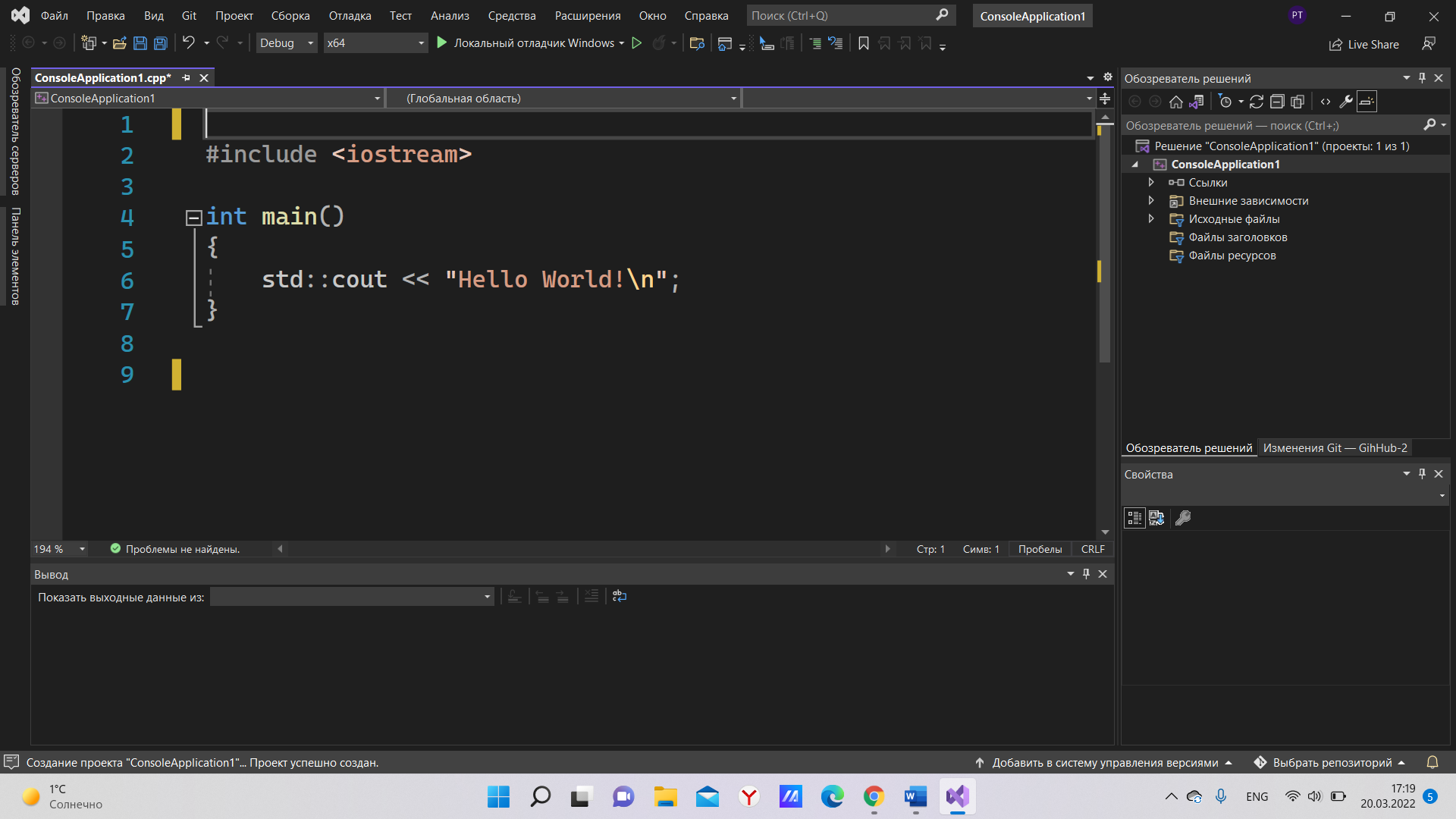


Рисунок 7. Программа на C++

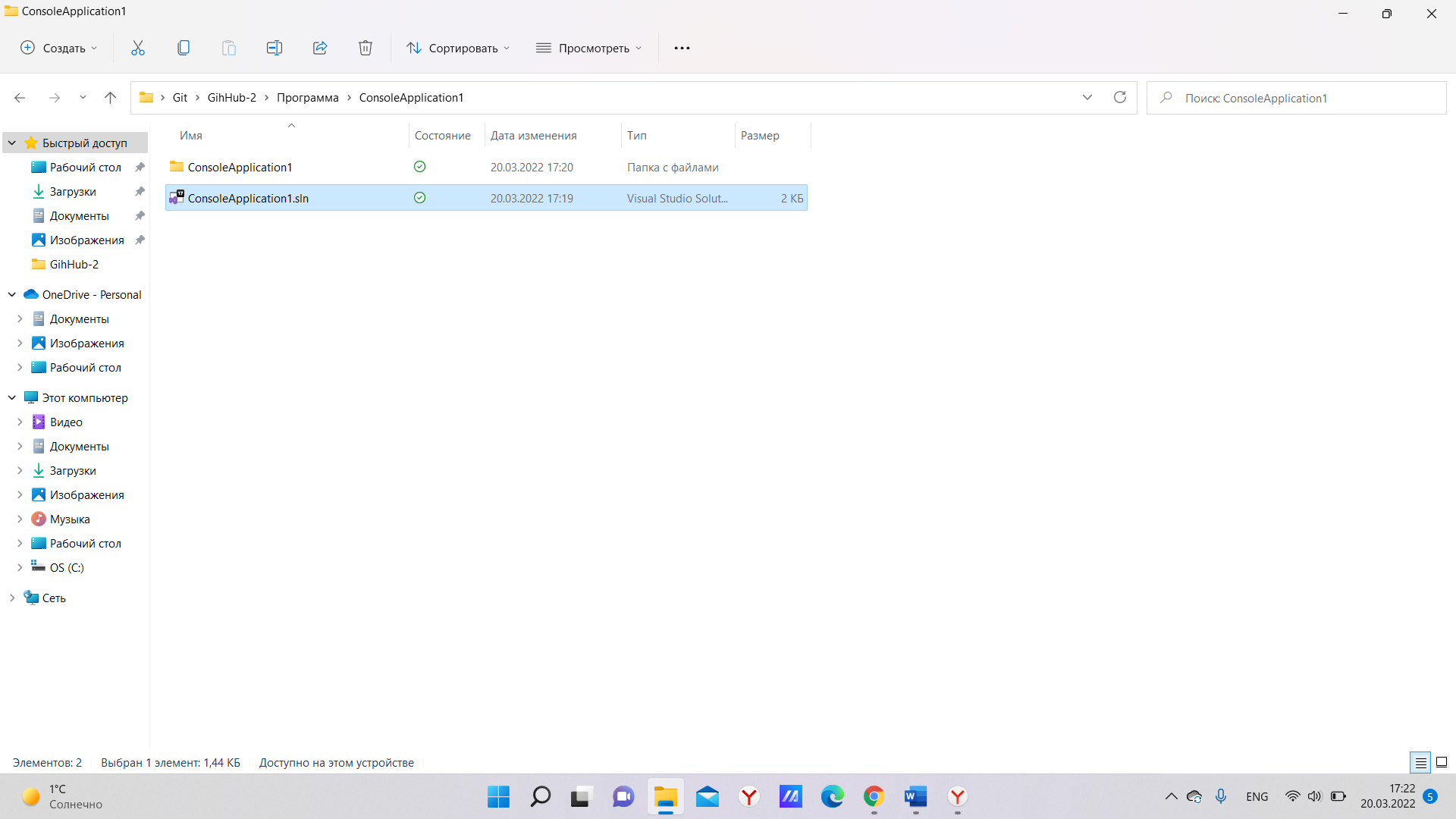


Рисунок 8. Папка с программой

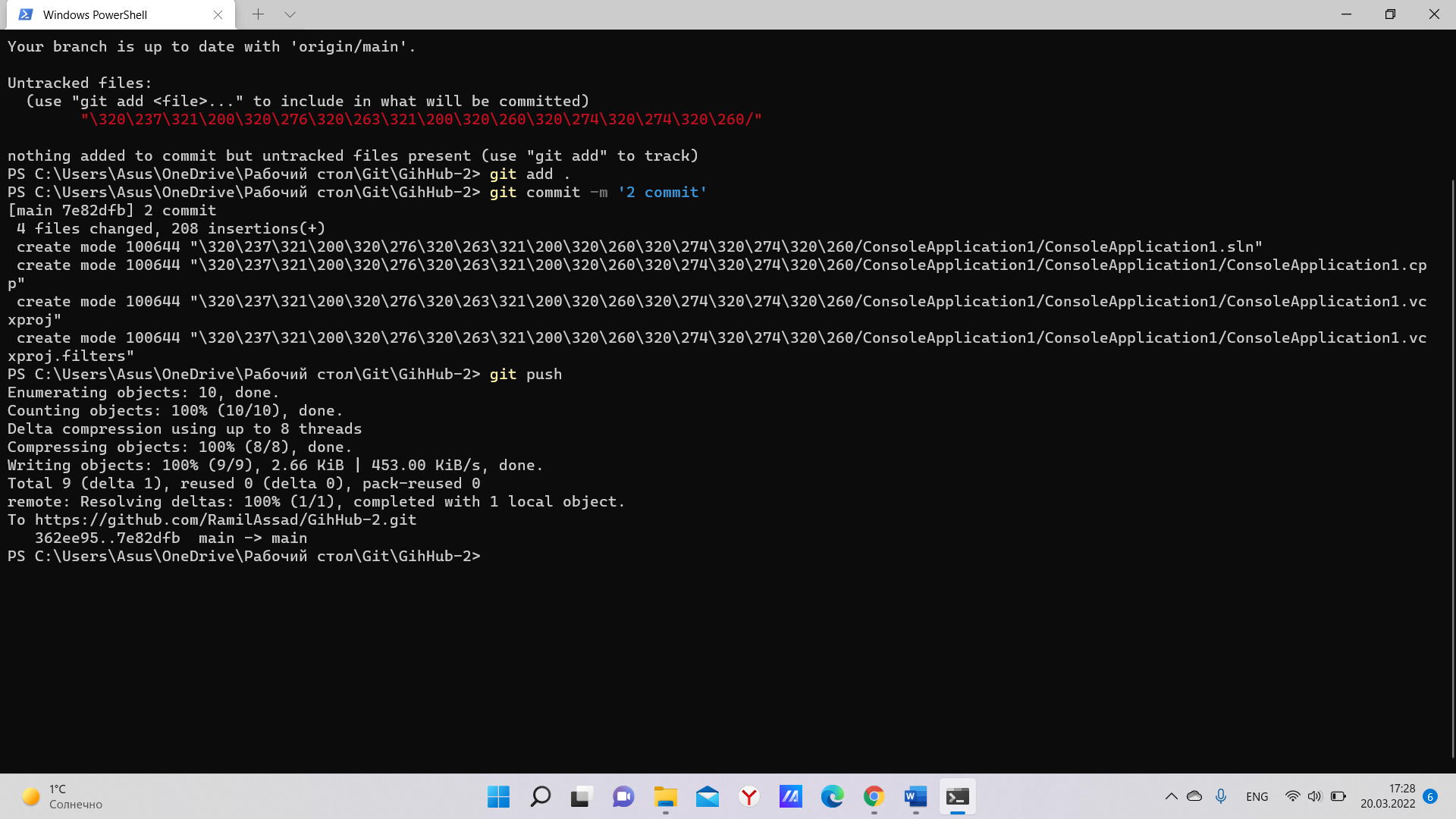


Рисунок 9. Работа в консоли

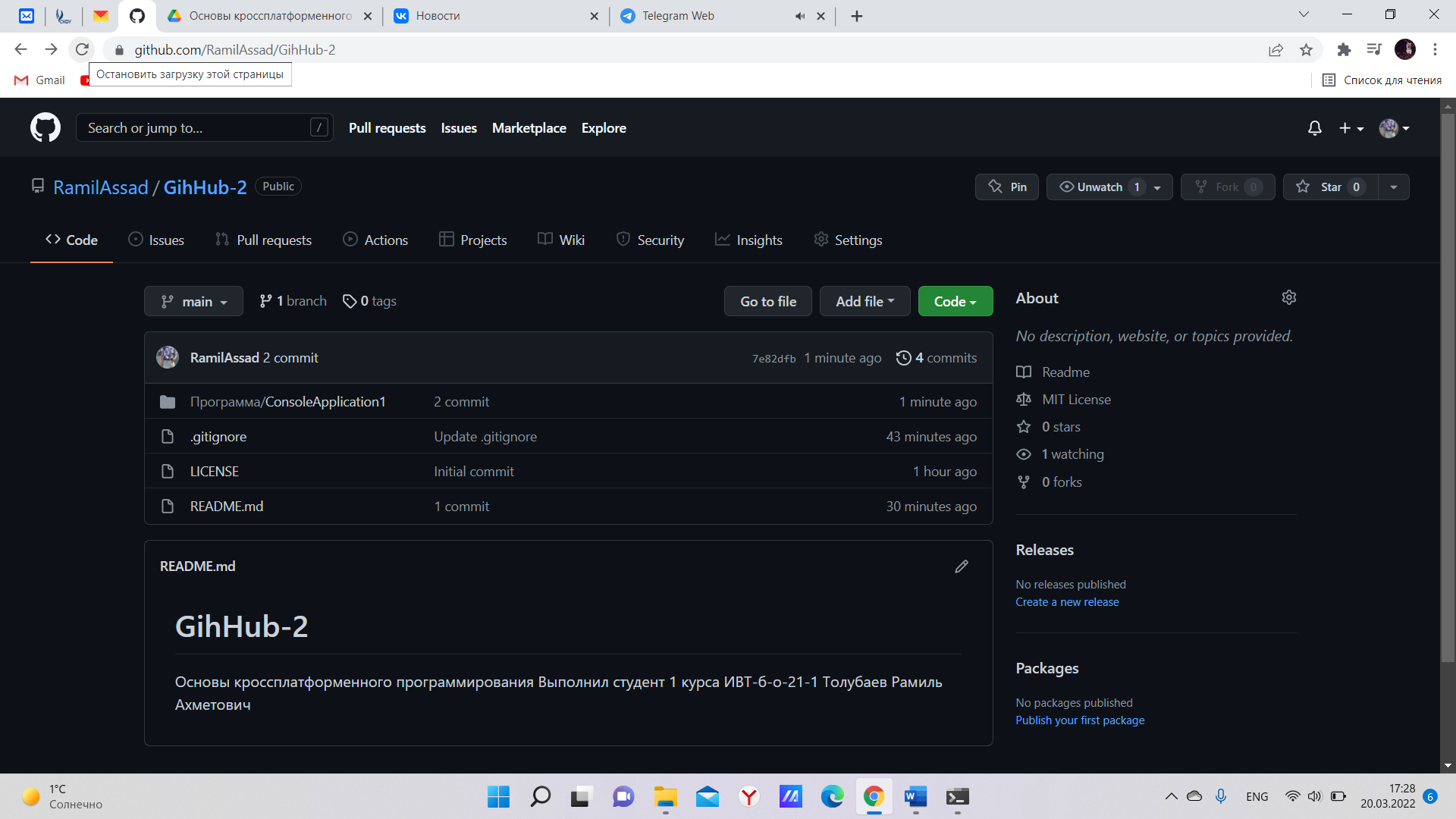


Рисунок 10. GitHub

1. Сделал коммиты в процессе изменения программы, отметил их тегами и запушил

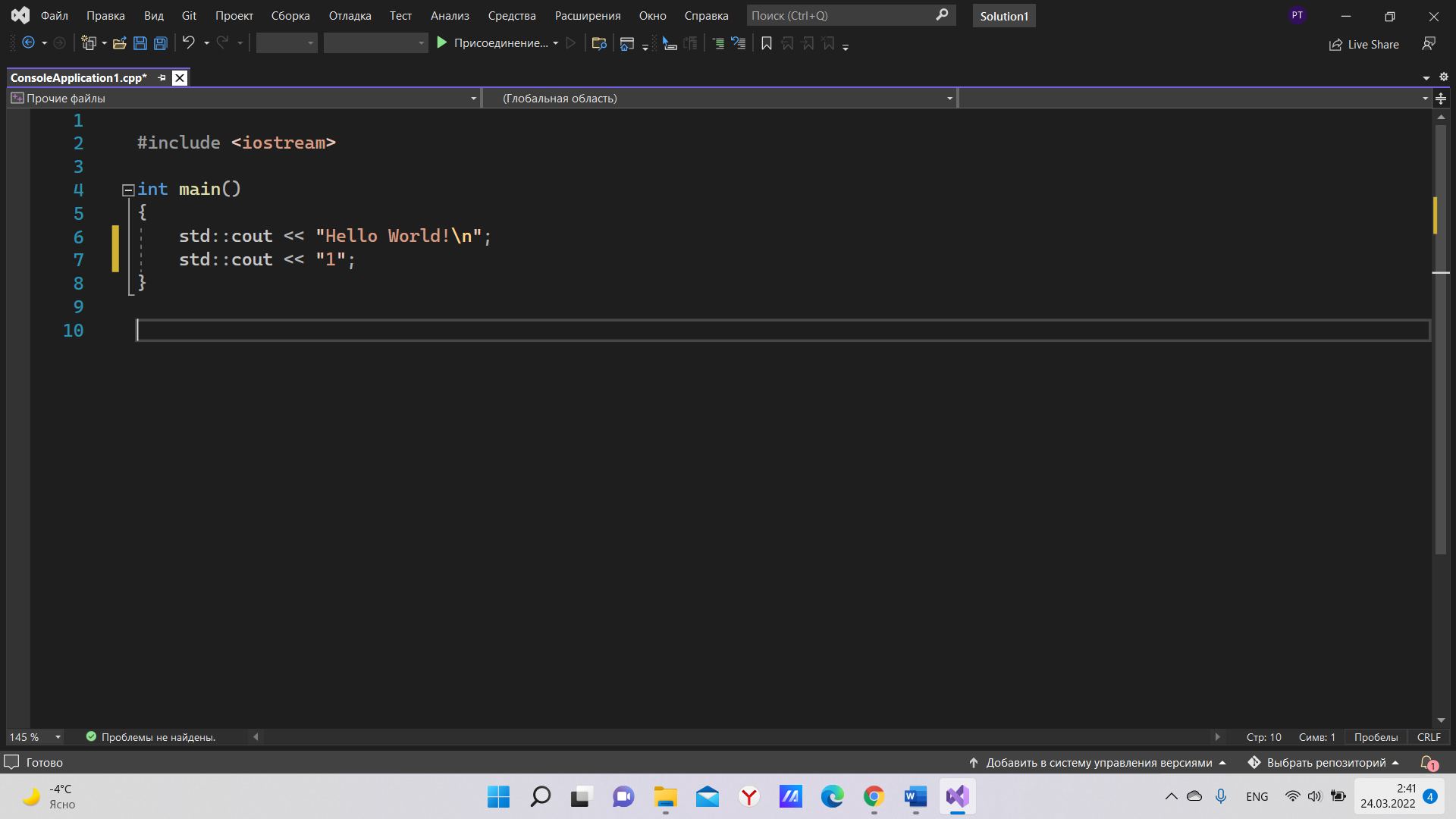


Рисунок 11. Изменение в программе

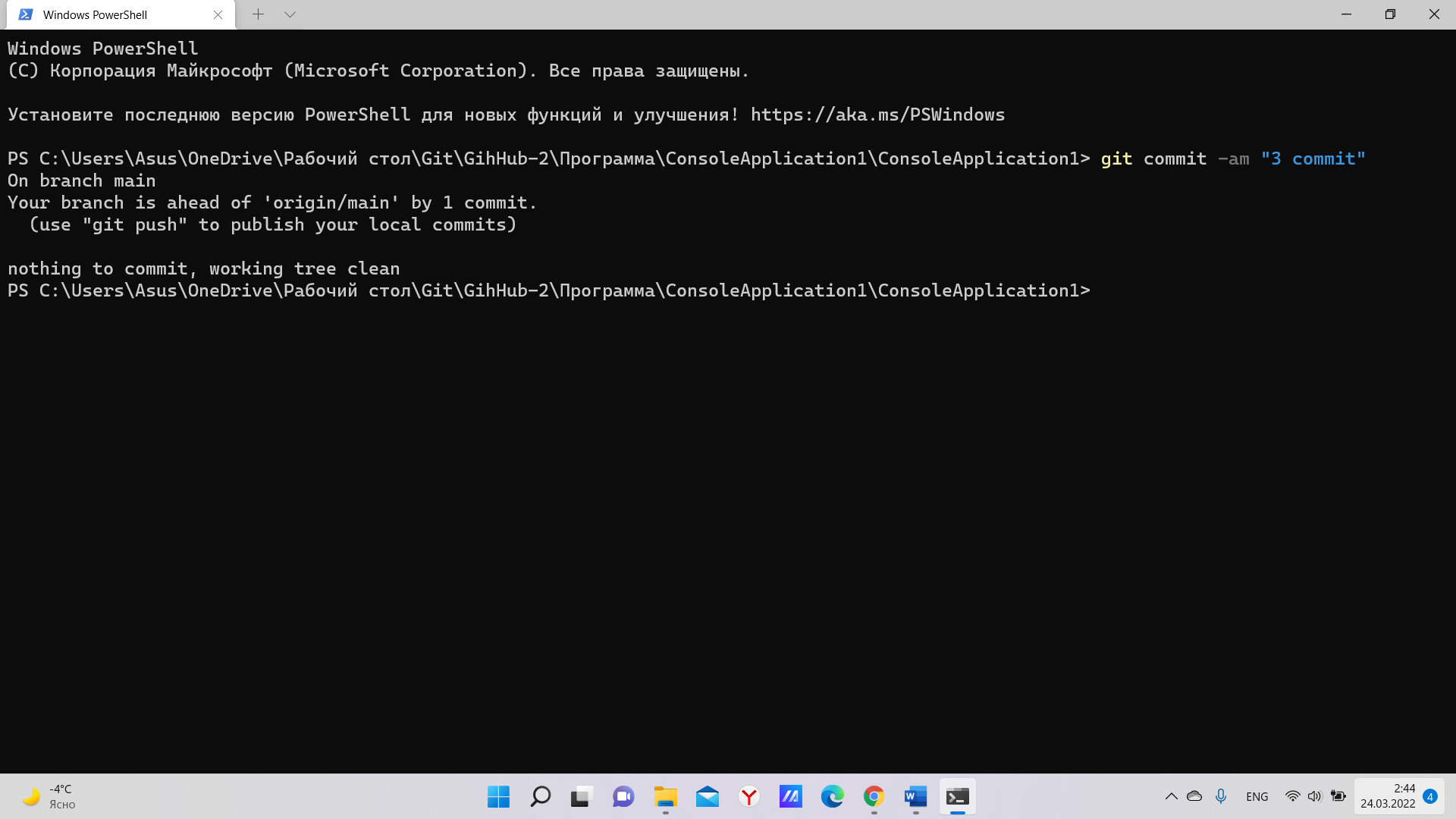


Рисунок 12. Коммит изменения

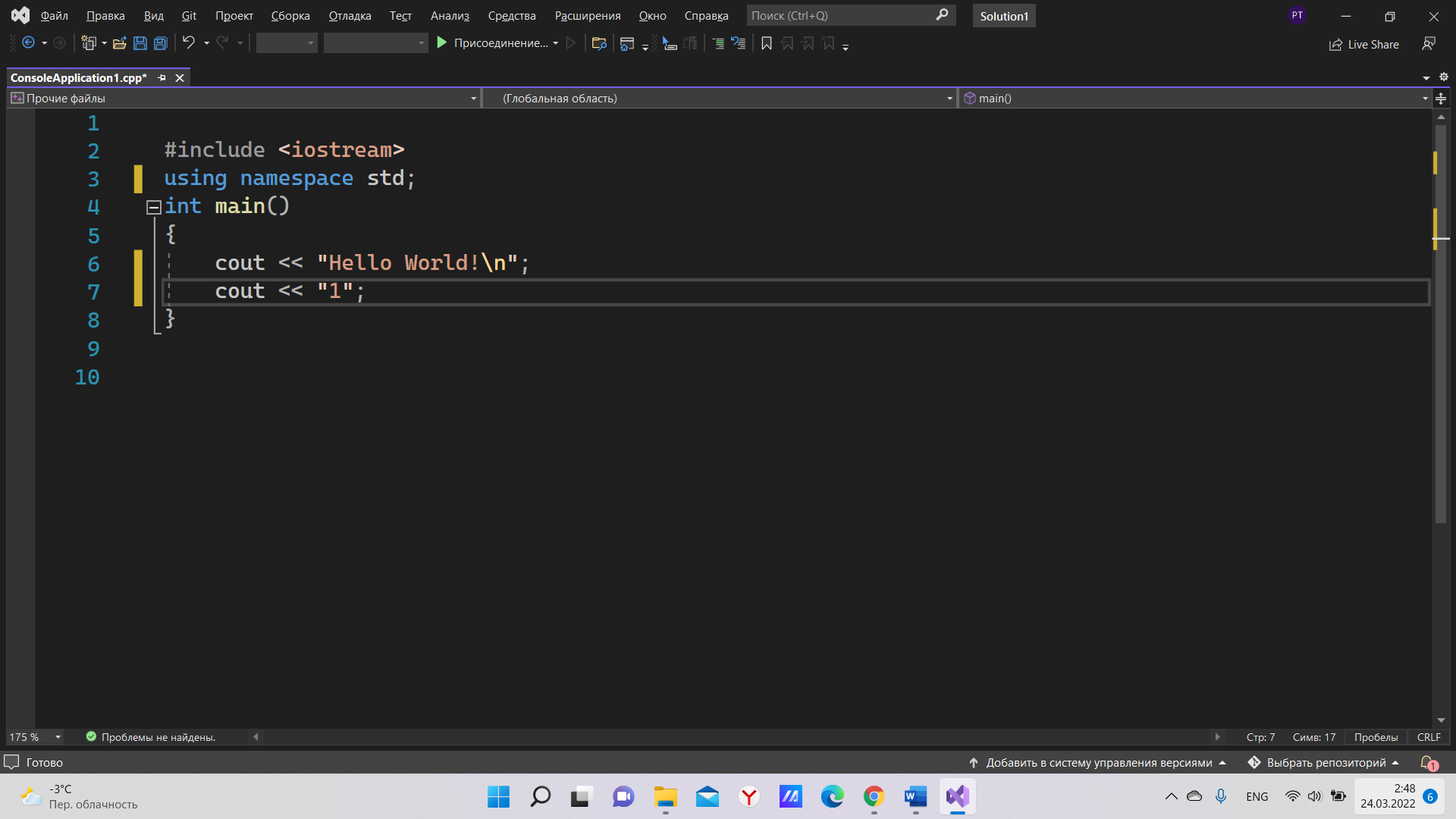


Рисунок 13. Изменение в программе

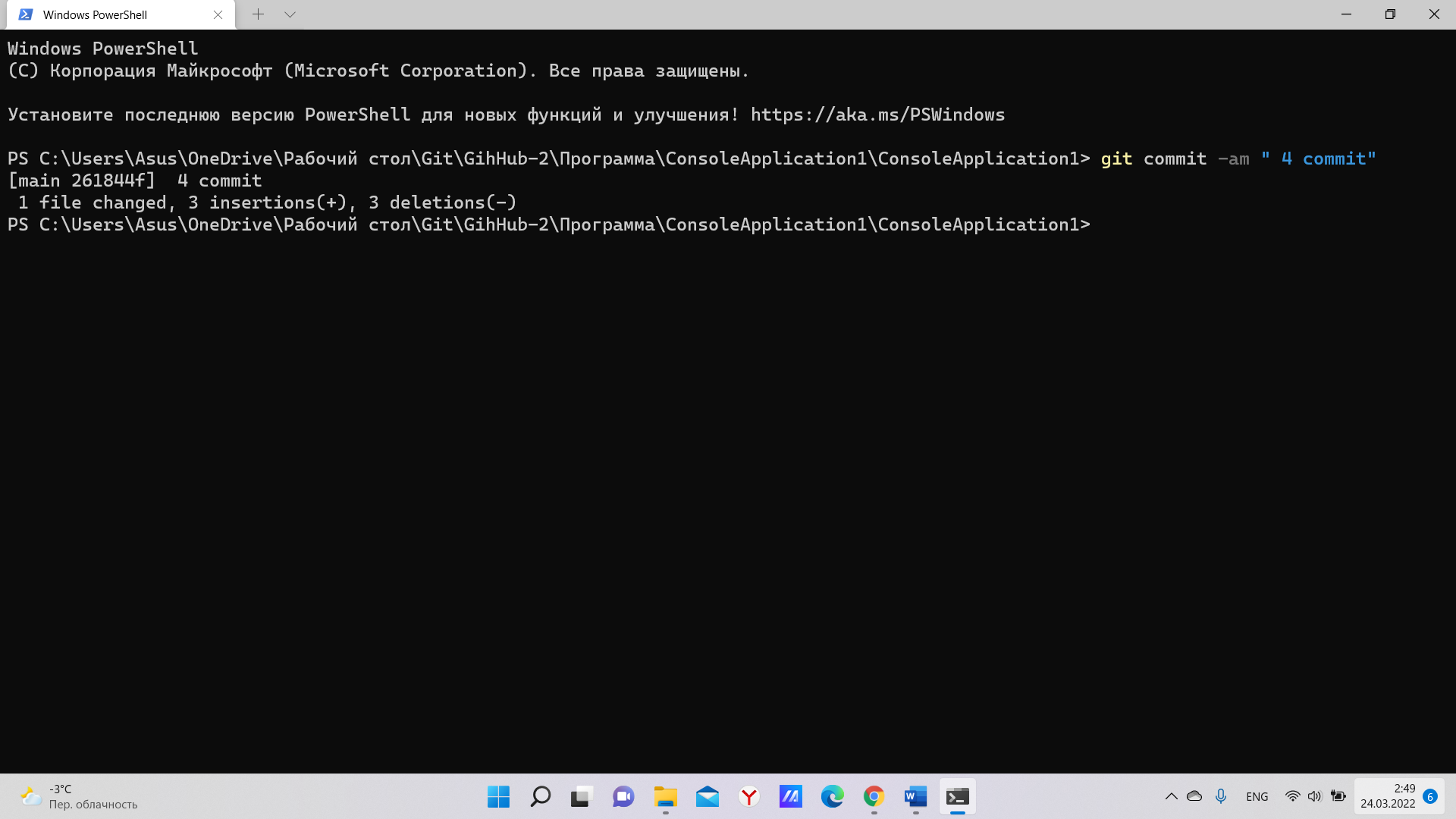


Рисунок 14. Коммит изменения

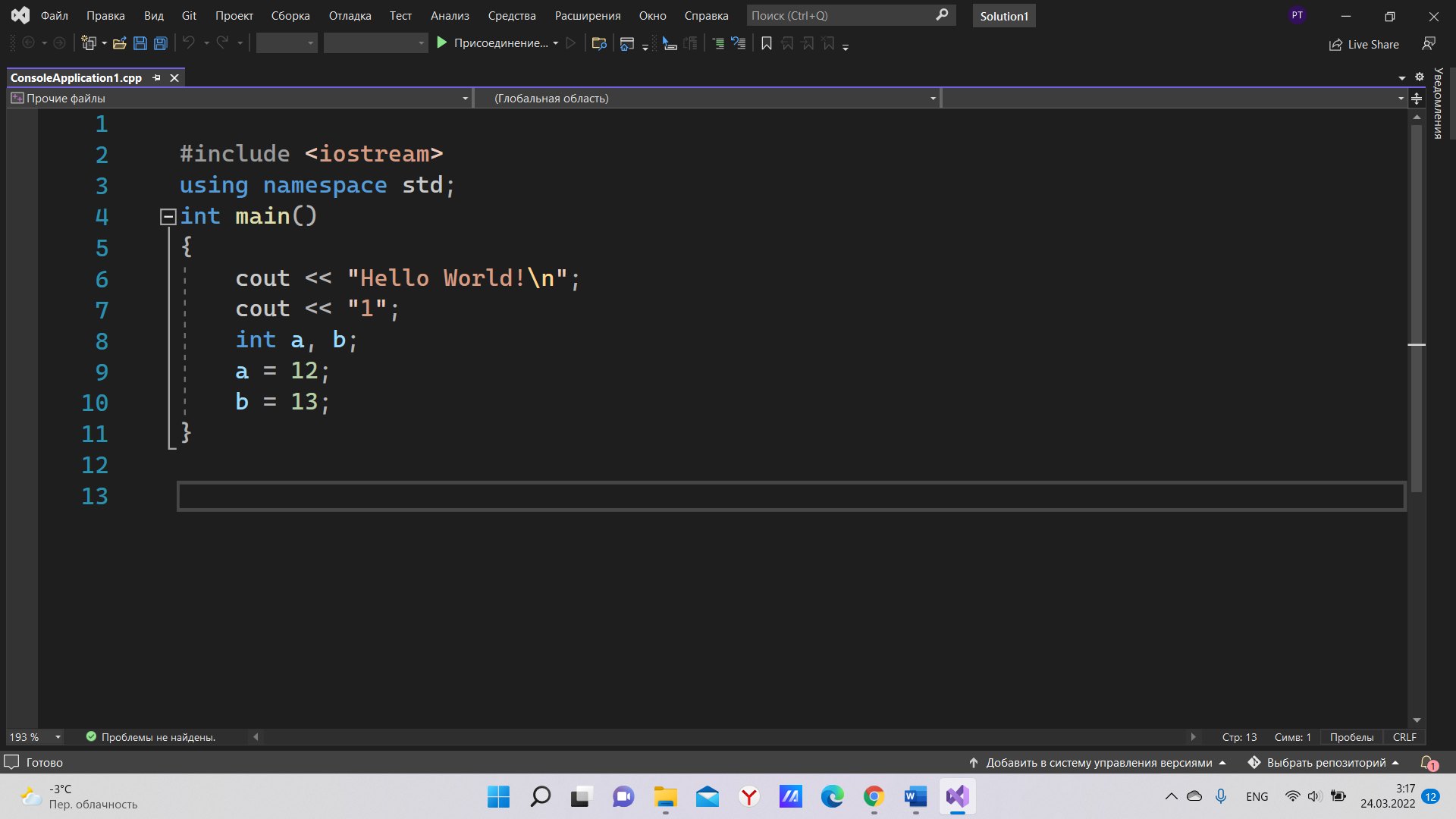


Рисунок 15. Изменение в программе

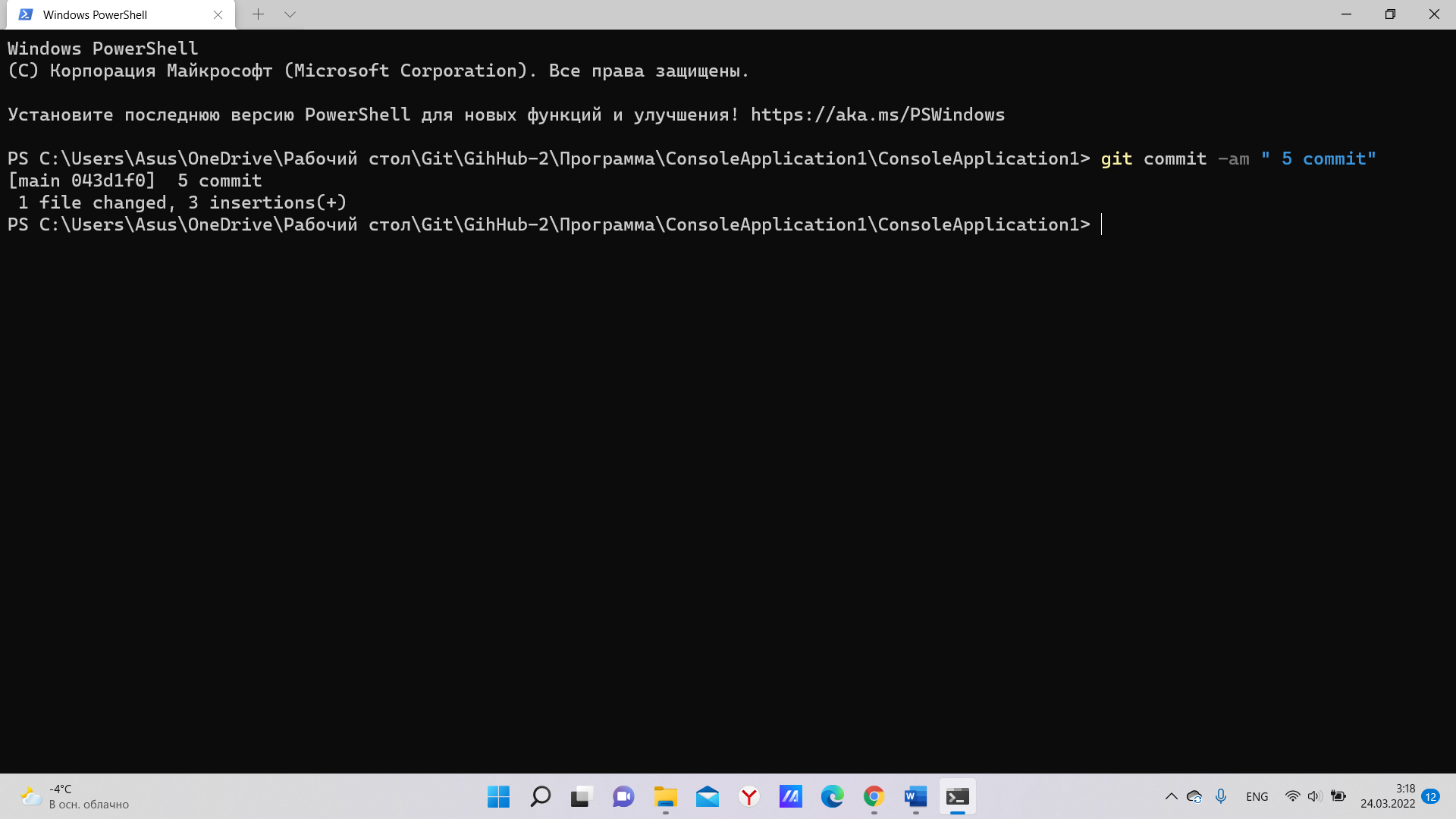


Рисунок 16. Коммит изменения

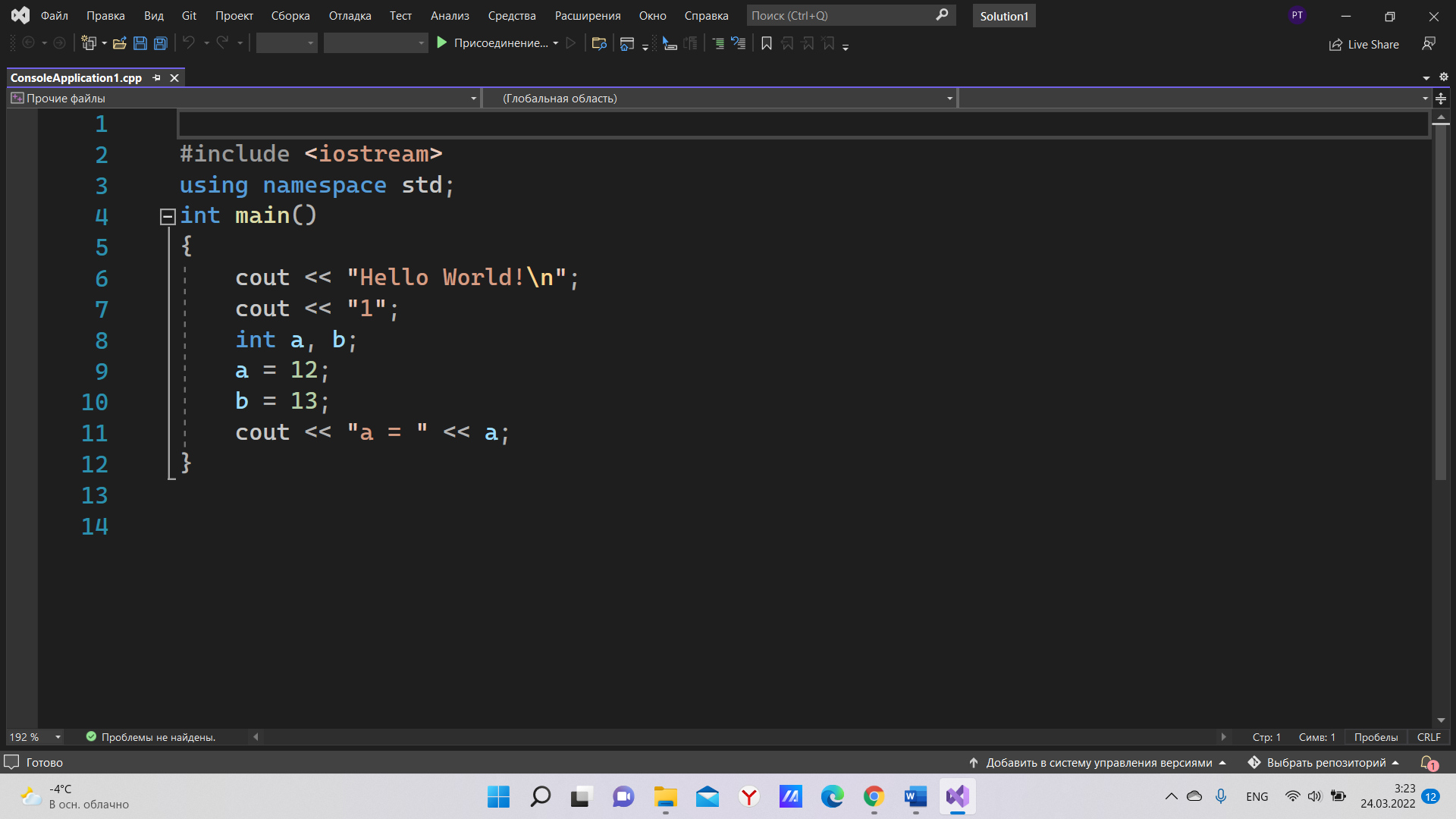


Рисунок 15. Изменение в программе

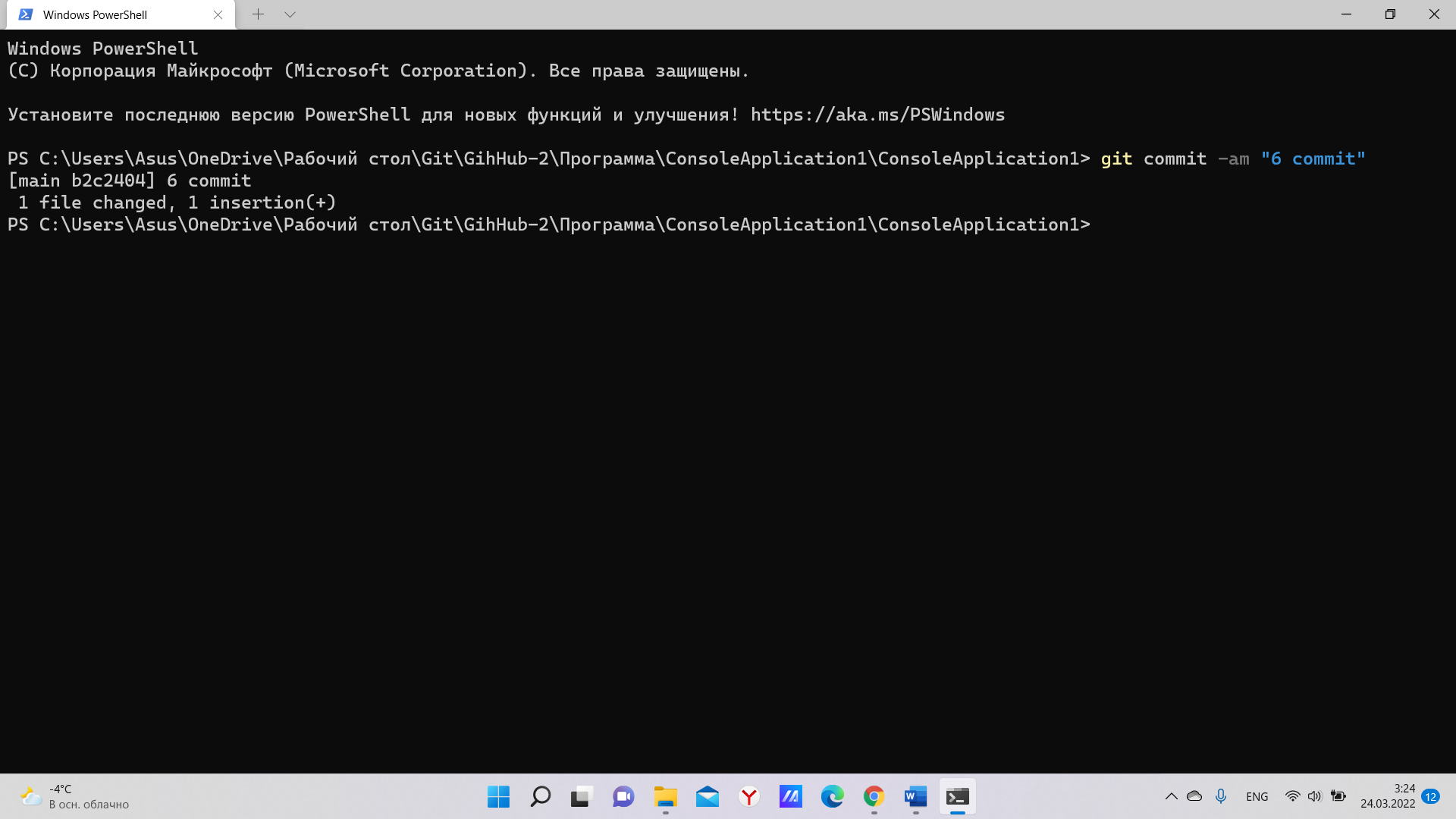
 Рисунок 16. Коммит изменения



Рисунок 17. Изменение в программе

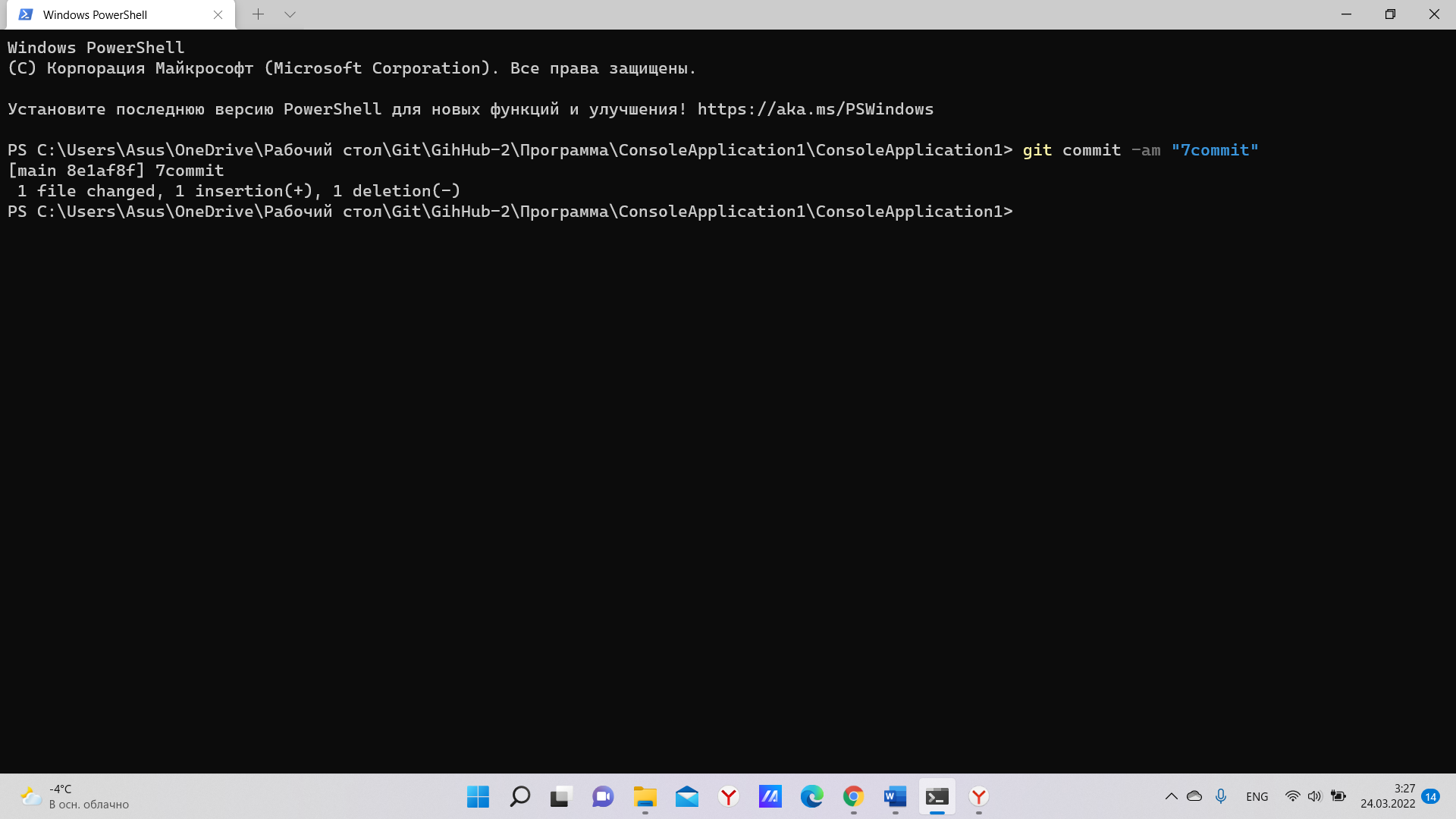


Рисунок 18. Коммит изменения

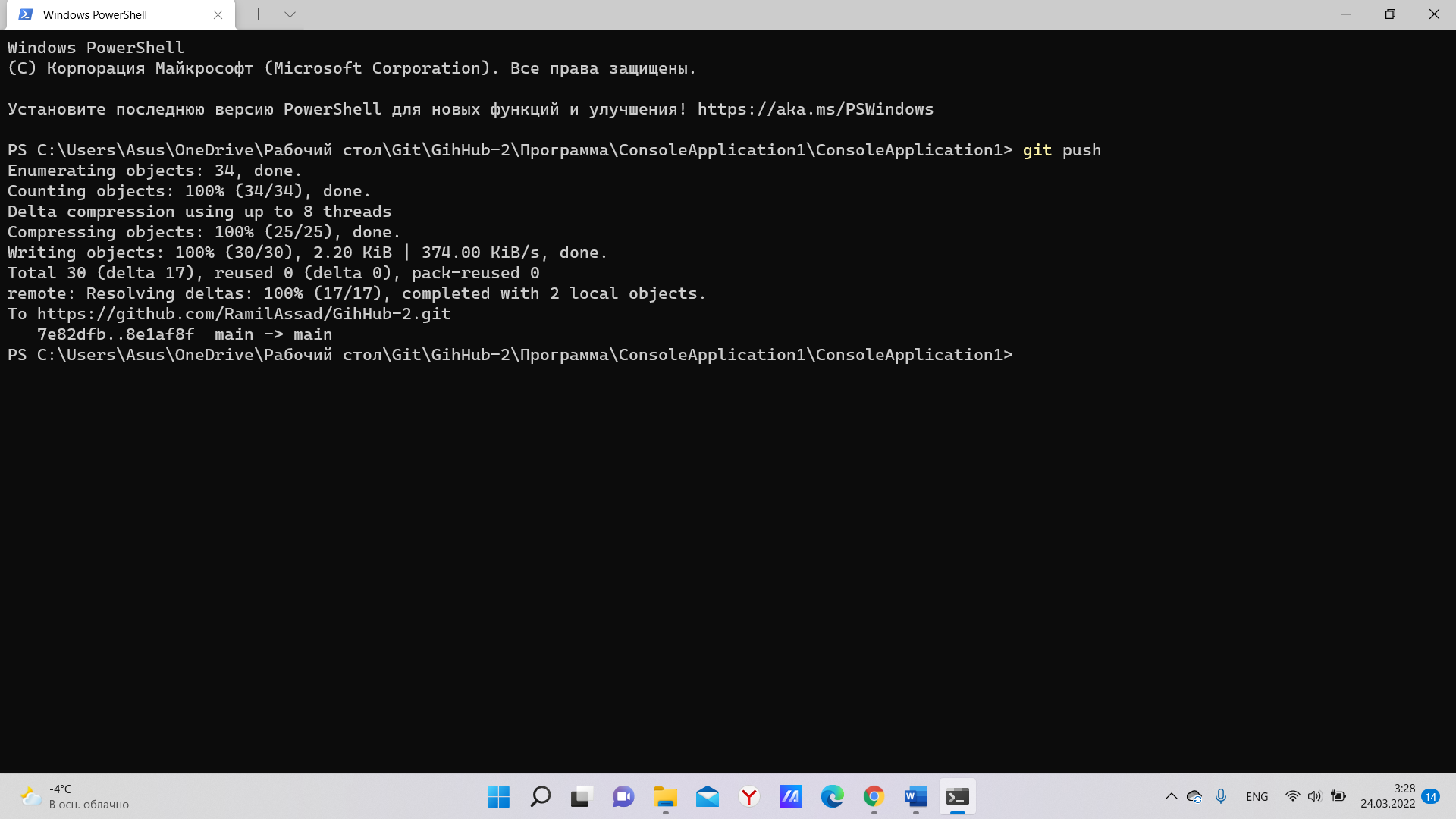


Рисунок 19. Пуш изменений

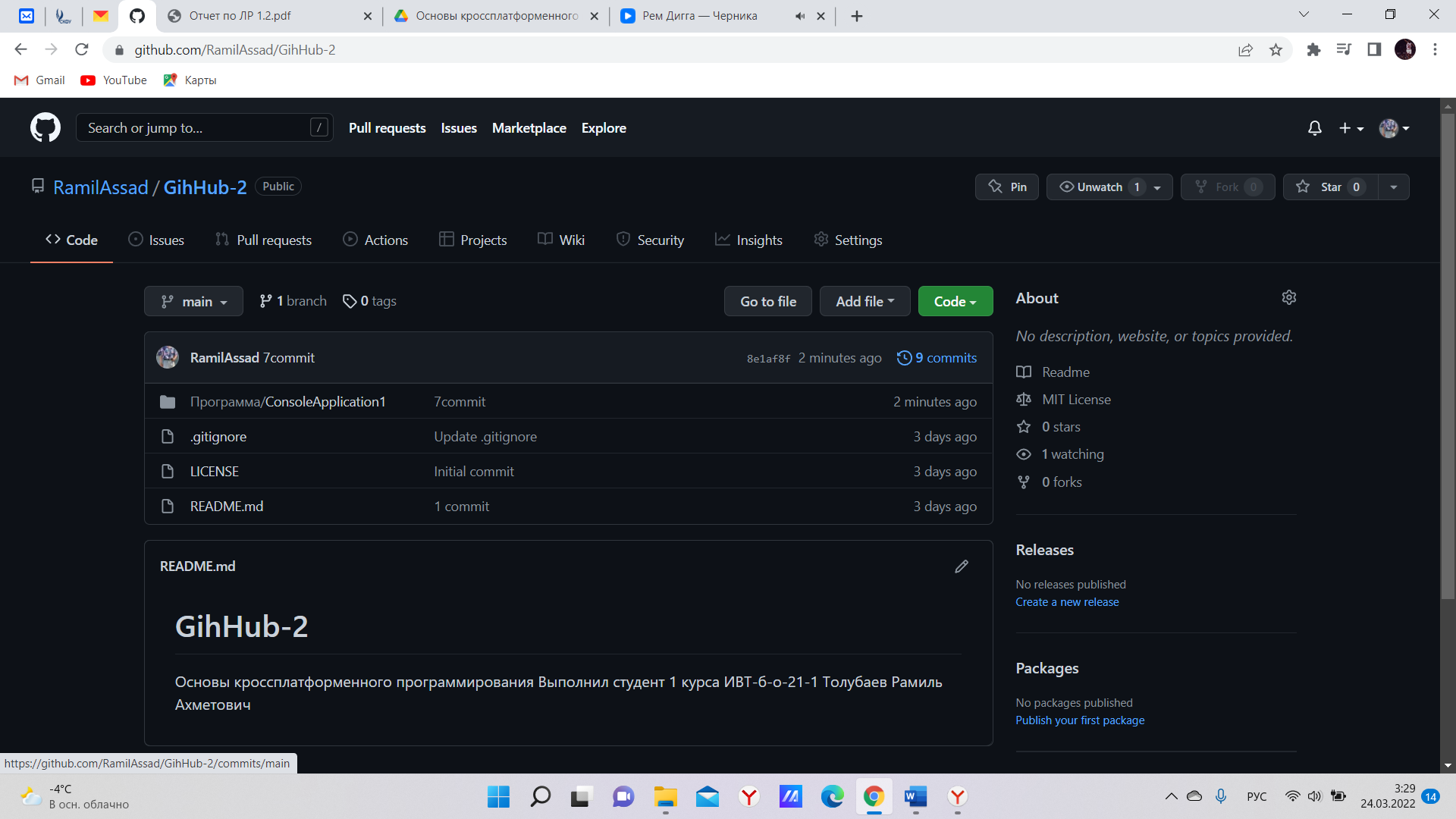


Рисунок 20. Изменение на GitHub

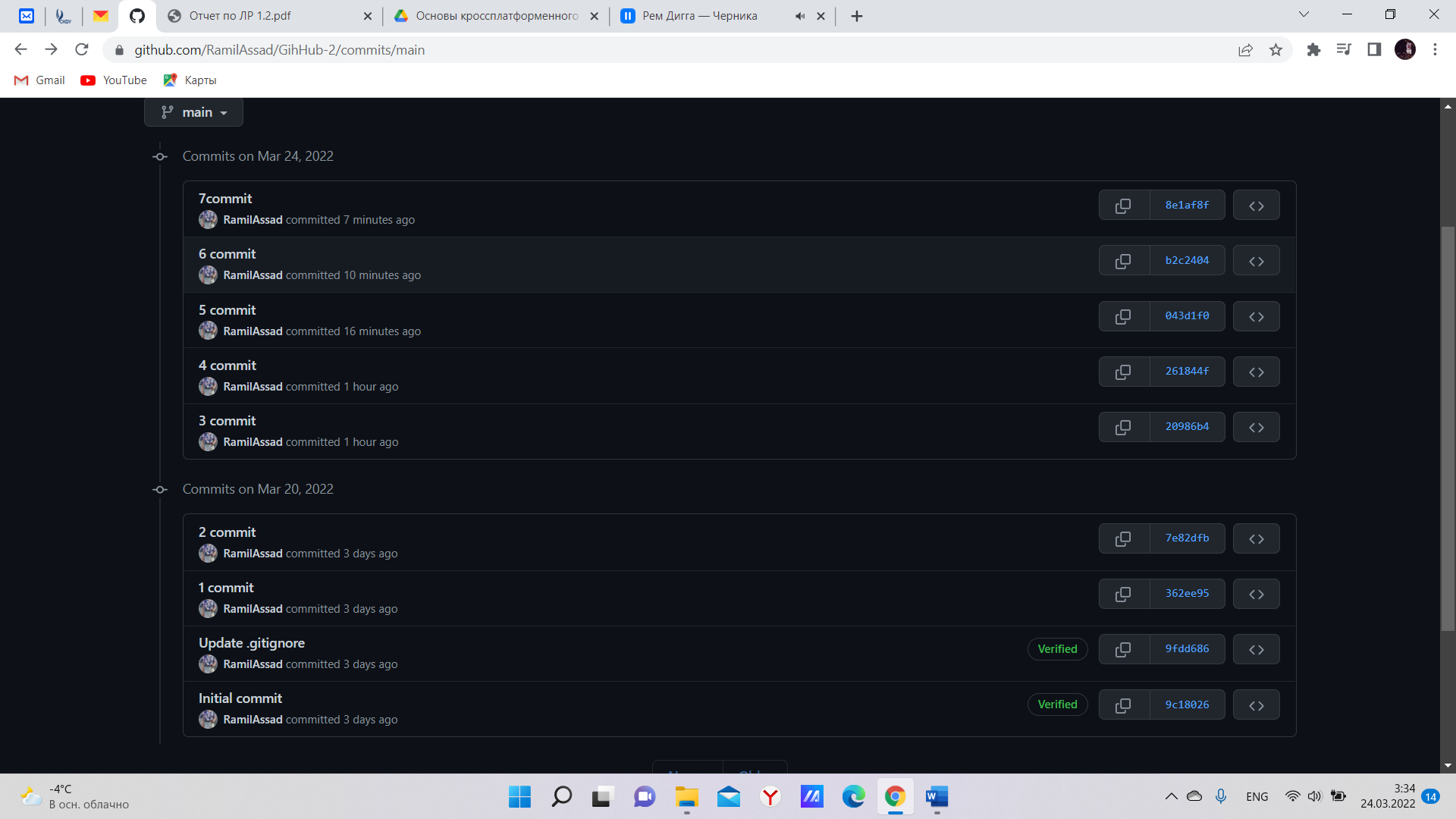


Рисунок 21. История коммитов

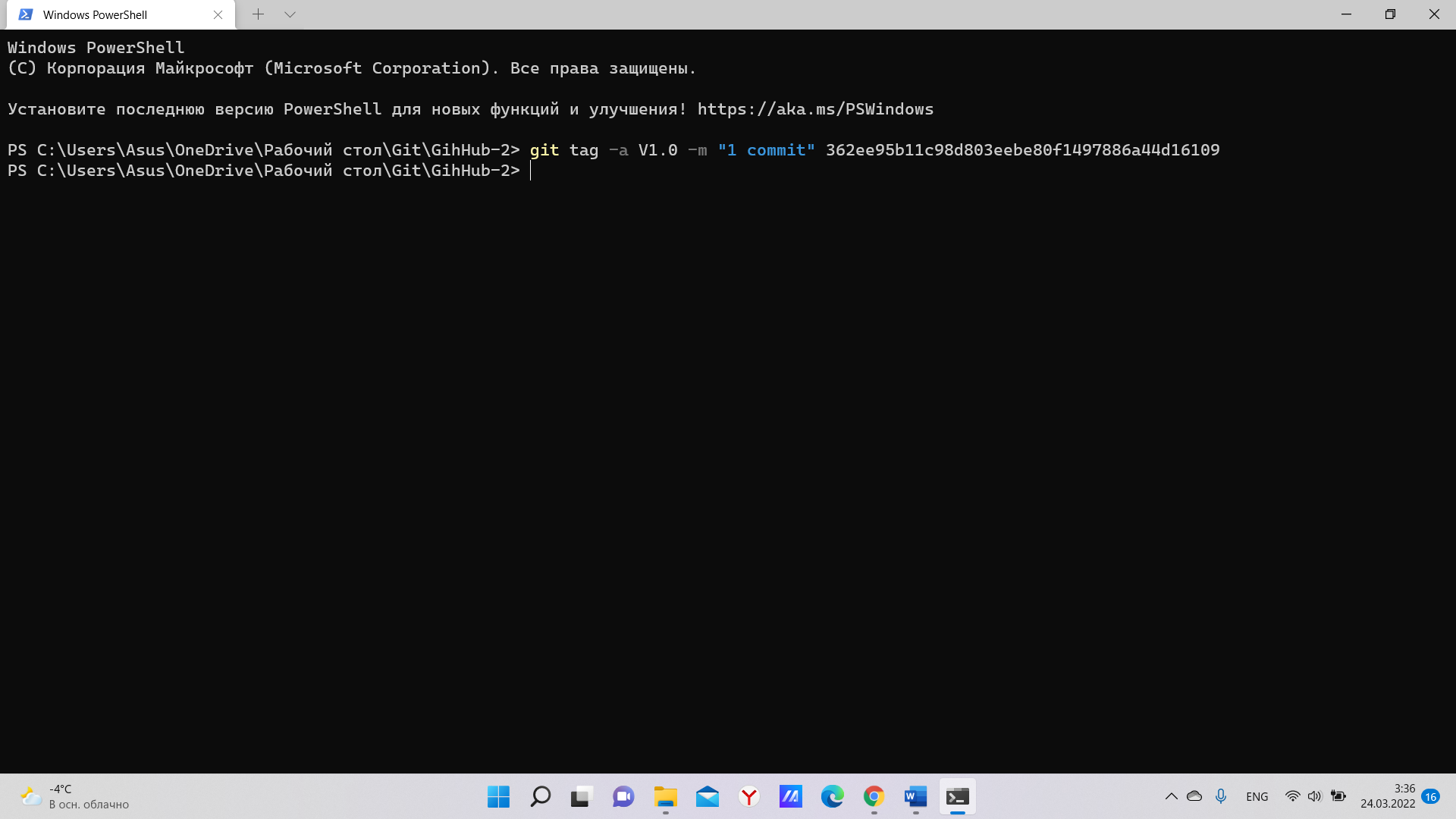


Рисунок 22. Присваивание тега к коммиту

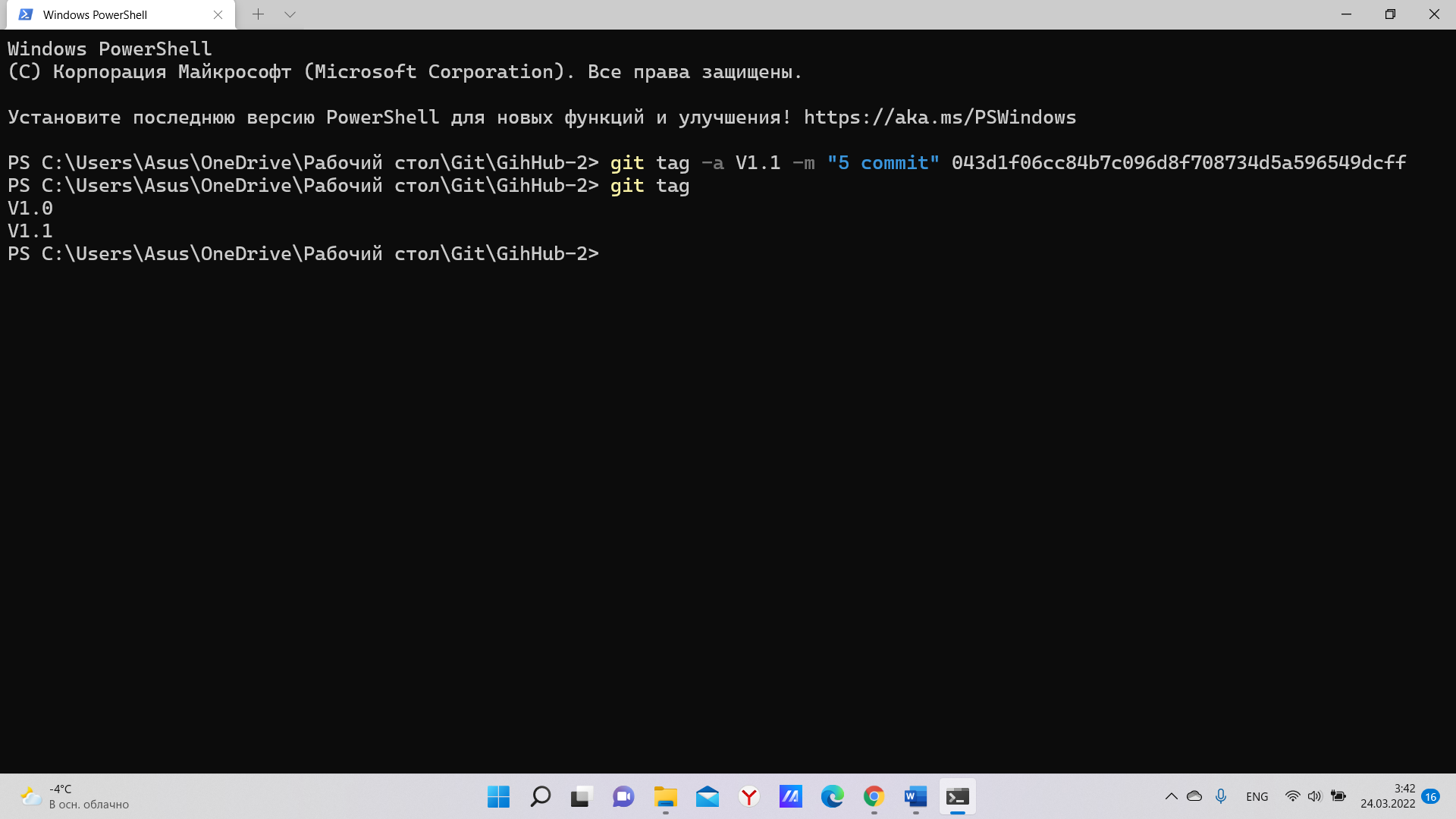


Рисунок 23. Присваивание тега к коммиту

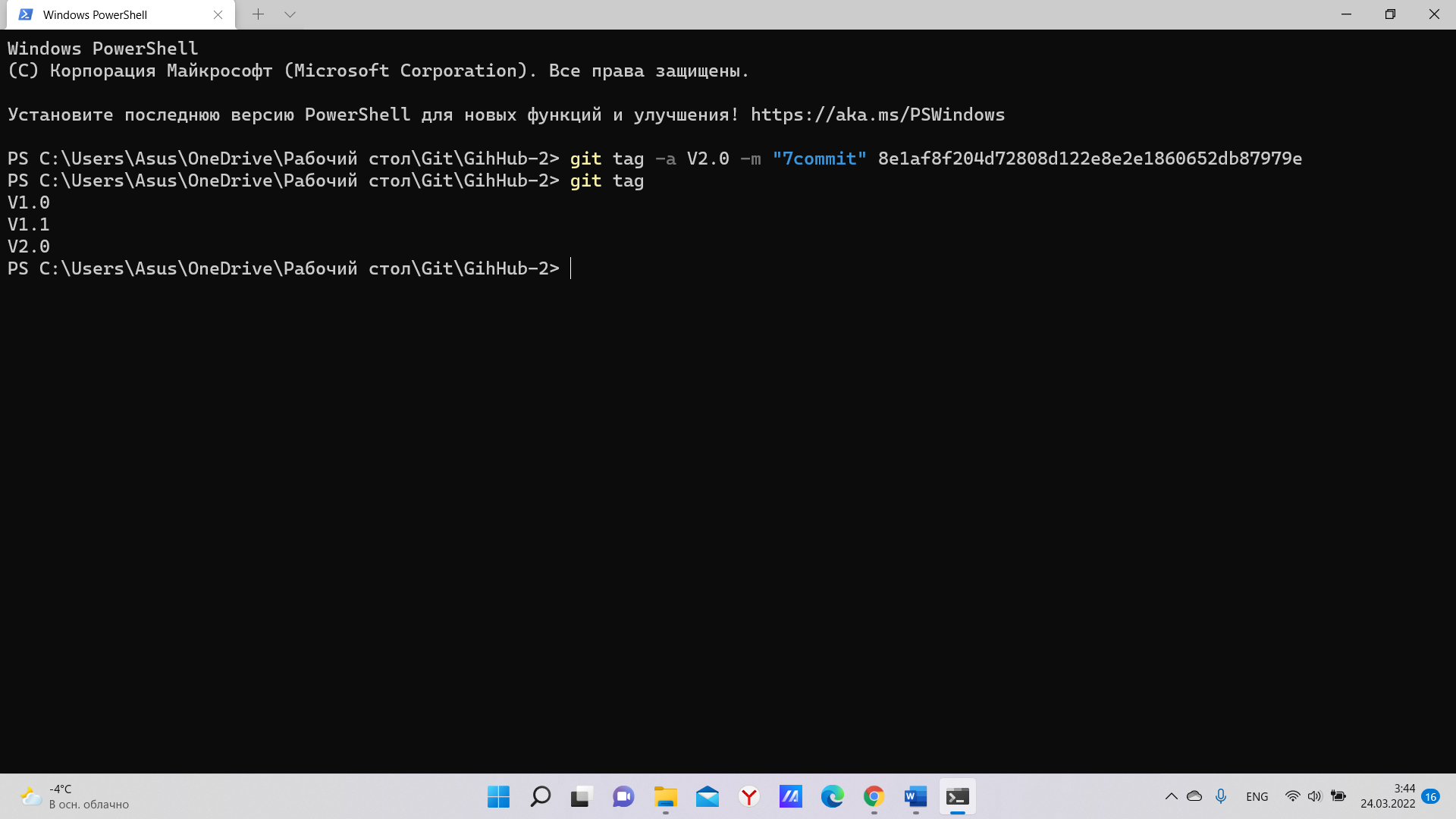


Рисунок 24. Присваивание тега к коммиту

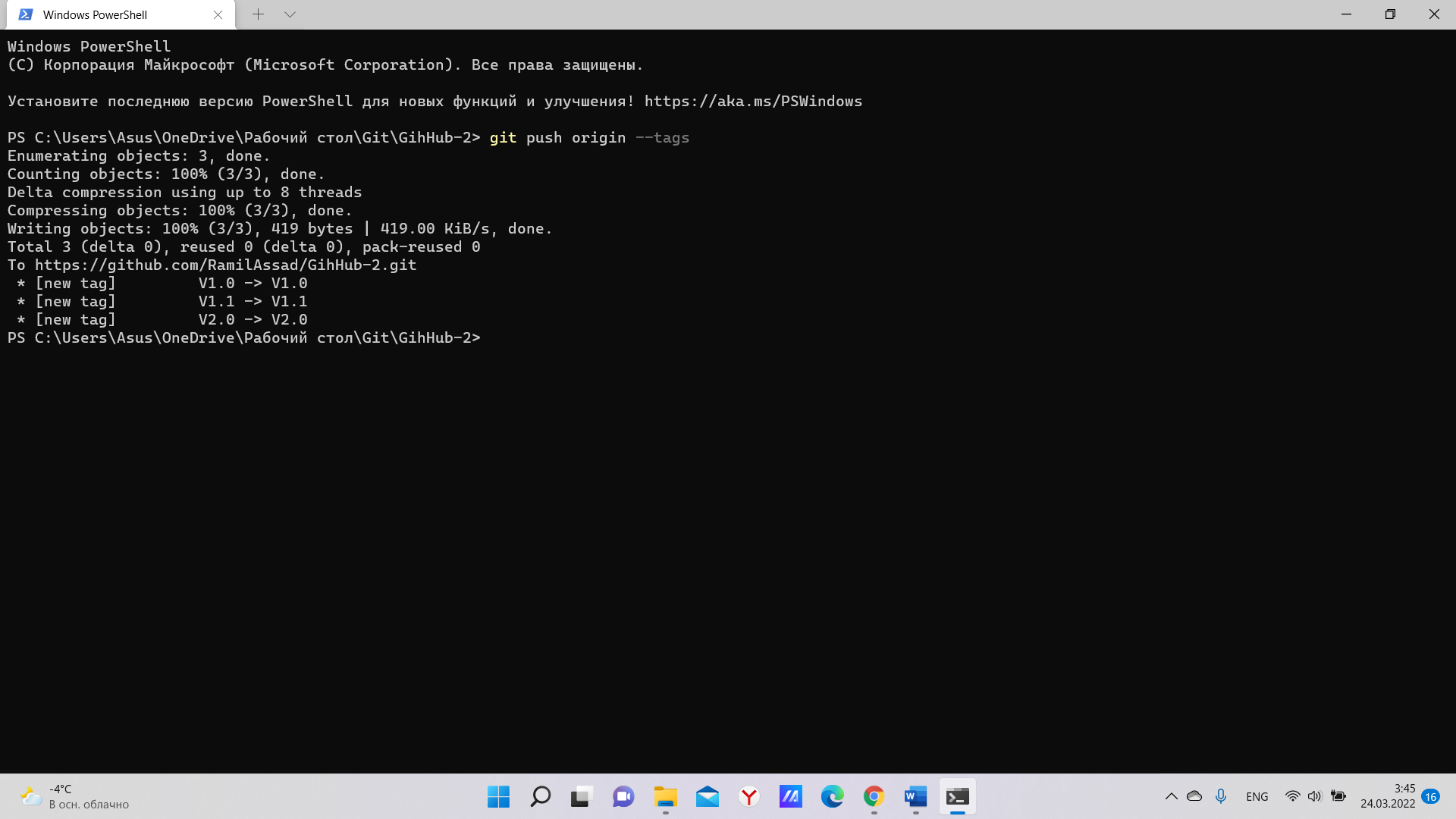


Рисунок 25. Пуш тегов

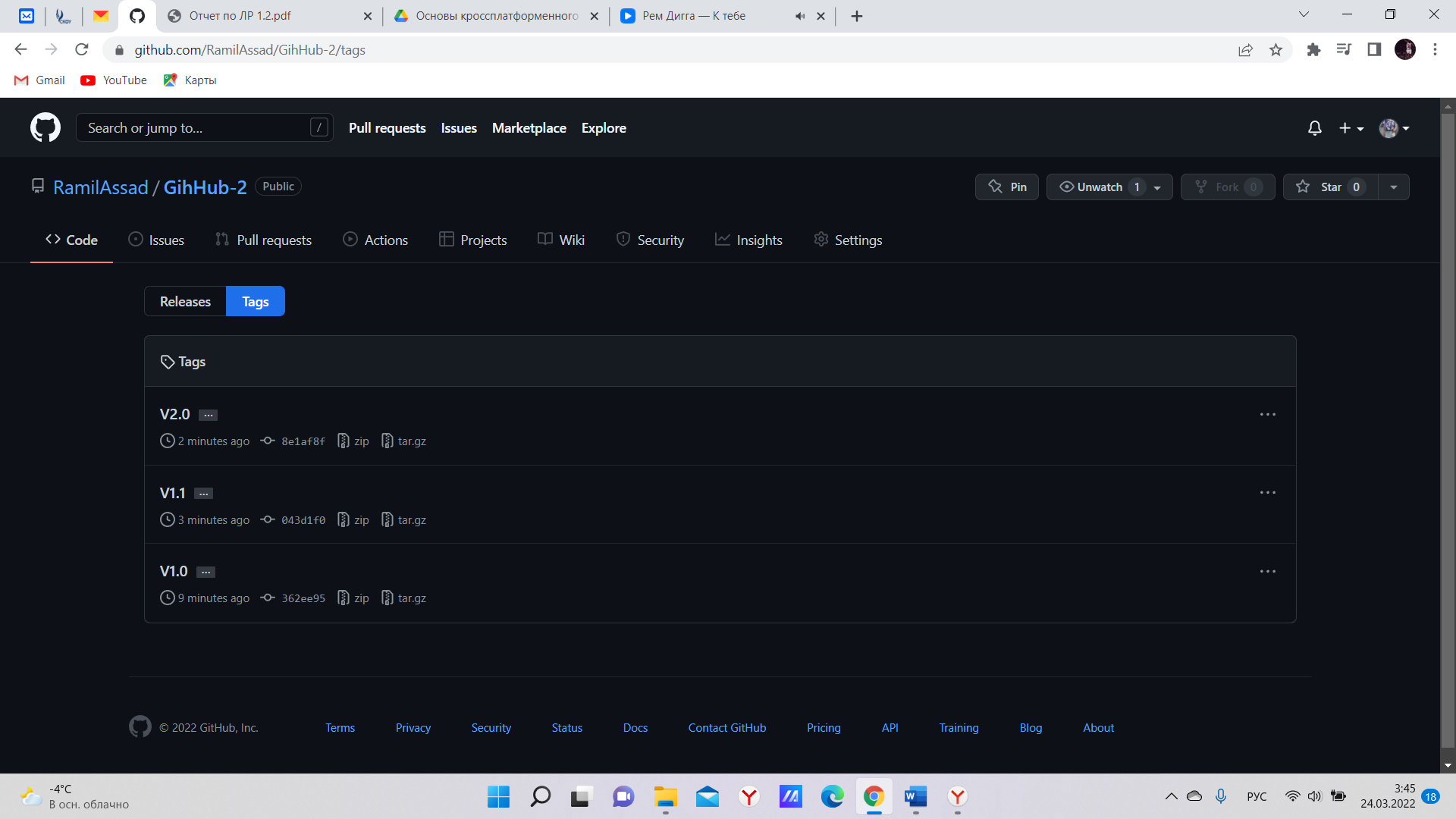


Рисунок 26. История тегов на удаленном сервере

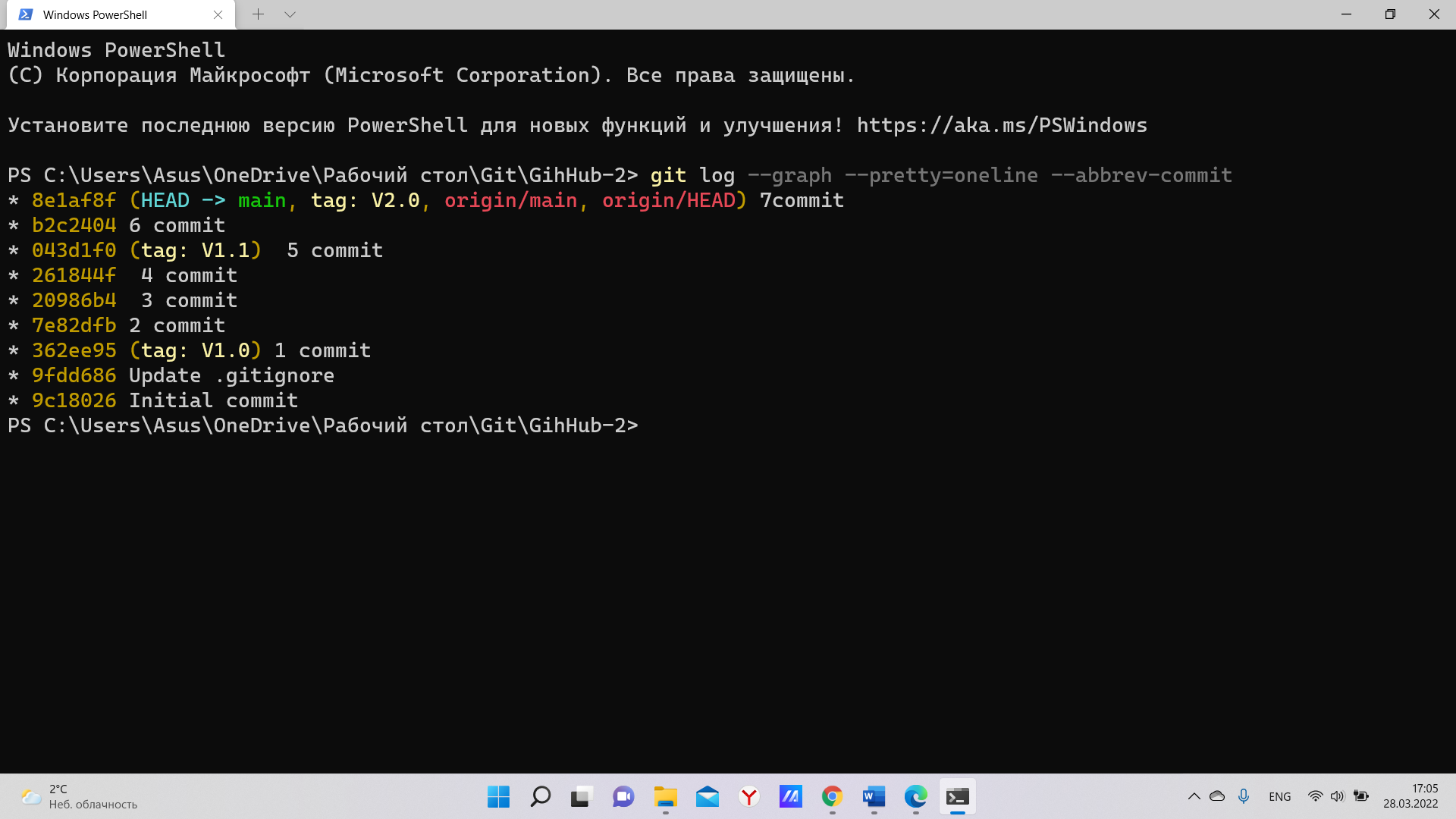


Рисунок 27. История коммитов с помощью git log

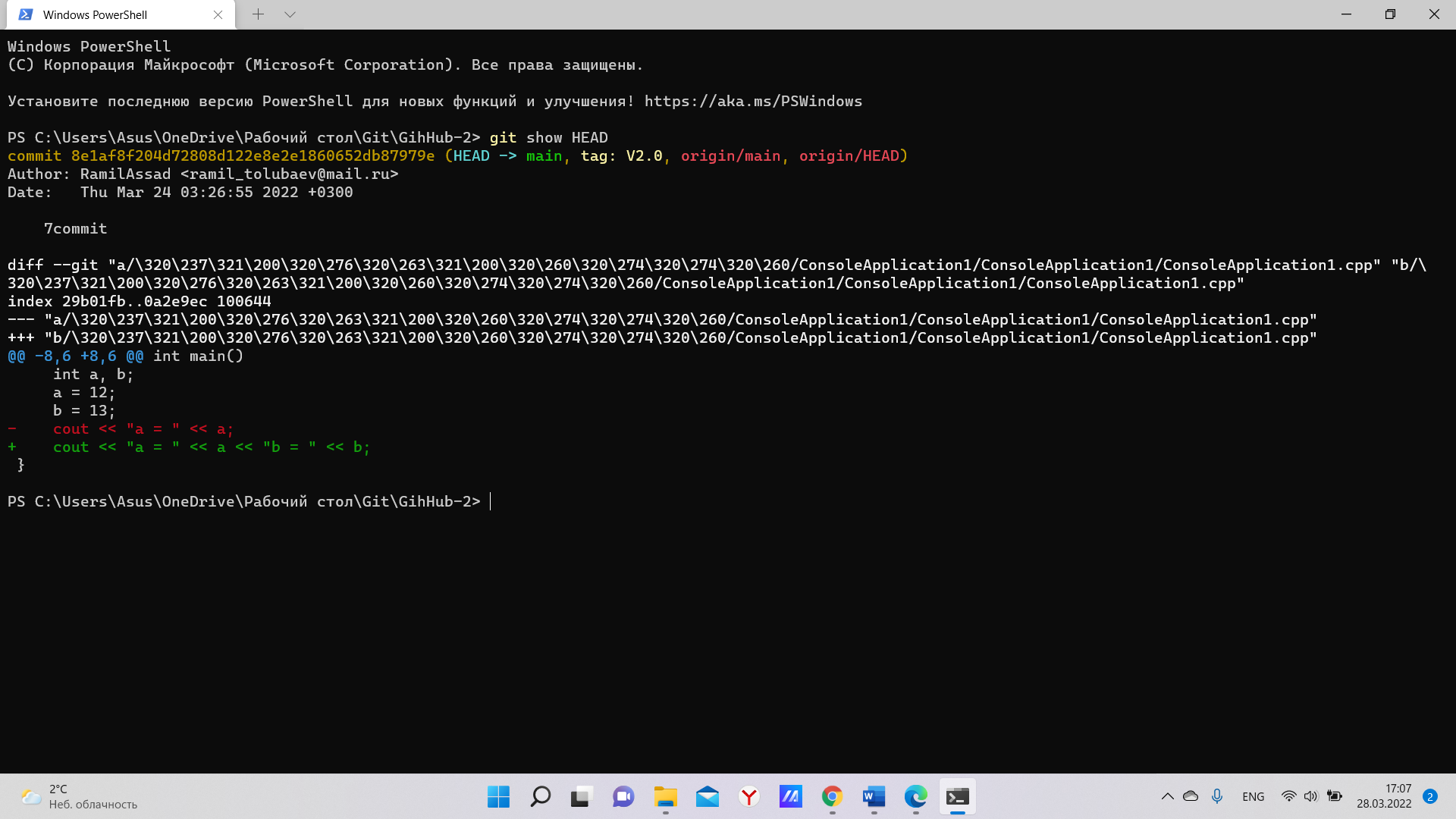


Рисунок 28. Команда git show HEAD

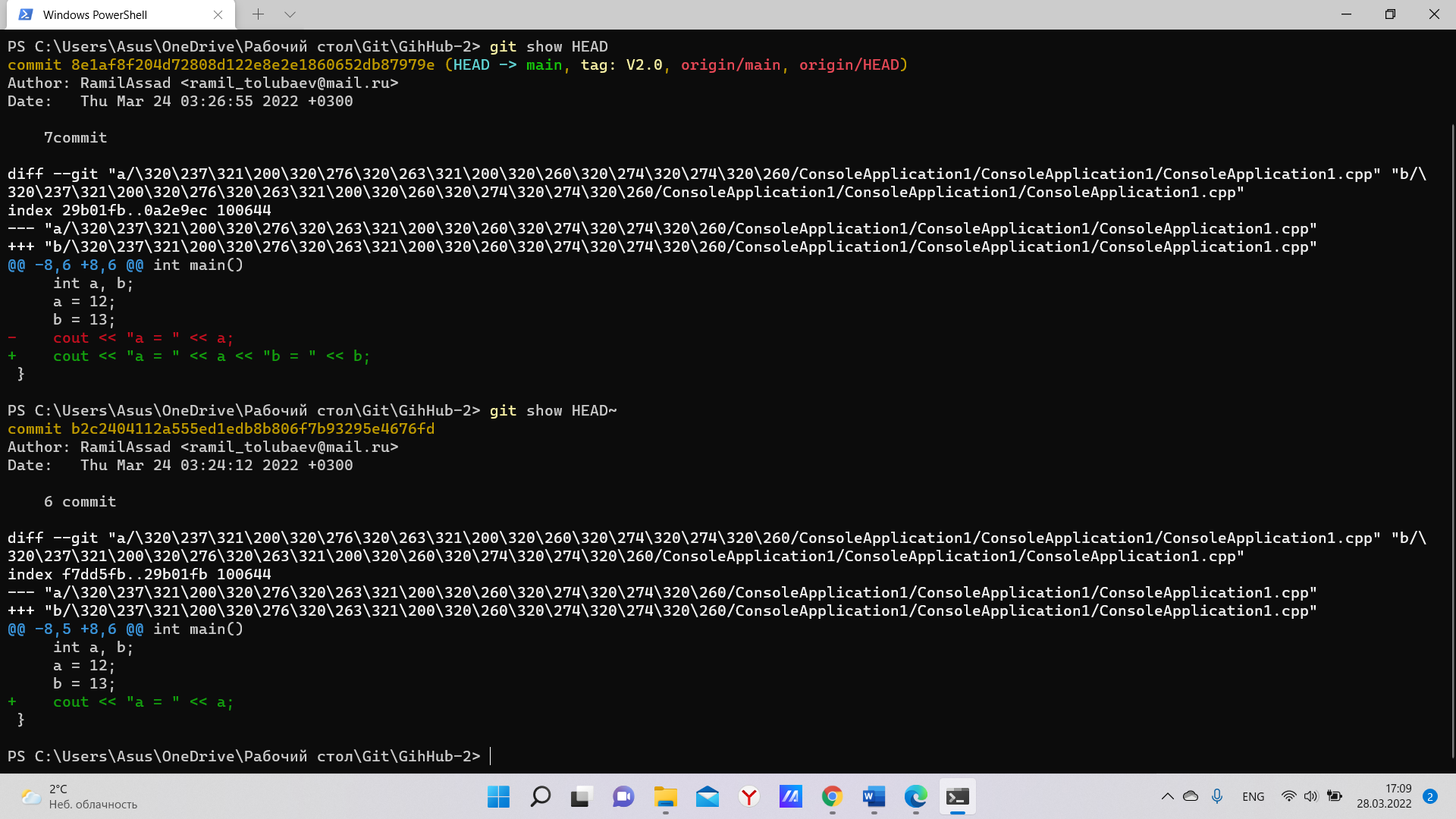


Рисунок 29. Команда git show HEAD~

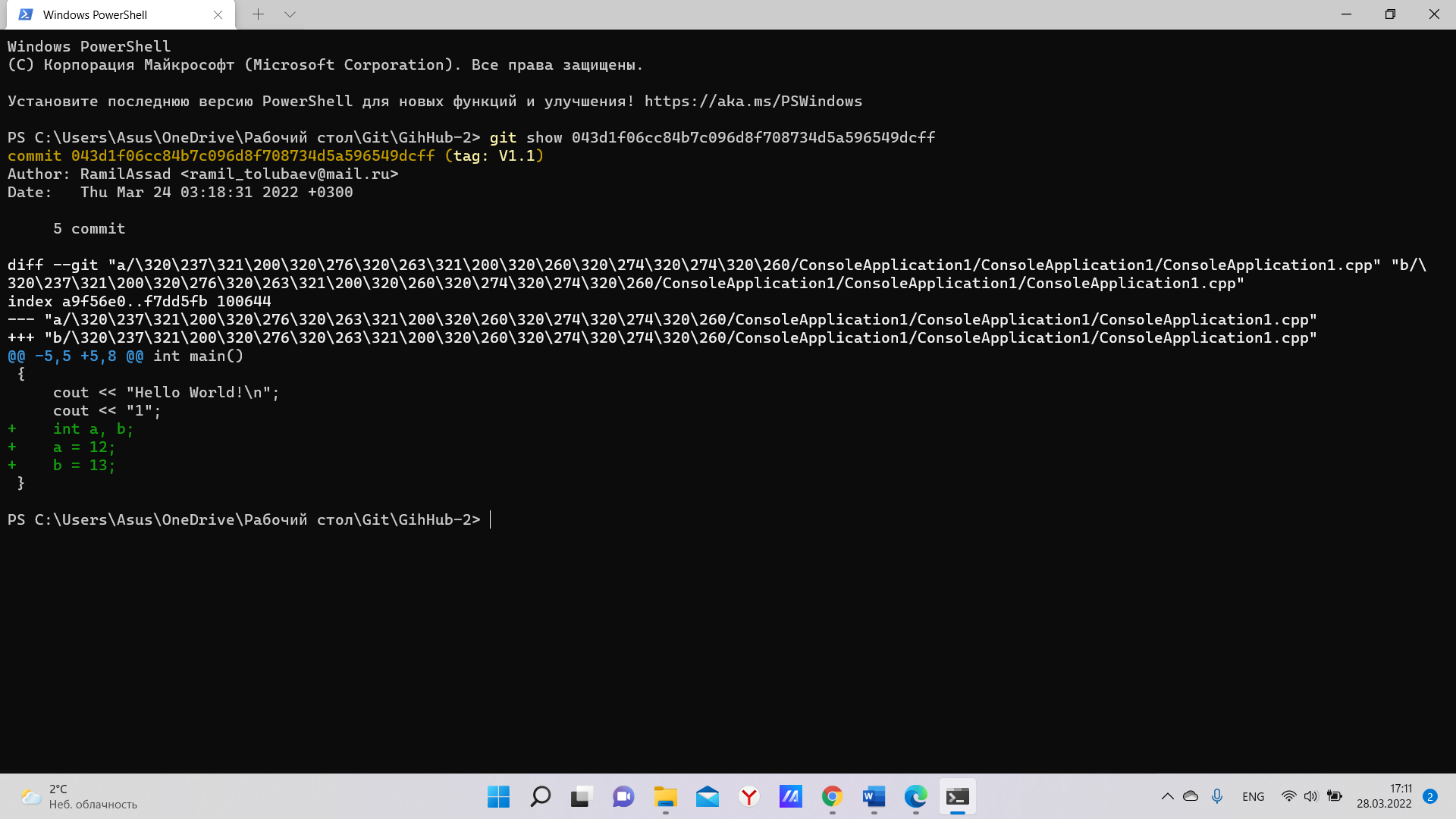


Рисунок 30. Команда git show 043d1f0

1. Откат к заданной версии

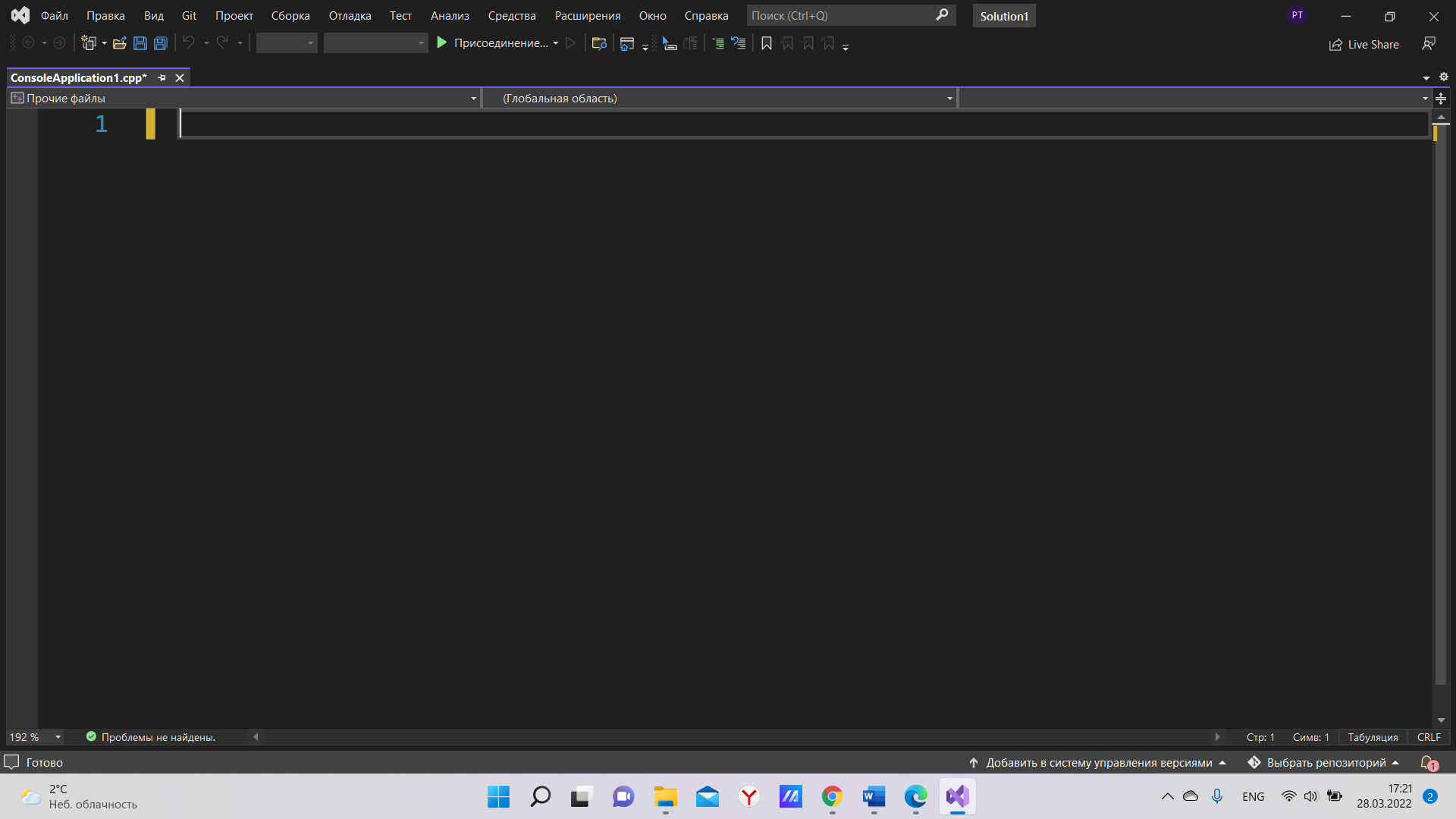


Рисунок 31. Удаление кода в файле

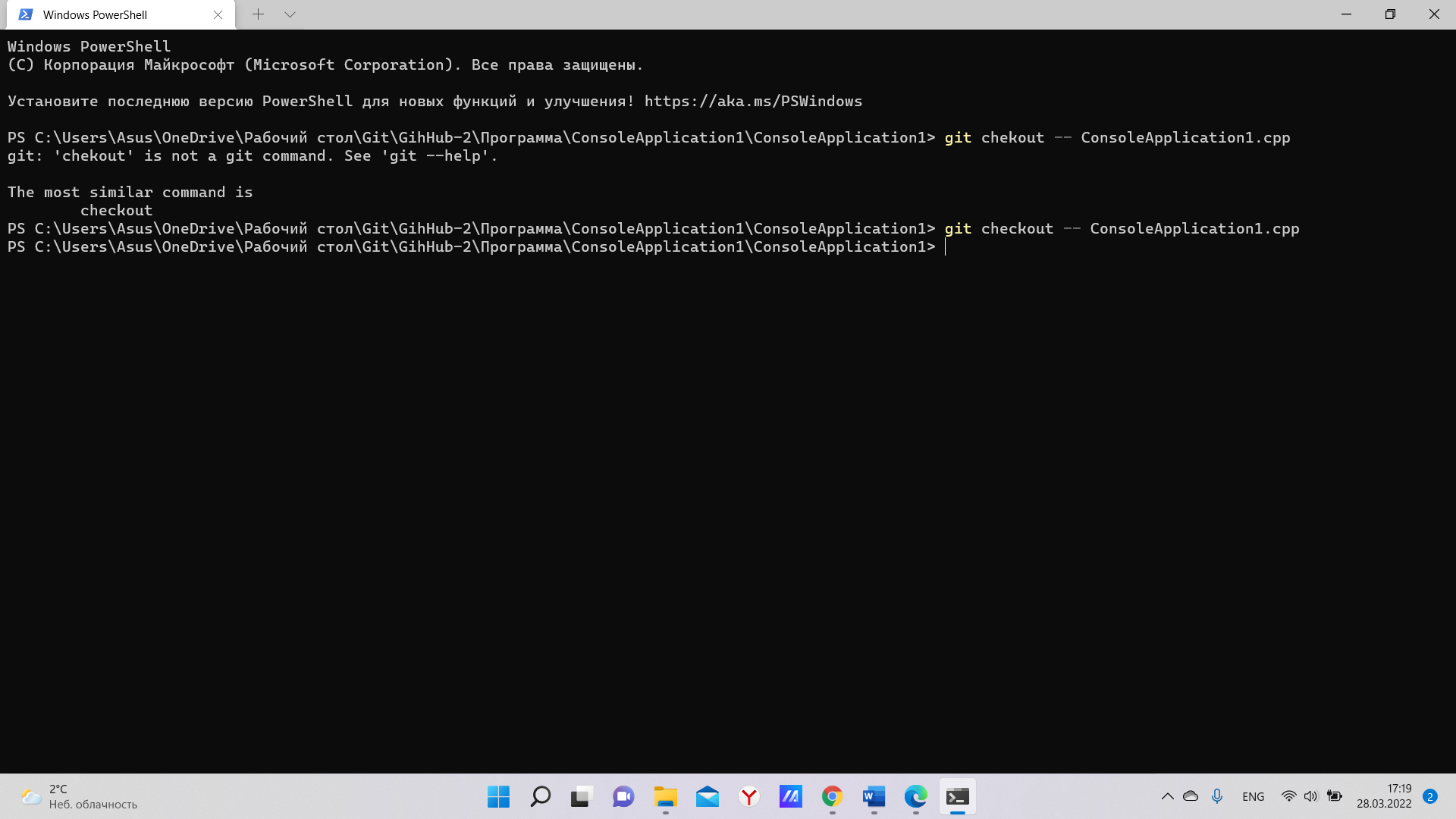


Рисунок 32. Команда checkout

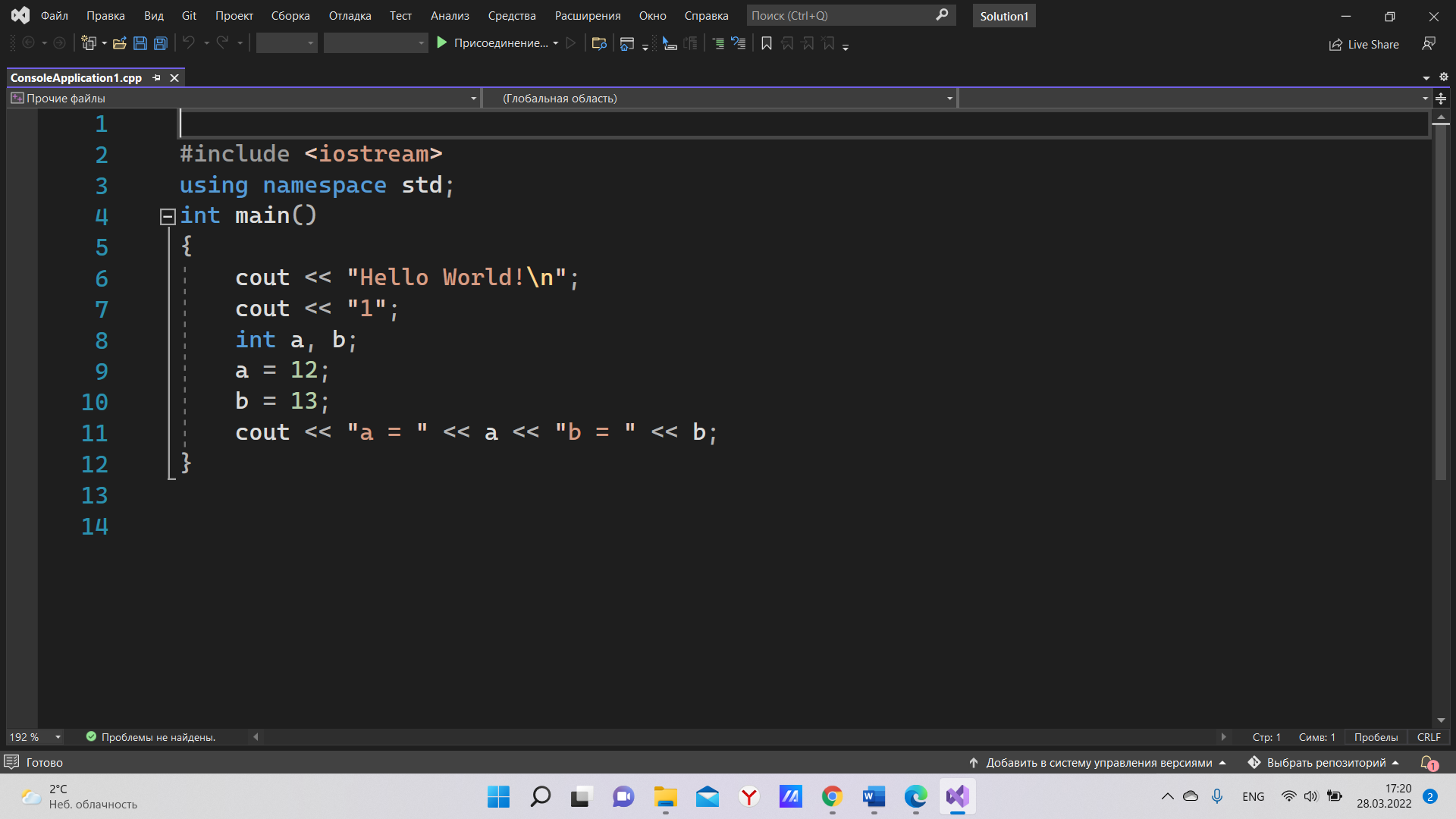


Рисунок 33. Изменение после команды checkout

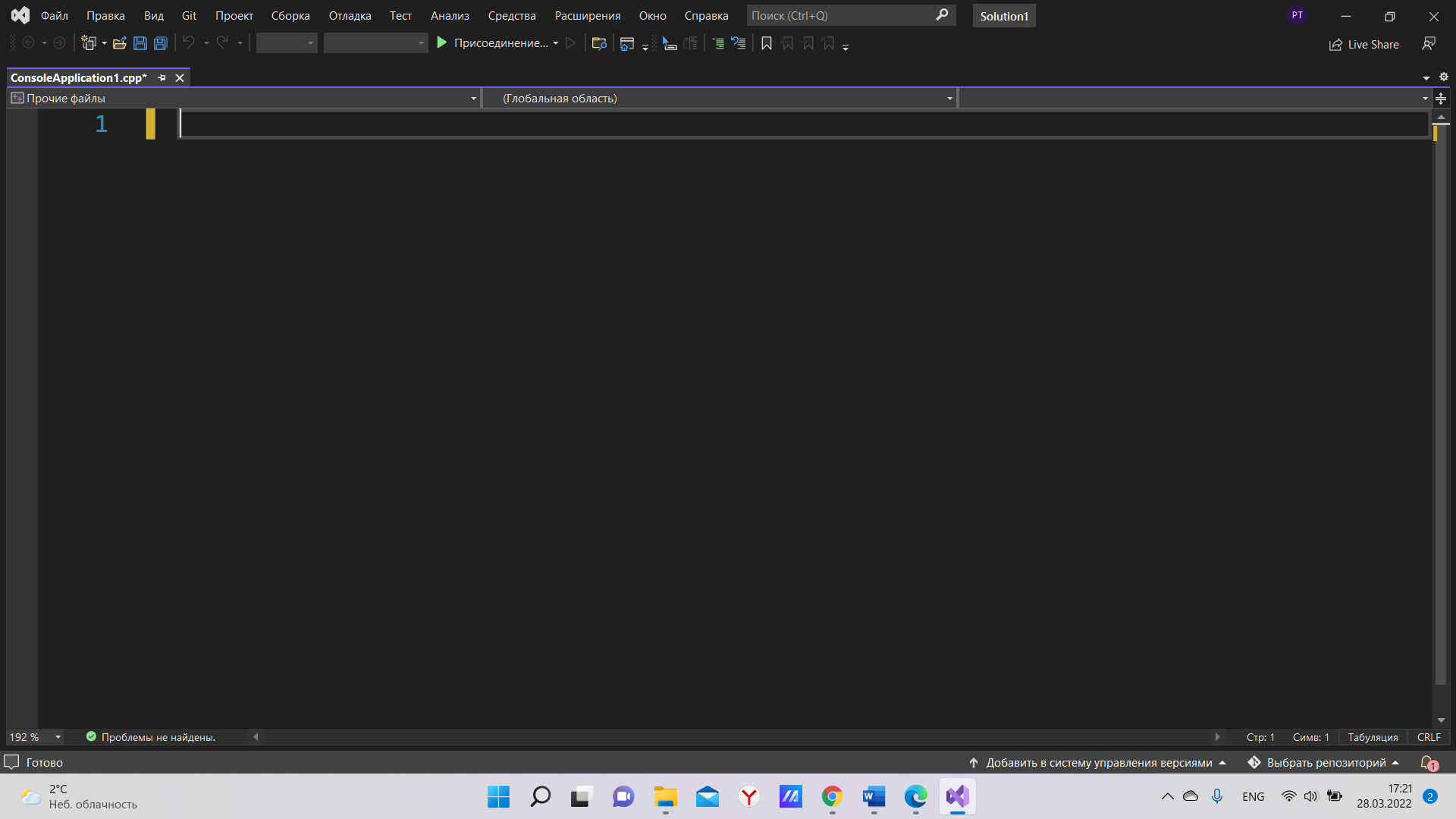


Рисунок 34. Удаление кода в файле

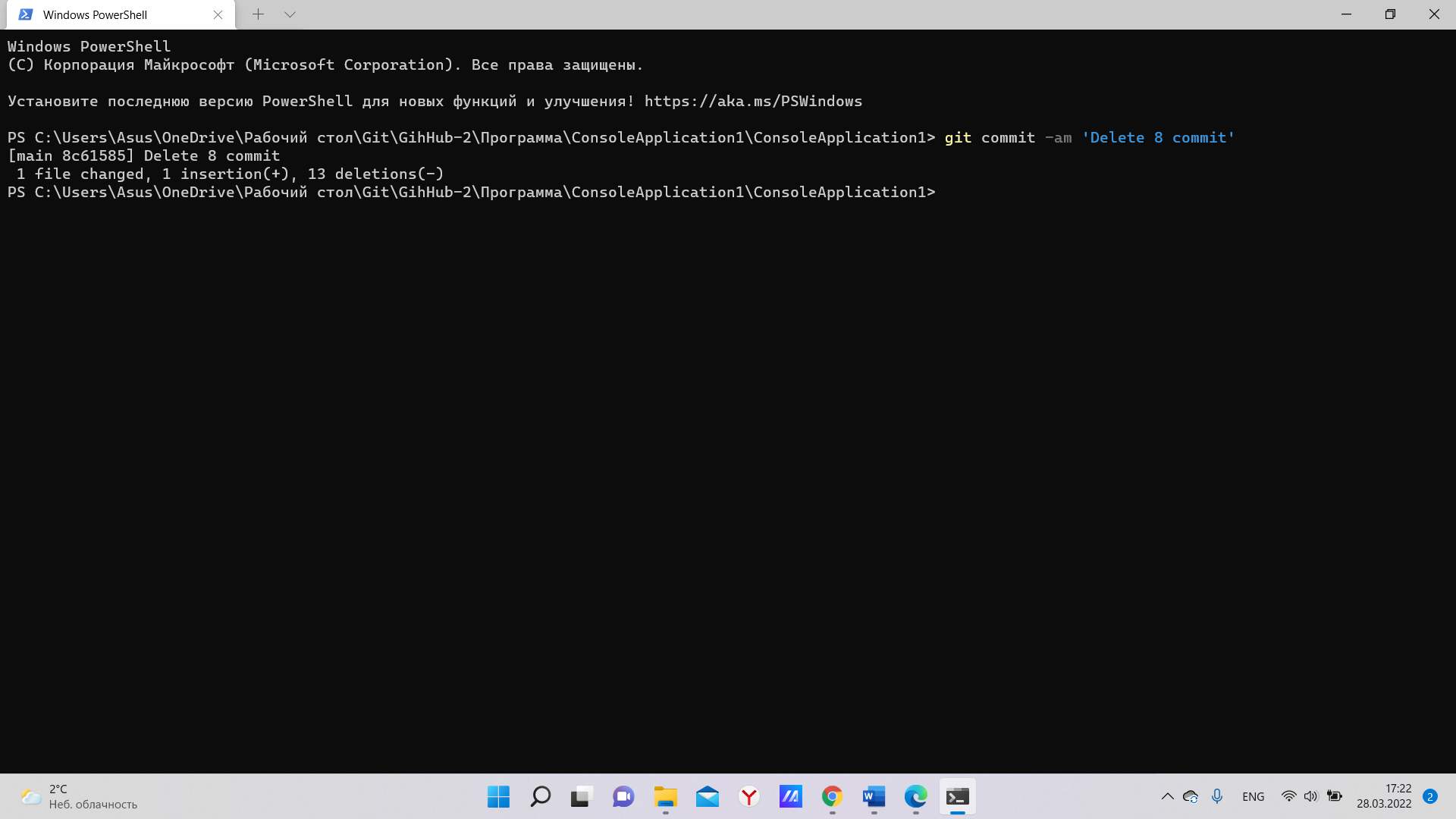


Рисунок 35. Коммит изменении

Вывод: Команда git checkout --<имя\_файла> удаляет изменения в репозитории по коммита.

**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 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 мо-жете

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 выпол-нили

предыдущий коммит, а еще раз выполнили команду git add и выпол-нили

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: используй-те

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 необходи-мо

вручную слить эти данные с вашими, когда вы будете готовы.

Если ветка настроена на отслеживание удалённой ветки, то вы може-те

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 про-стая:

git push <remote-name> <branch-name>.

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

Для просмотра удалённого репозитория, можно использовать ко-манду

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 выпол-нить

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