«Утверждаю»

Преподаватель спец дисциплины

«Технология разработки программного обеспечения»

Абдрахманова Зульфия Амуровна

«27» Марта 2025г.

**Изучение работы в системе контроля версий**

Челябинск, 2025

**Практическая работа 9 «Изучение работы в системе контроля версий»**

**Цель:** изучить на практике понятия и компоненты систем контроля версий (СКВ), приемы работы с ними. Освоить специализированное ПО и распространенный сервис для работы с распределенной СКВ Git — TortoiseGit и GitHub.com.

**Выполнил:** Студент Челябинского Радиотехнического техникума, группа ПР-273, Голощапова Анастасия

**Задания:**

1. Заполните основные данные на [https://github.com](https://github.com/).
2. Создать базовую конфигурацию для работы с git.
3. Создать ключ SSH.
4. Создать ключ PGP.
5. Настроить подписи git и автоматические подписи git.
6. Зарегистрироваться на Github.
7. Создать локальный каталог для выполнения заданий по предмету.
8. Настройка gh.

**1. Заполните основные данные на**[**https://github.com**](https://github.com/)**.**

Так как у меня уже есть аккаунт, я просто в него зашла (Рисунок 1).

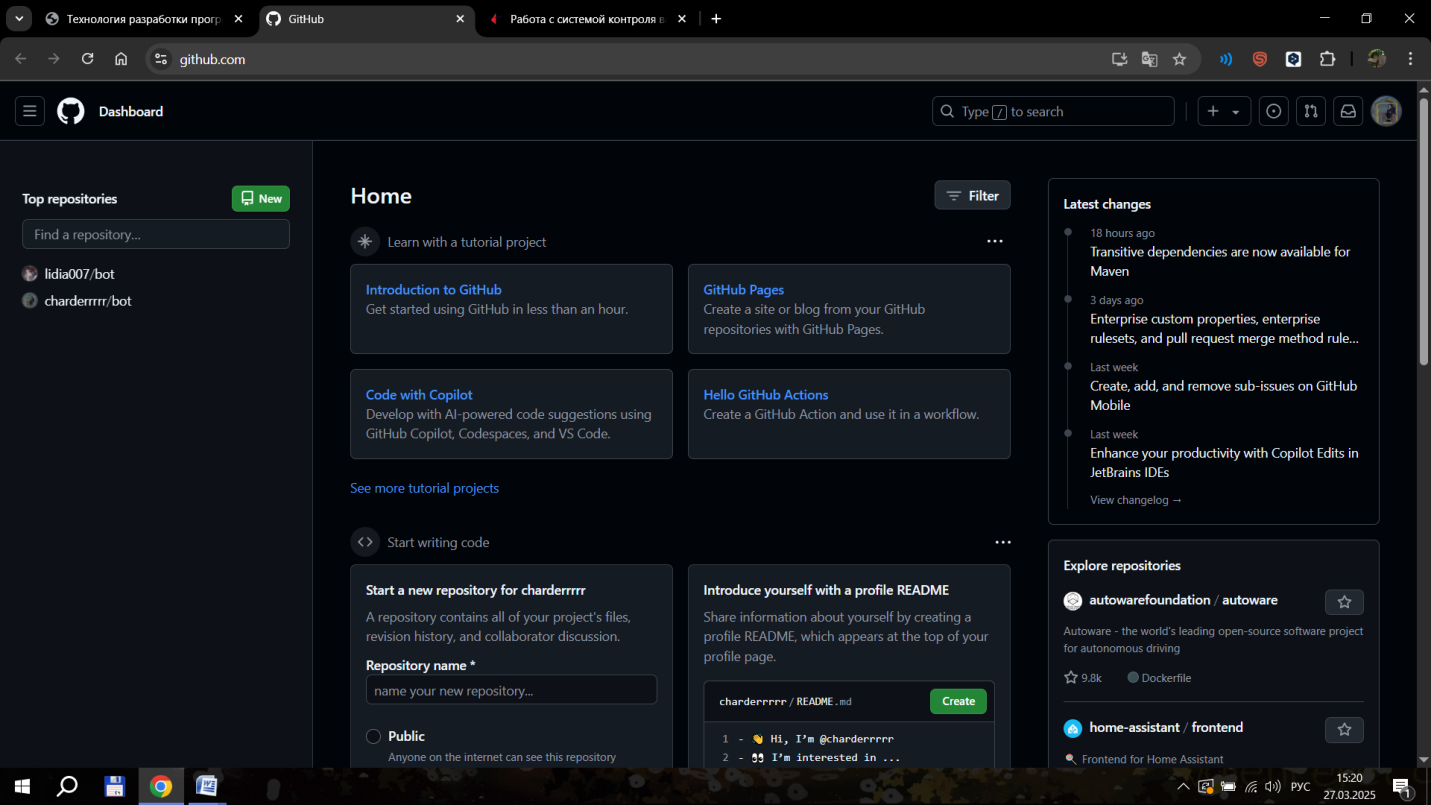


Рисунок 1 - Главная страница

**2. Создать базовую конфигурацию для работы с git.**

Поскольку я выполняю практическую работу на своем рабочем ноутбуке, где уже настроен GitHub, то базовую конфигурацию настраивать снова мне не нужно. Базовая конфигурация представлена ниже (Рисунок 2).

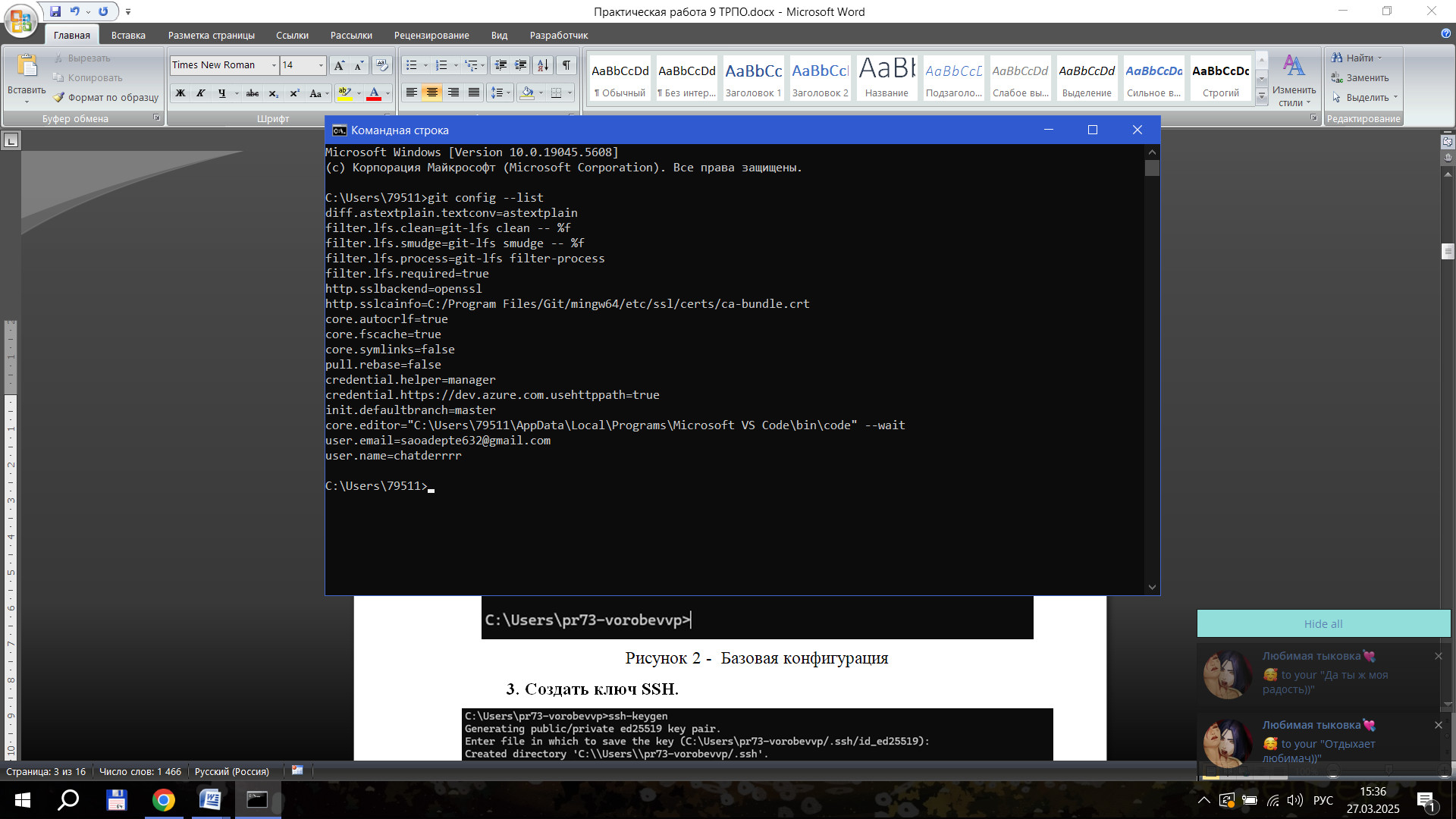


Рисунок 2 - Базовая конфигурация

**3. Создать ключ SSH.**

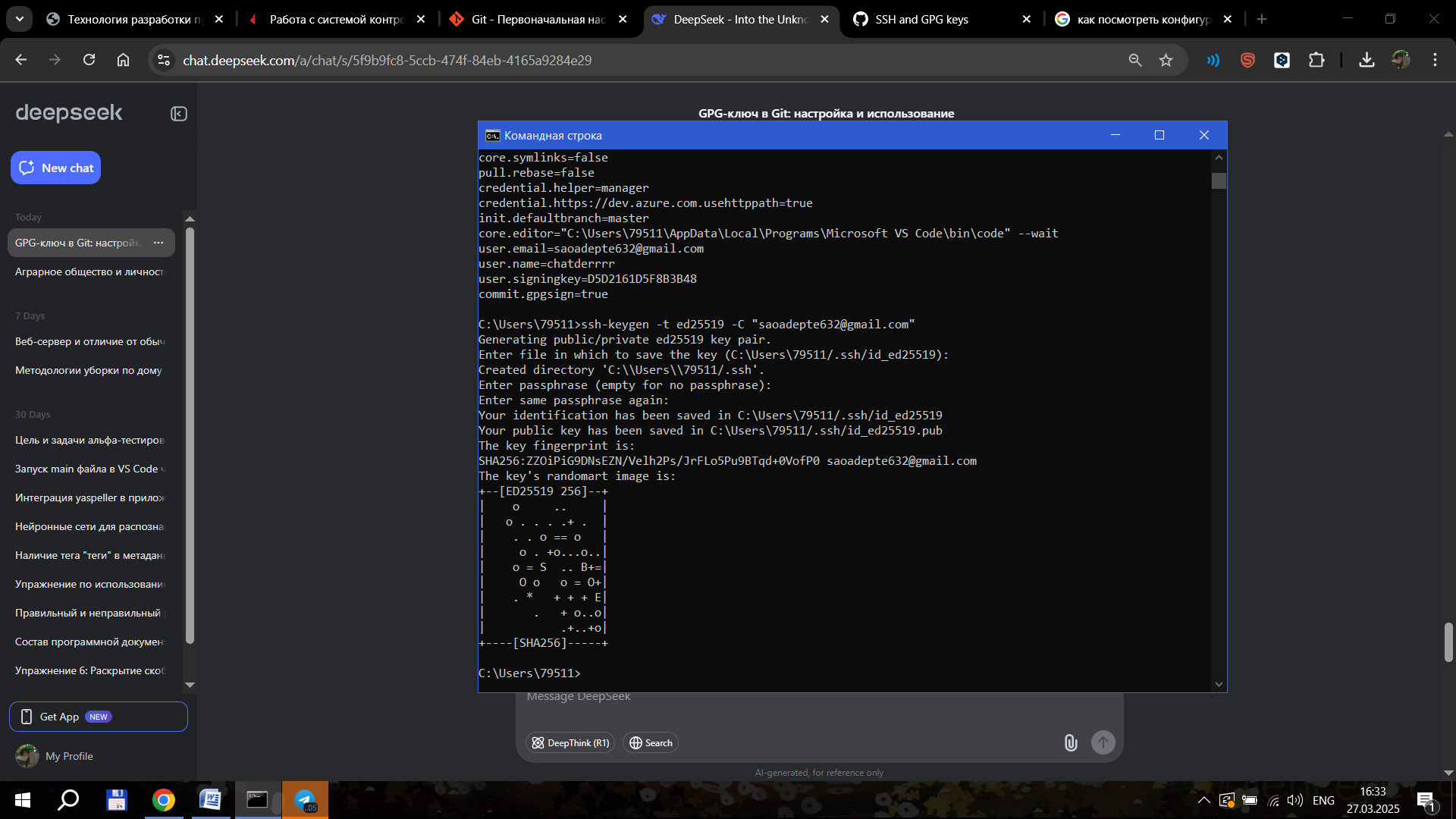


Рисунок 3 - Создание ключа SSH

**4. Создать ключ PGP.**

Имеющийся у меня ключ представлен ниже (Рисунок 4).

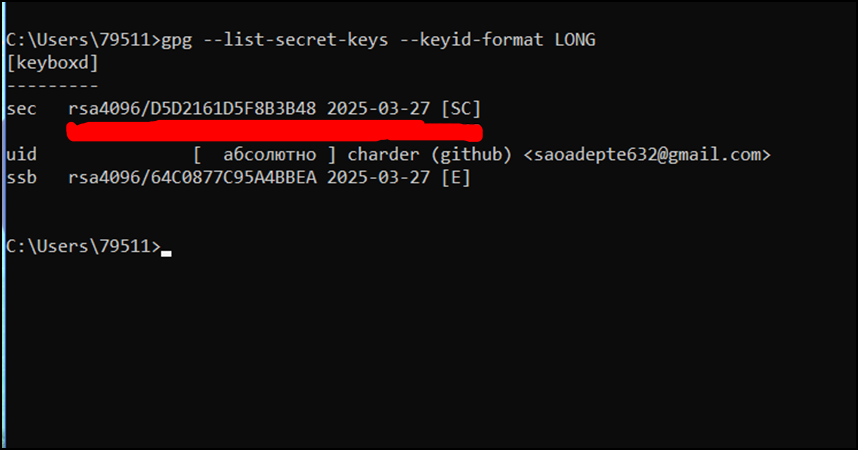


Рисунок 4 – Ошибка

**5. Настроить подписи git.**

Для начала необходимо получить публичный ключ и скопировать его (Рисунок 5).

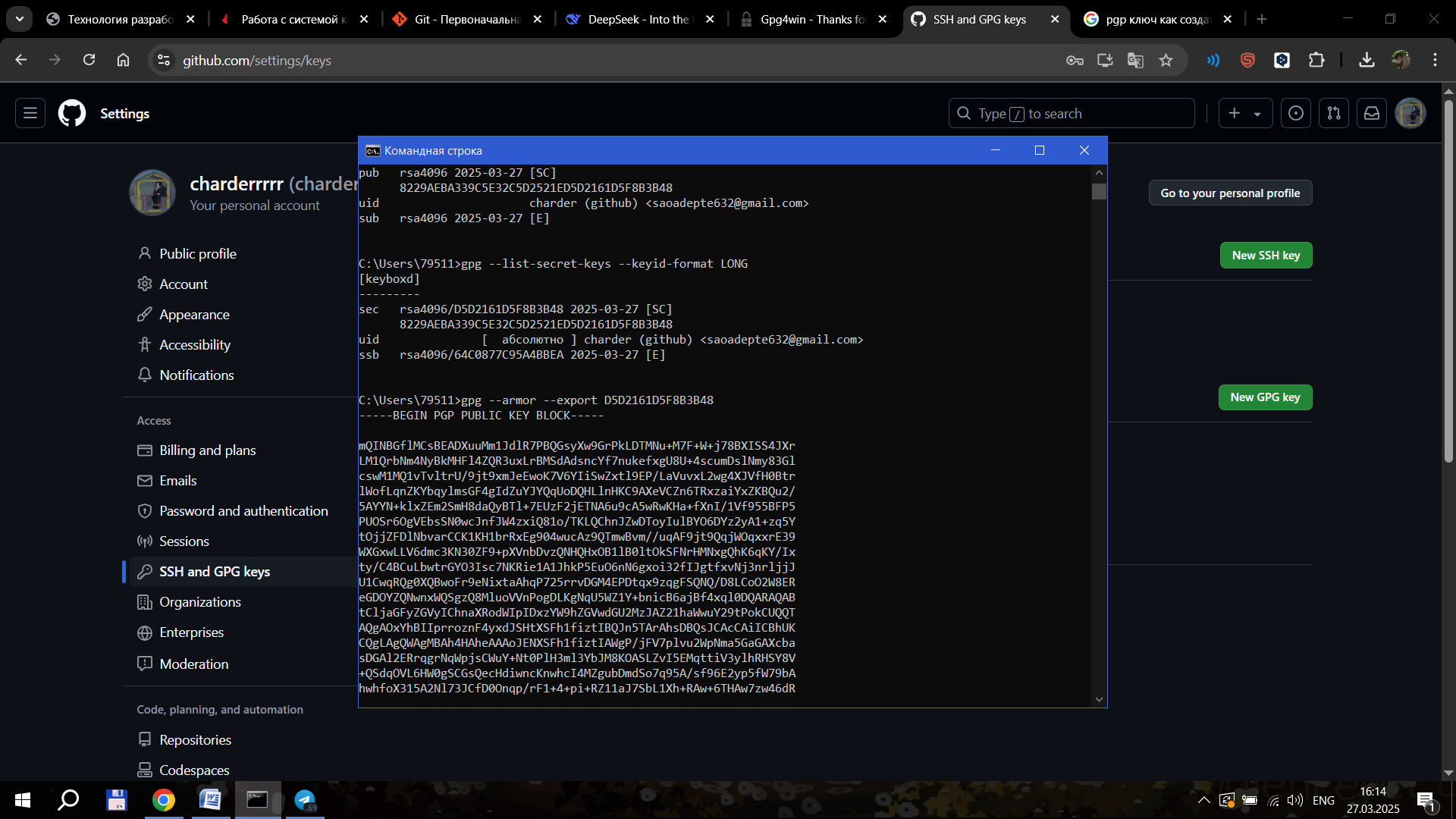


Рисунок 5 – публичный ключ.

Затем возвращаемся на GitHub и переходим в разделы Setting – SSH and GPG keys и нажимаем на кнопку New GPG key (Рисунок 6).



Рисунок 6 – добавление GPG ключа.

Вставляем заранее скопированный ключ и добавляем его (Рисунок 7).

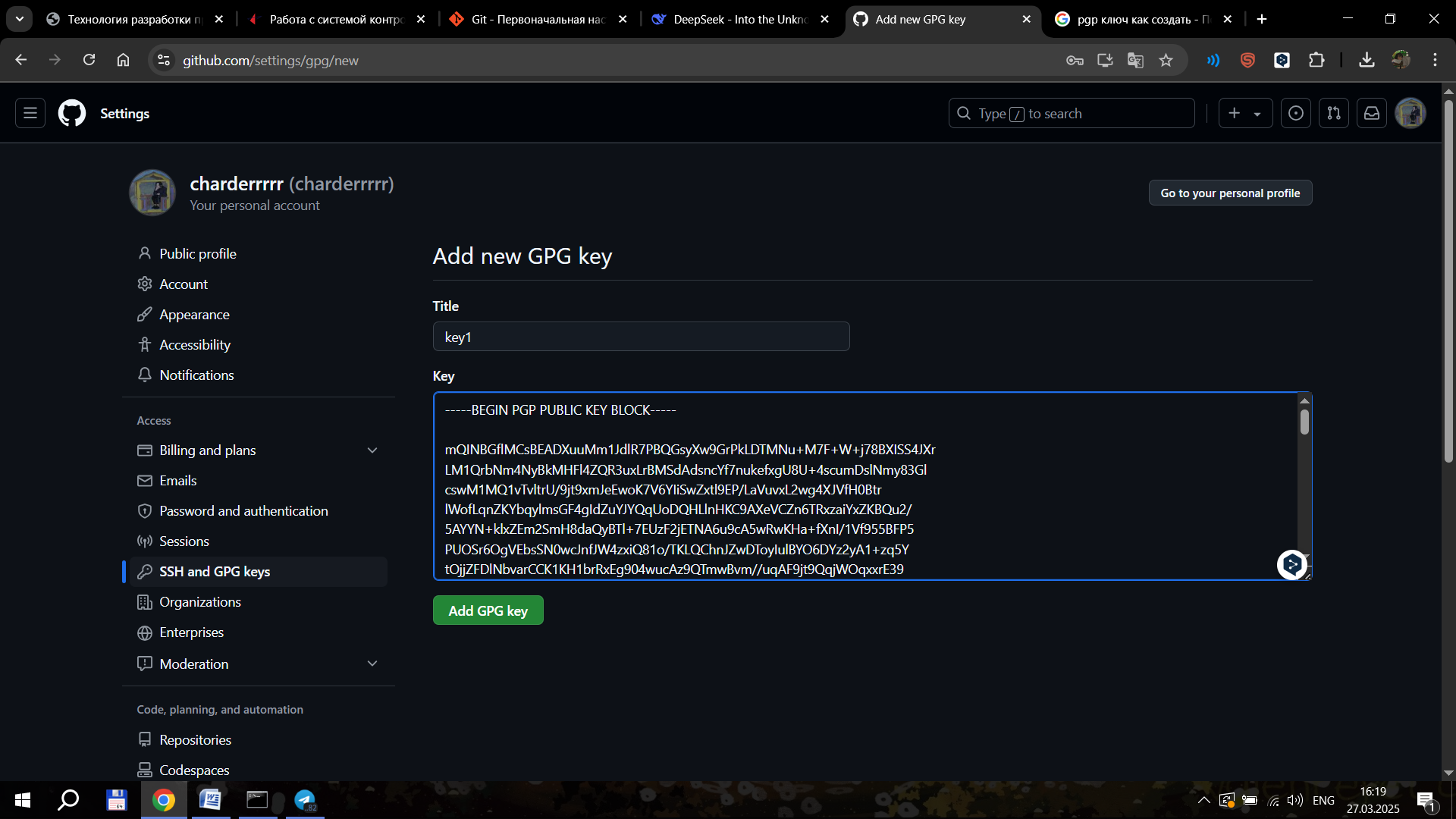


Рисунок 7 – добавление GPG ключа.

Теперь на сайте отображается добавленный ключ.

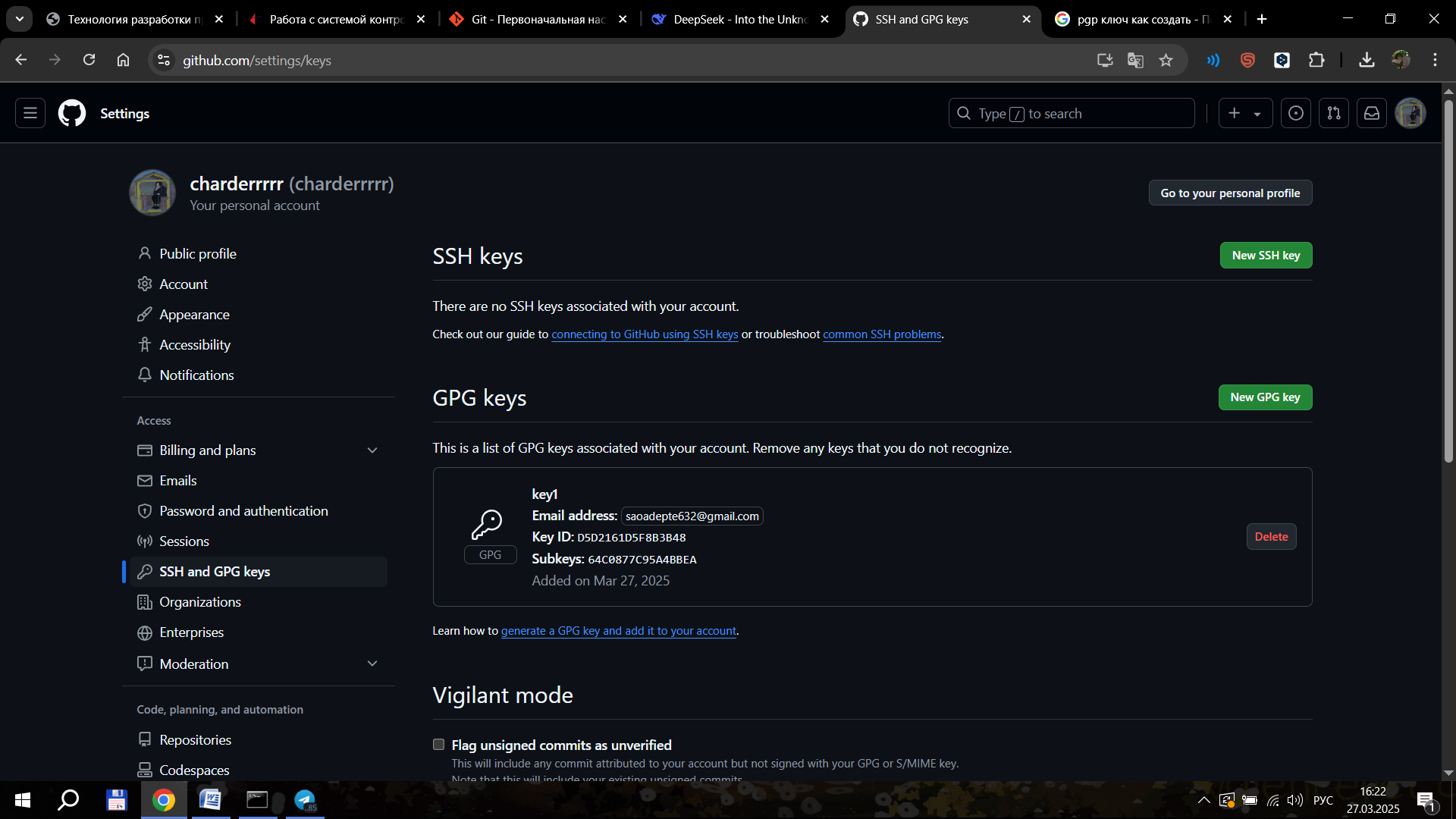


Рисунок 8 – добавленный ключ.

Переходим в командную строку, вводим команды «git config --global user.signingkey *ваш\_код*» для указания ключа и «git config --global commit.gpgsign true» для подписи всех коммитов. Результаты представлены на скриншоте (Рисунок 9).

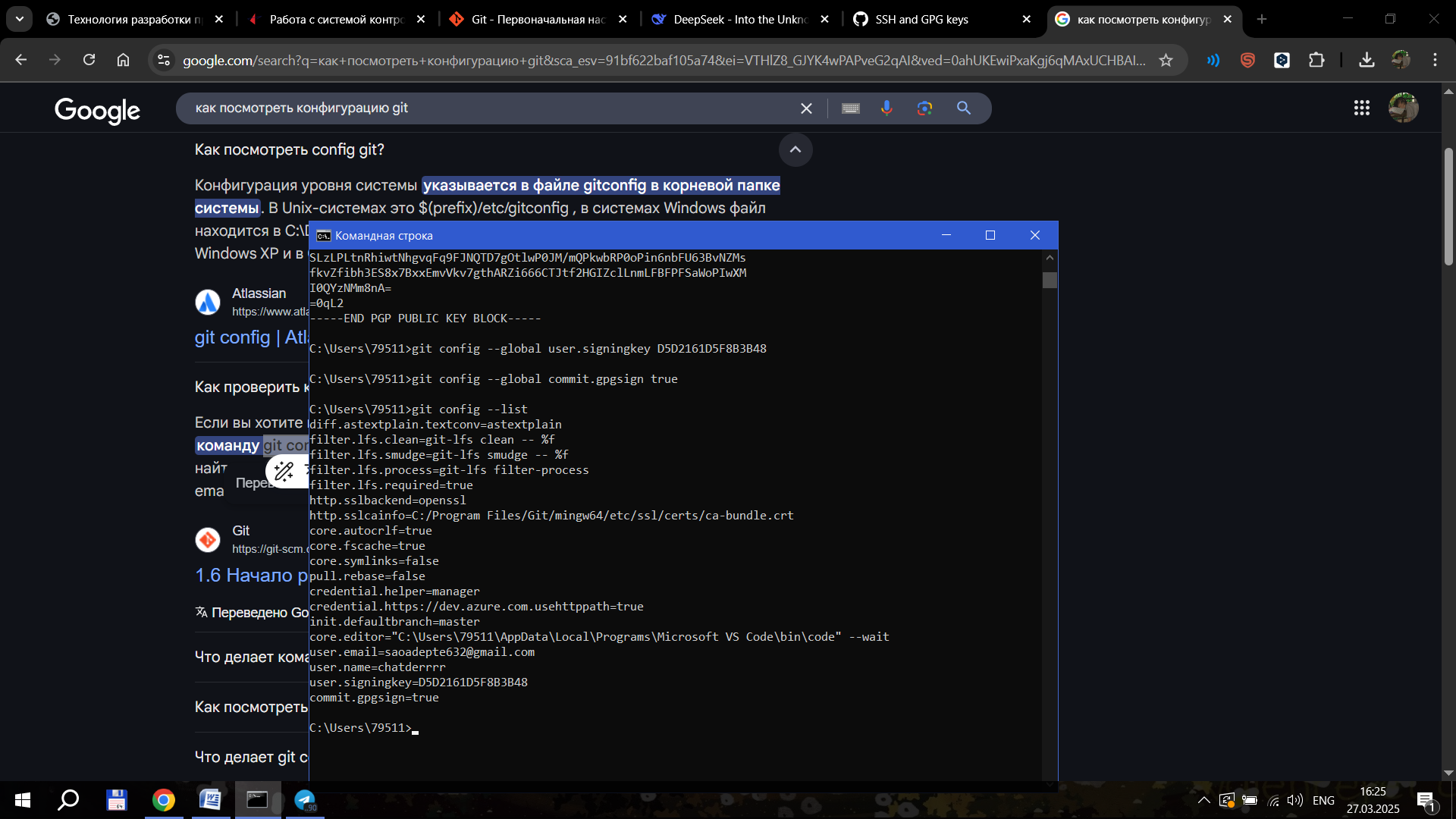


Рисунок 9 – добавление GPG ключа.

**6. Зарегистрироваться на Github.**

Так как у меня уже есть аккаунт, я просто в него зашла (Рисунок 9).

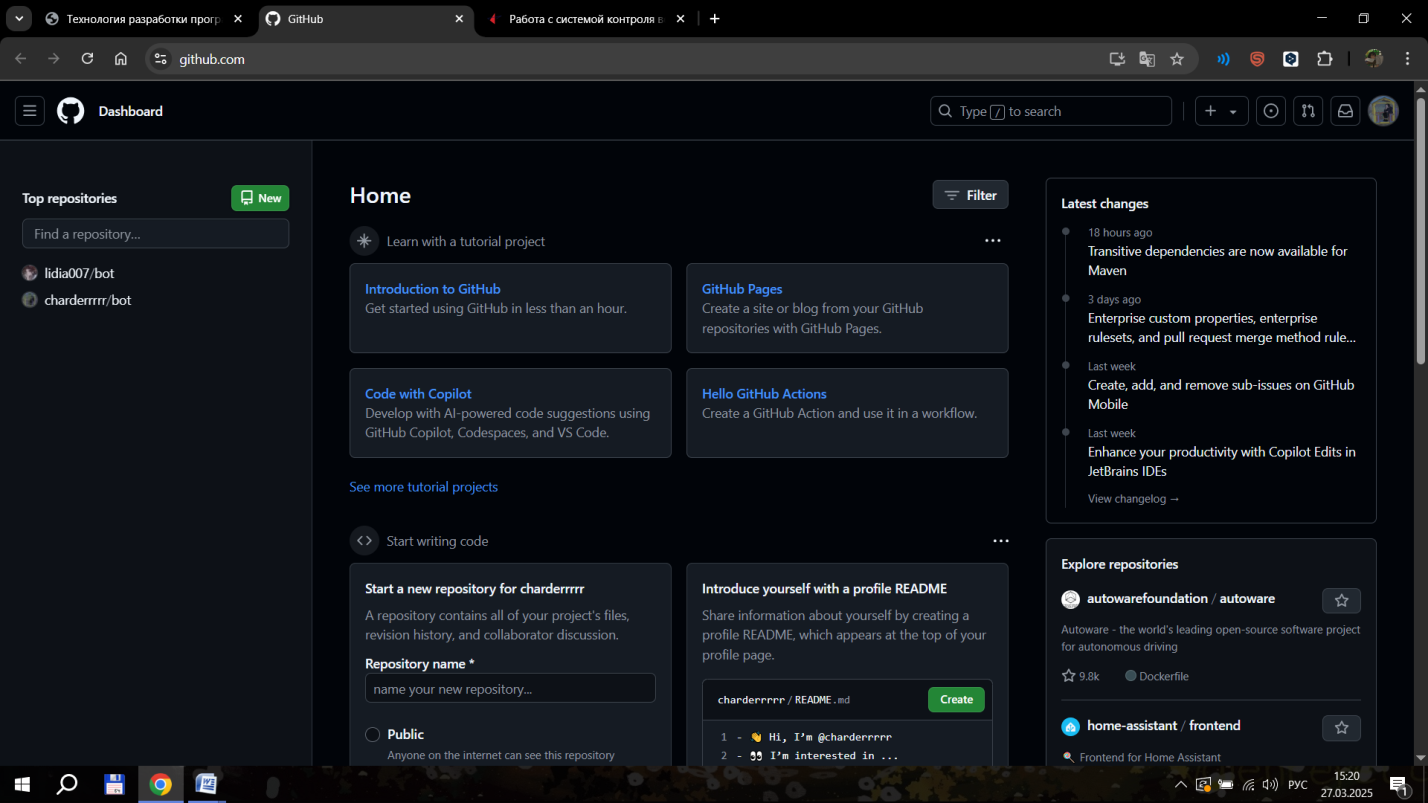


Рисунок 9 - Главная страница

**7. Создать локальный каталог для выполнения заданий по предмету.**

Выбираем кнопку «New» (Рисунок 10)

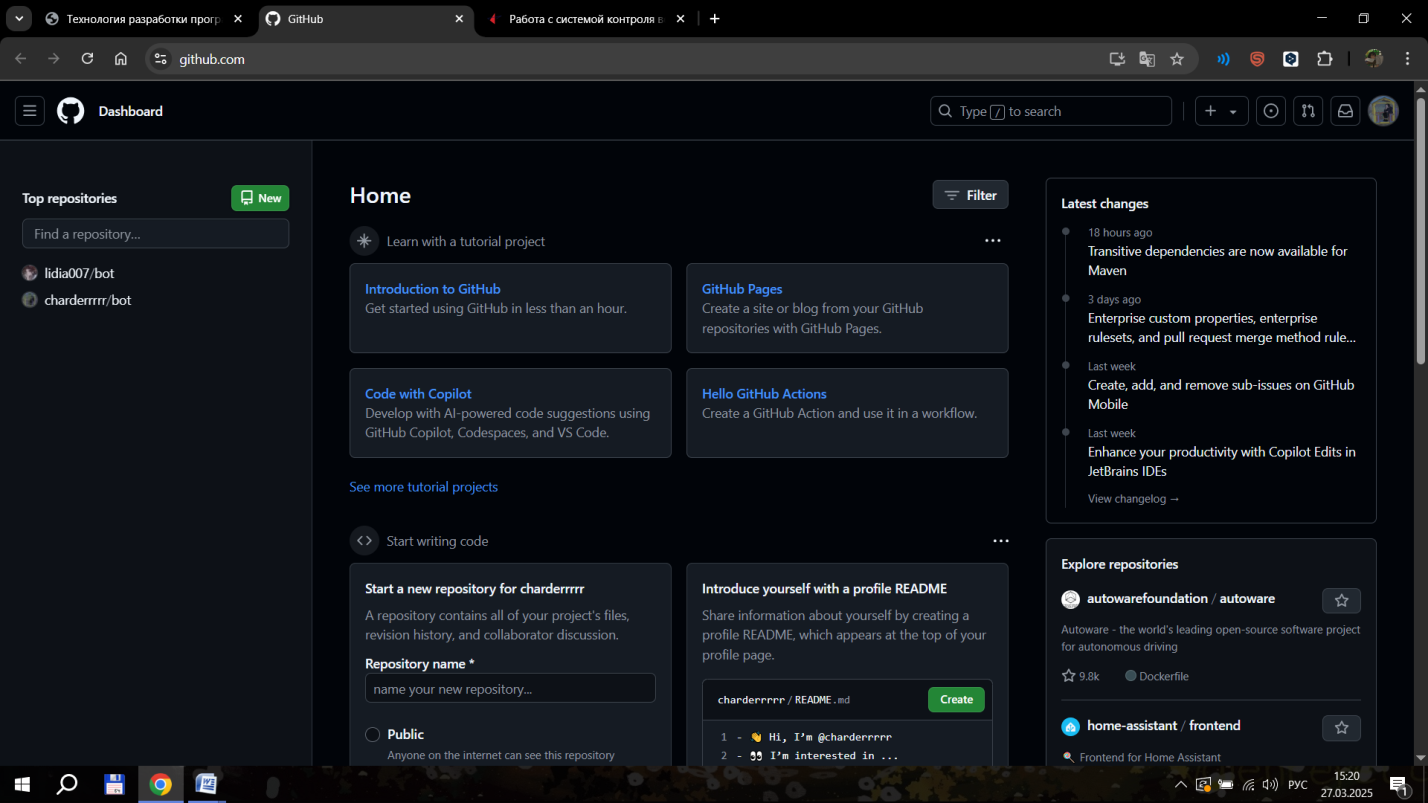
****

Рисунок 10 – главная страница

Заполняем основные данные, после нажимаем на «Create repository» (Рисунок 11).

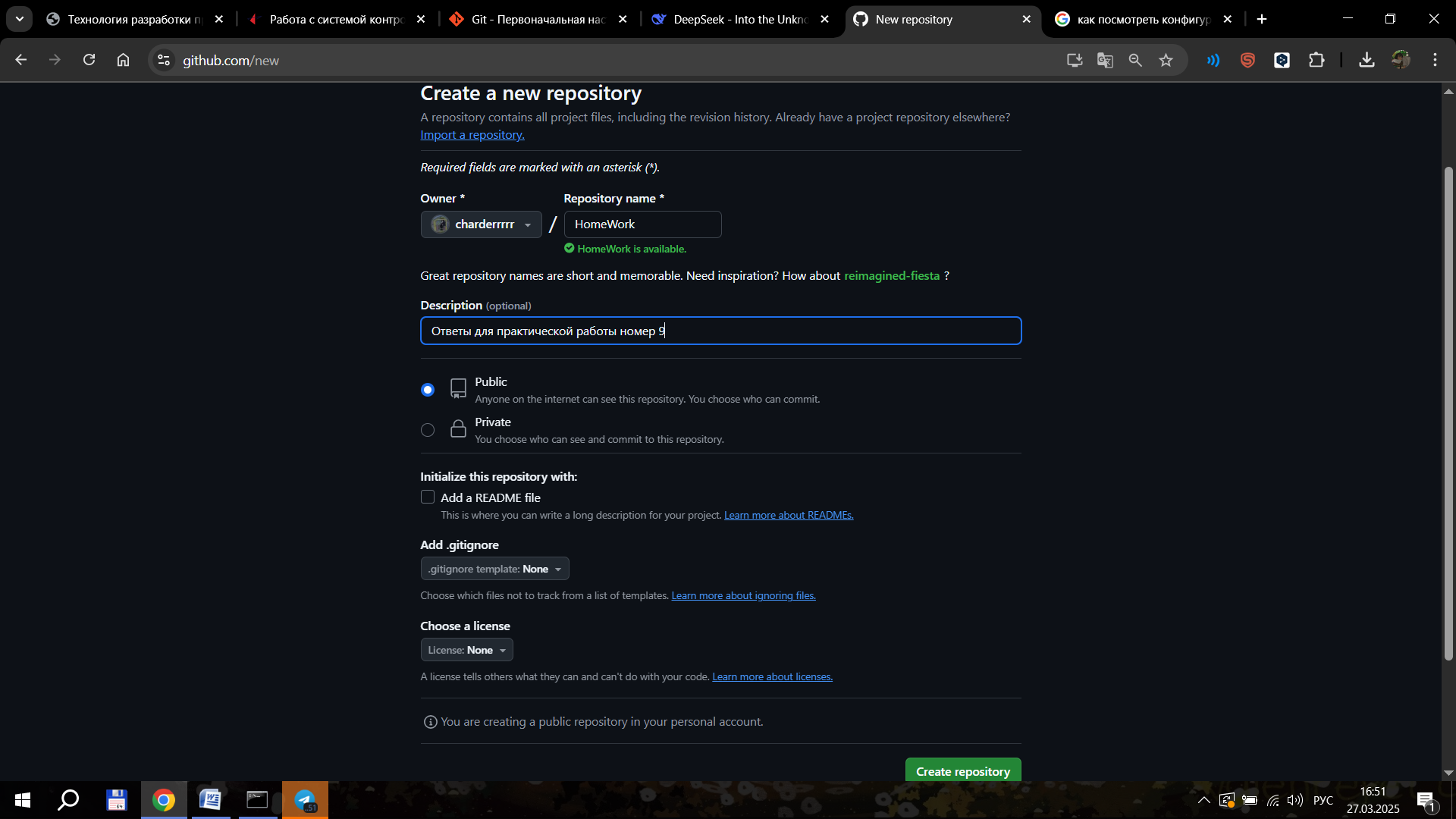


Рисунок 11 – заполнение информации о репозитории.

Отображение созданного репозитория (Рисунок 12)

https://github.com/charderrrrr/HomeWork

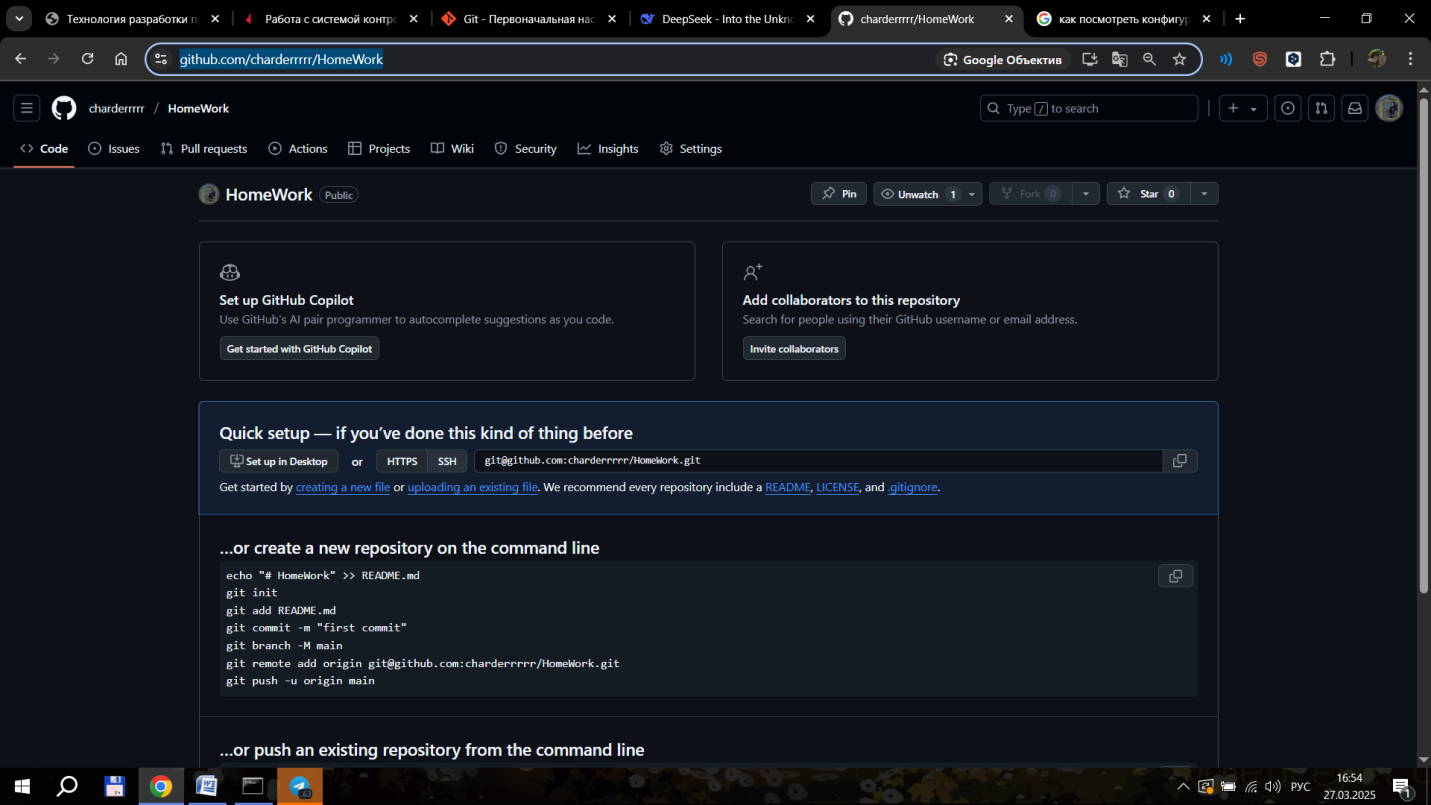


Рисунок 12 – созданный репозиторий.

**8. Настроить gh.**

Переходим на сайт <https://cli.github.com/> и нажимаем кнопку «Download from Windows» (Рисунок 13).

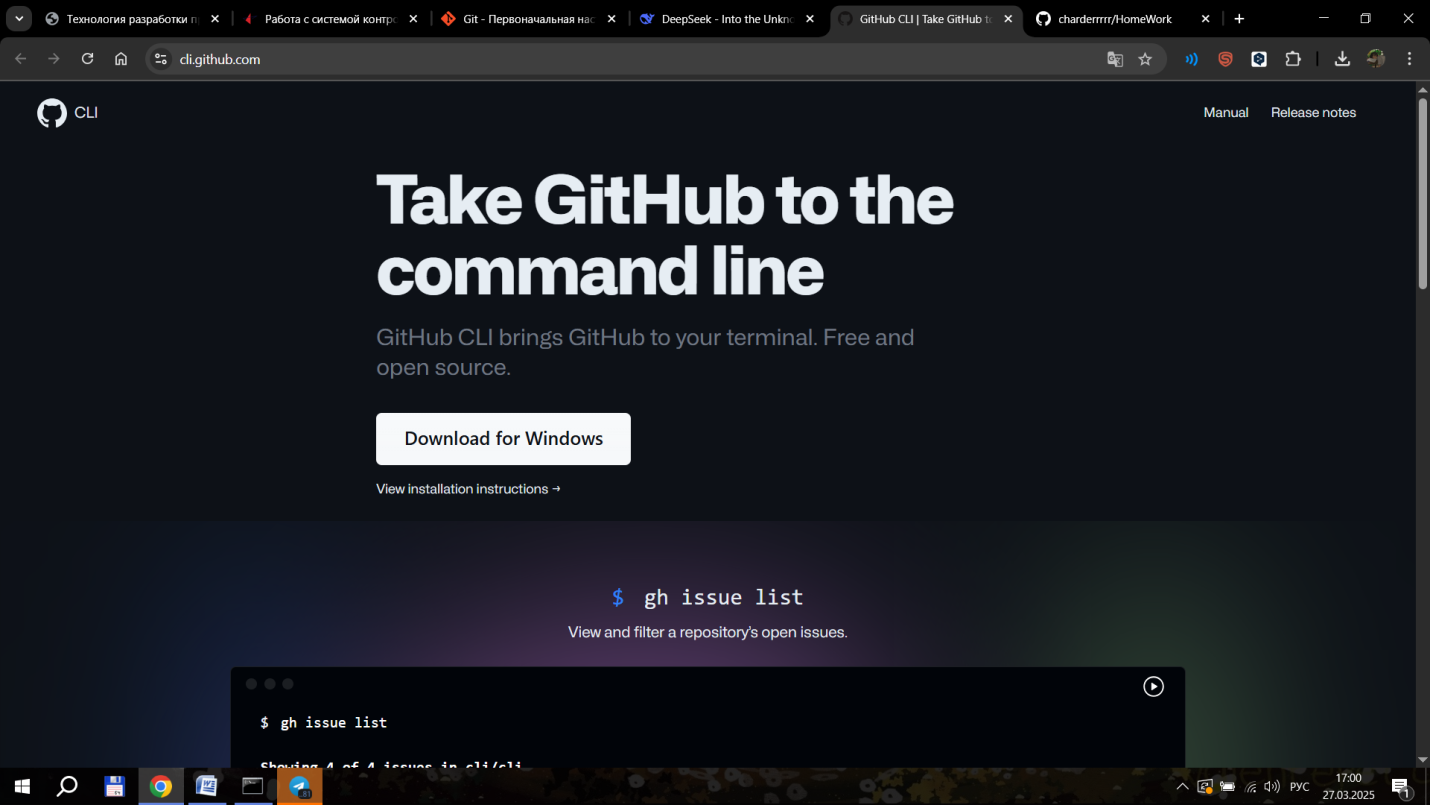
****

Рисунок 13 – сайт с установщиком.

Устанавливаем (Рисунок 14).

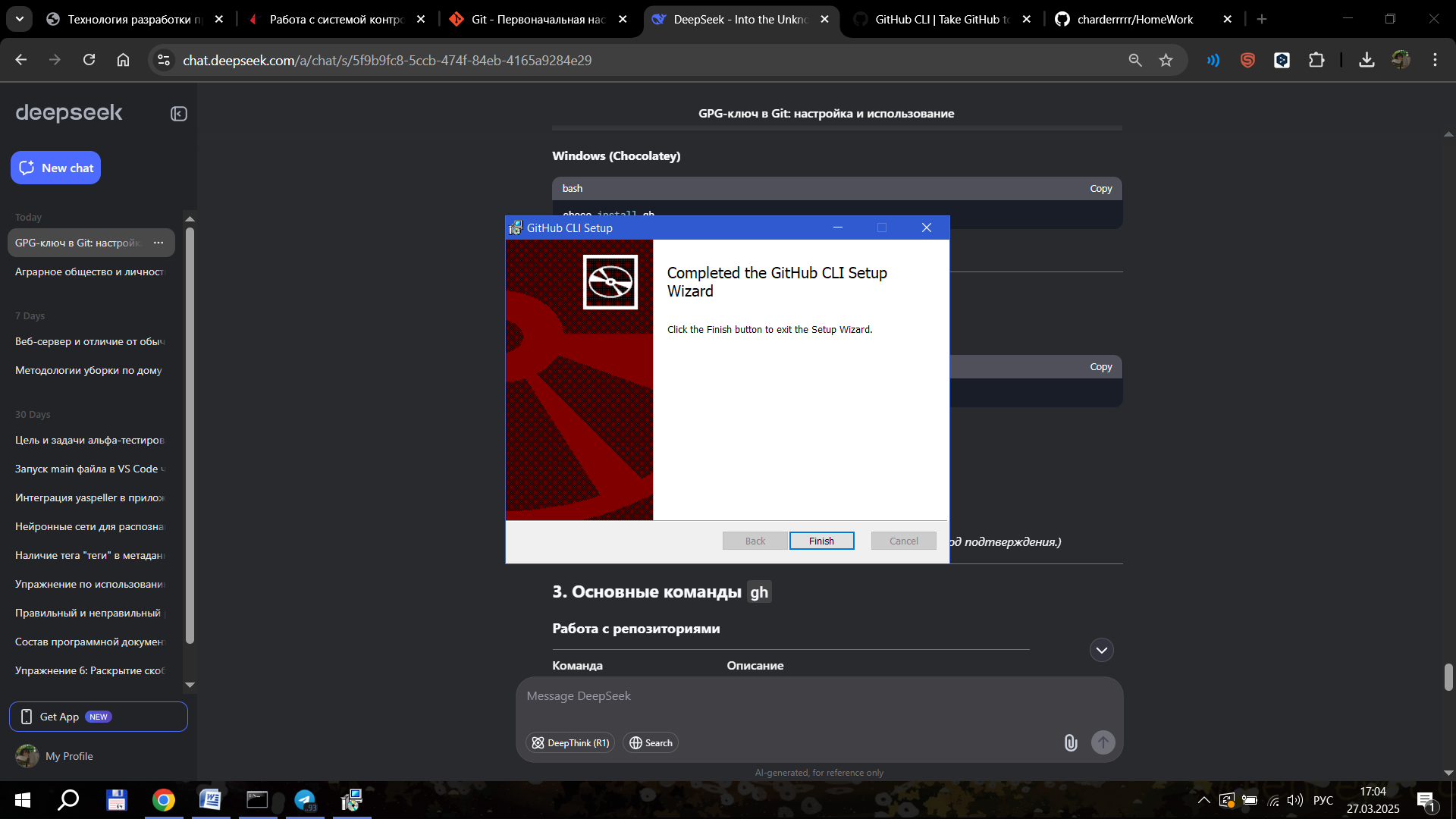


Рисунок 14 – завершение установки.

Далее в командной строке вбиваем «gh auth login» для привязки к аккаунту и следуем инструкции (Рисунок 15).

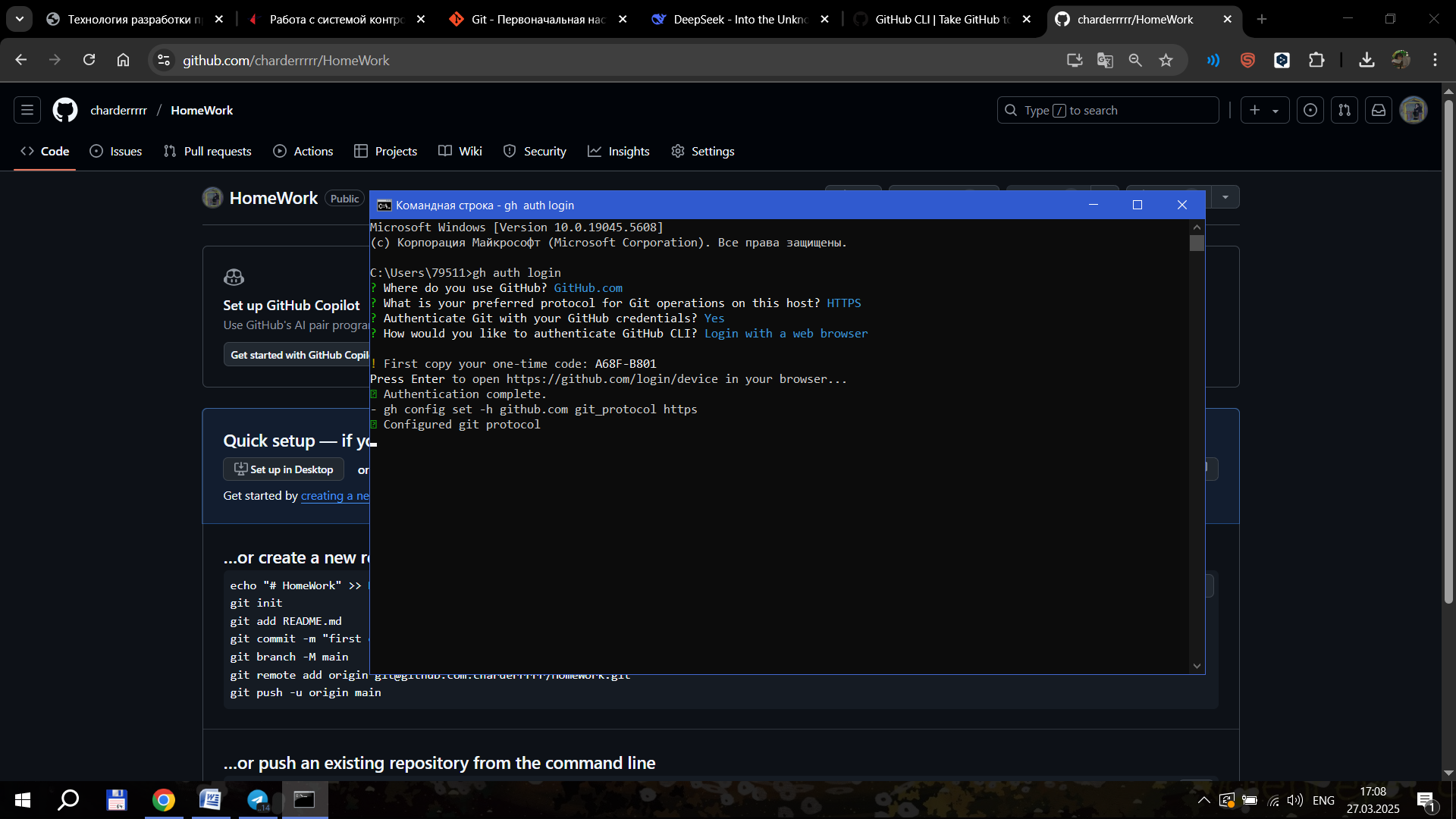


Рисунок 15 – привязка к текущему аккаунту.

**Интерактивное задание на сайте:**

1. Нажимаем на знак плюса в правом верхнем углу, заполняем данные и жмем на кнопку «Создать репозиторий» (рисунок 16, рисунок 17)

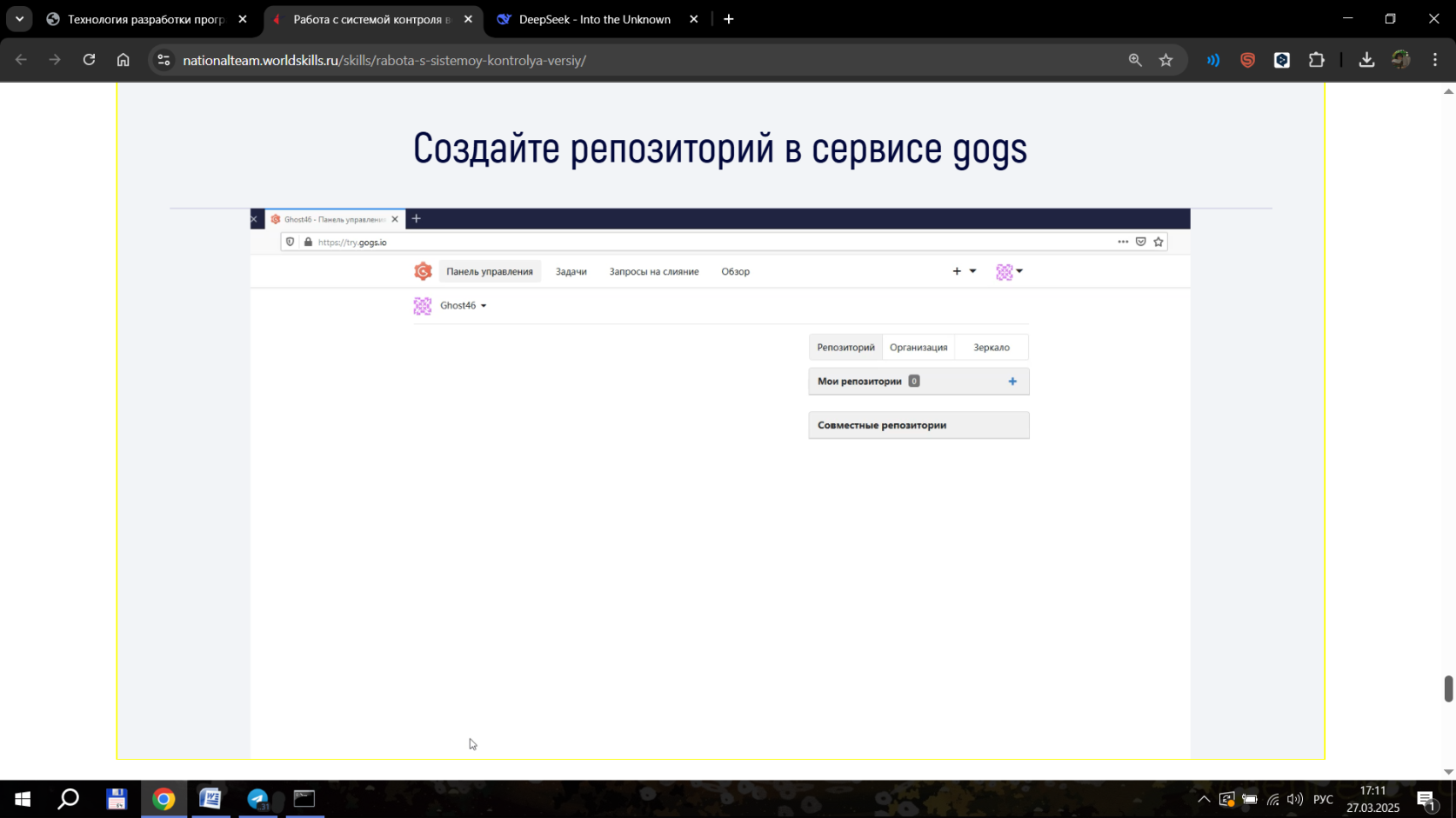


Рисунок 16 – интерактивное задание.

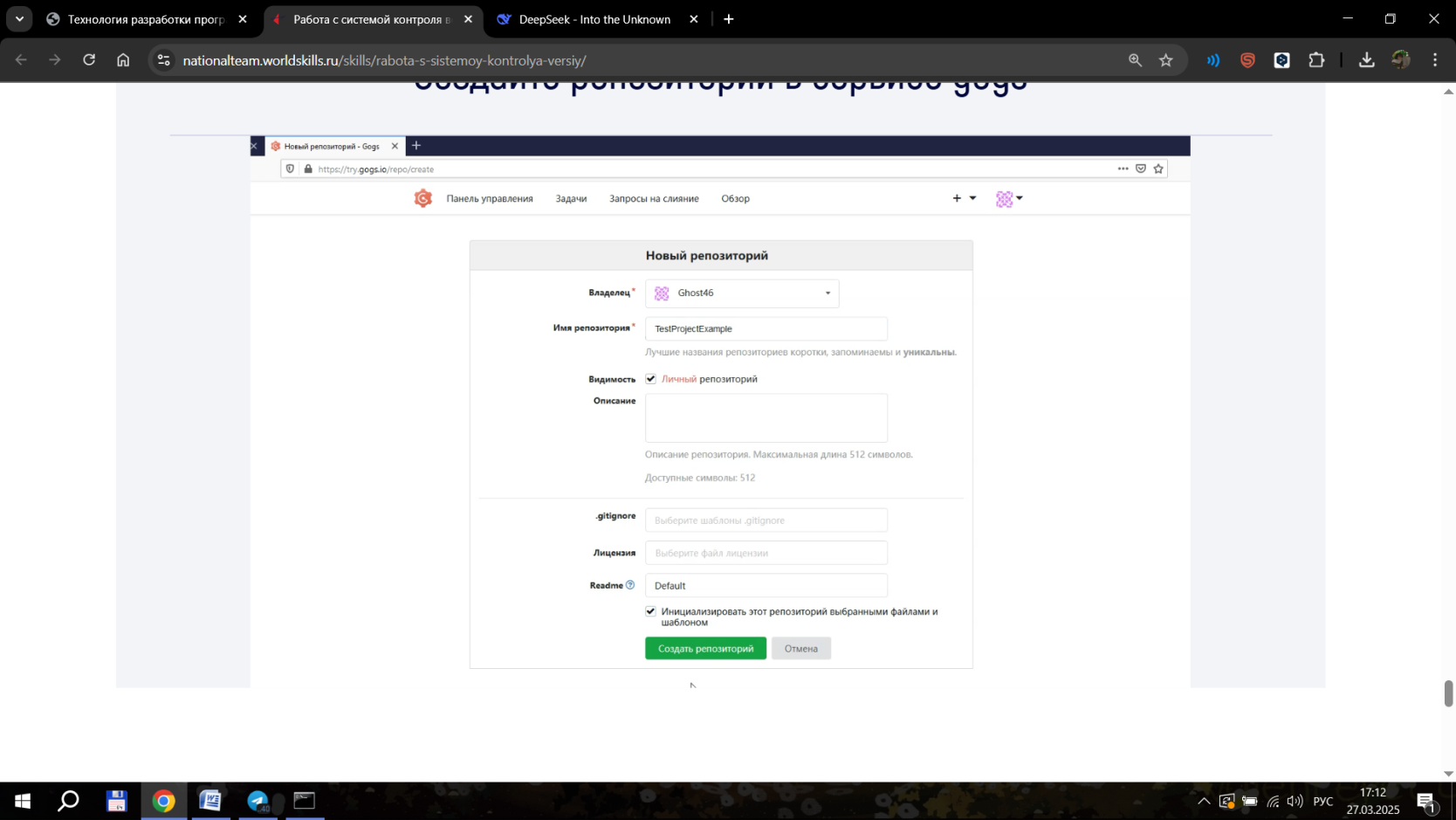


Рисунок 17 – интерактивное задание.

3. Задание пройдено (Рисунок 18).

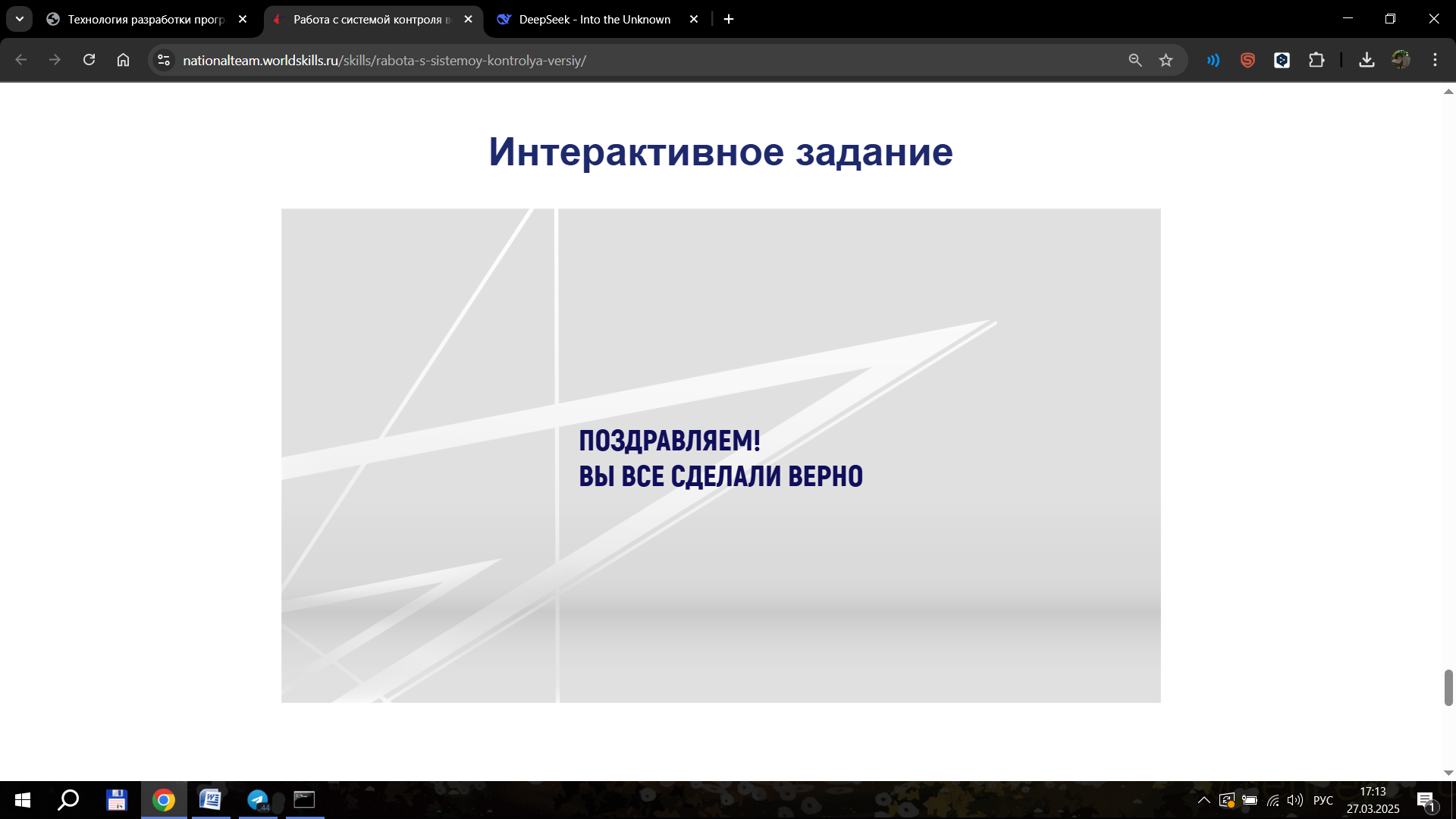


Рисунок 18 - интерактивное задание.

Проходим тест:

1. Нажимаем на кнопку «Стартуем!» (Рисунок 19)

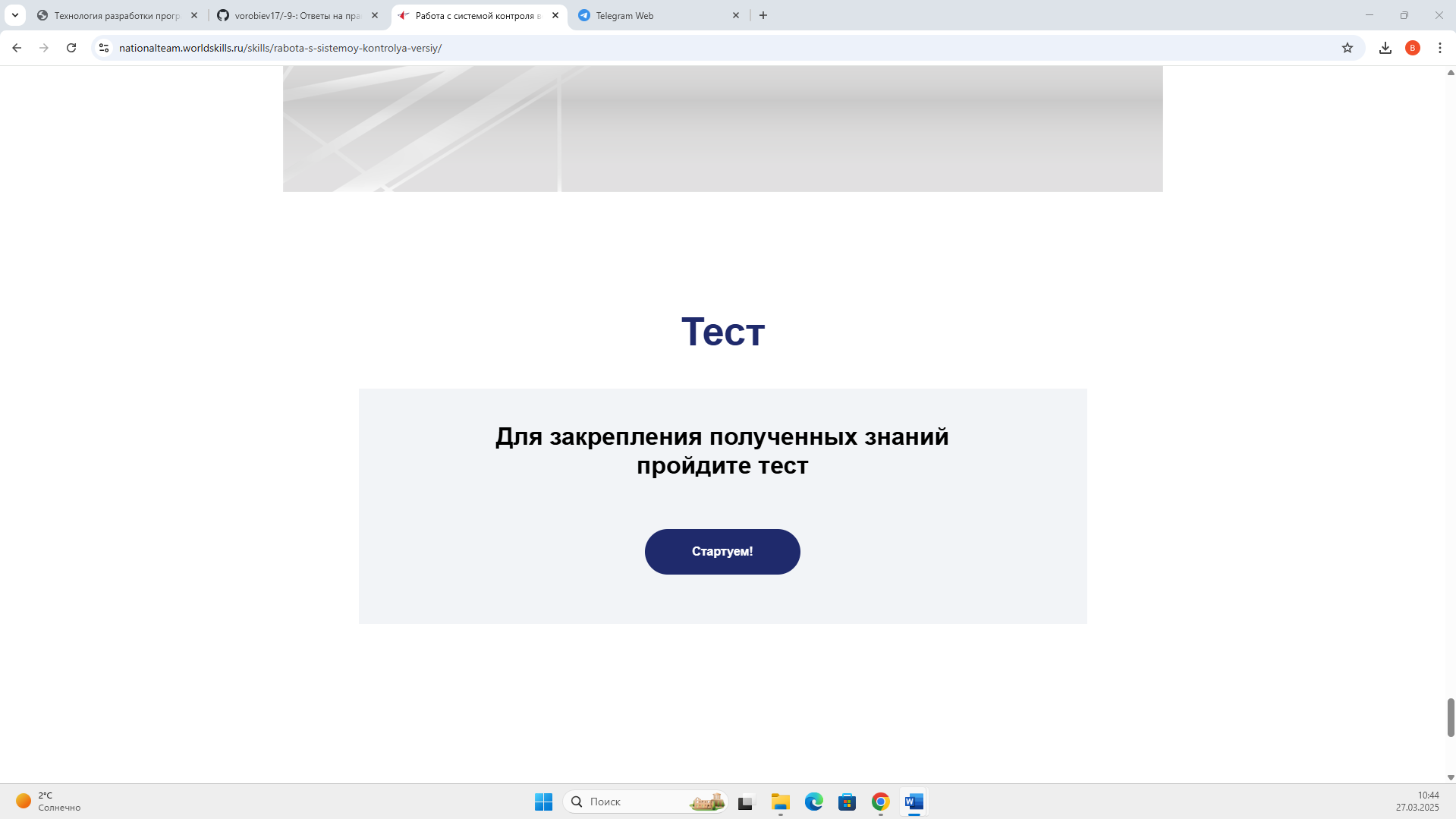


Рисунок 19 – тест.

2. Ответы представлены ниже (Рисунок 20)

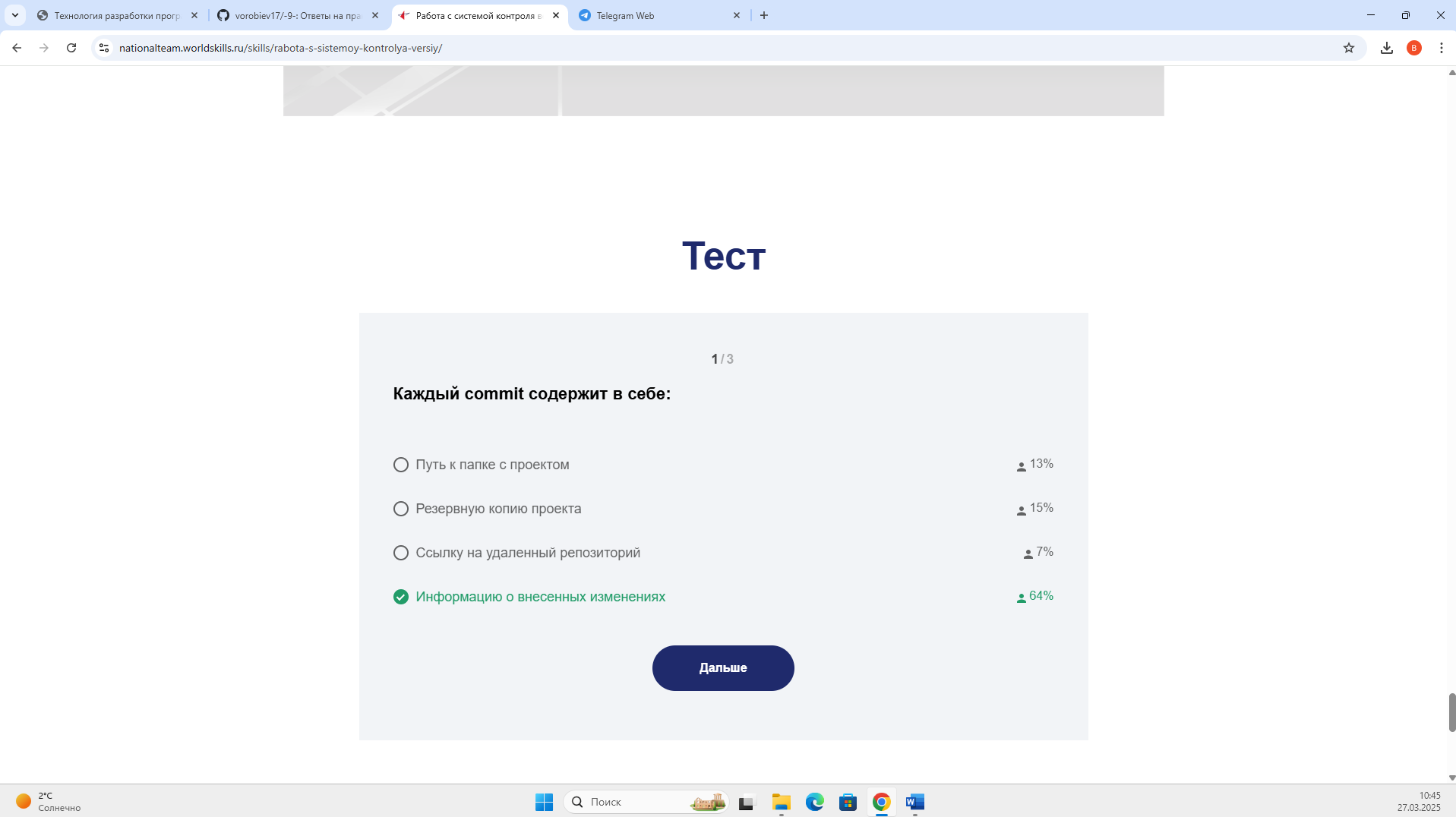


Рисунок 20 – 1 вопрос.

3. Ответы представлены ниже (Рисунок 21)

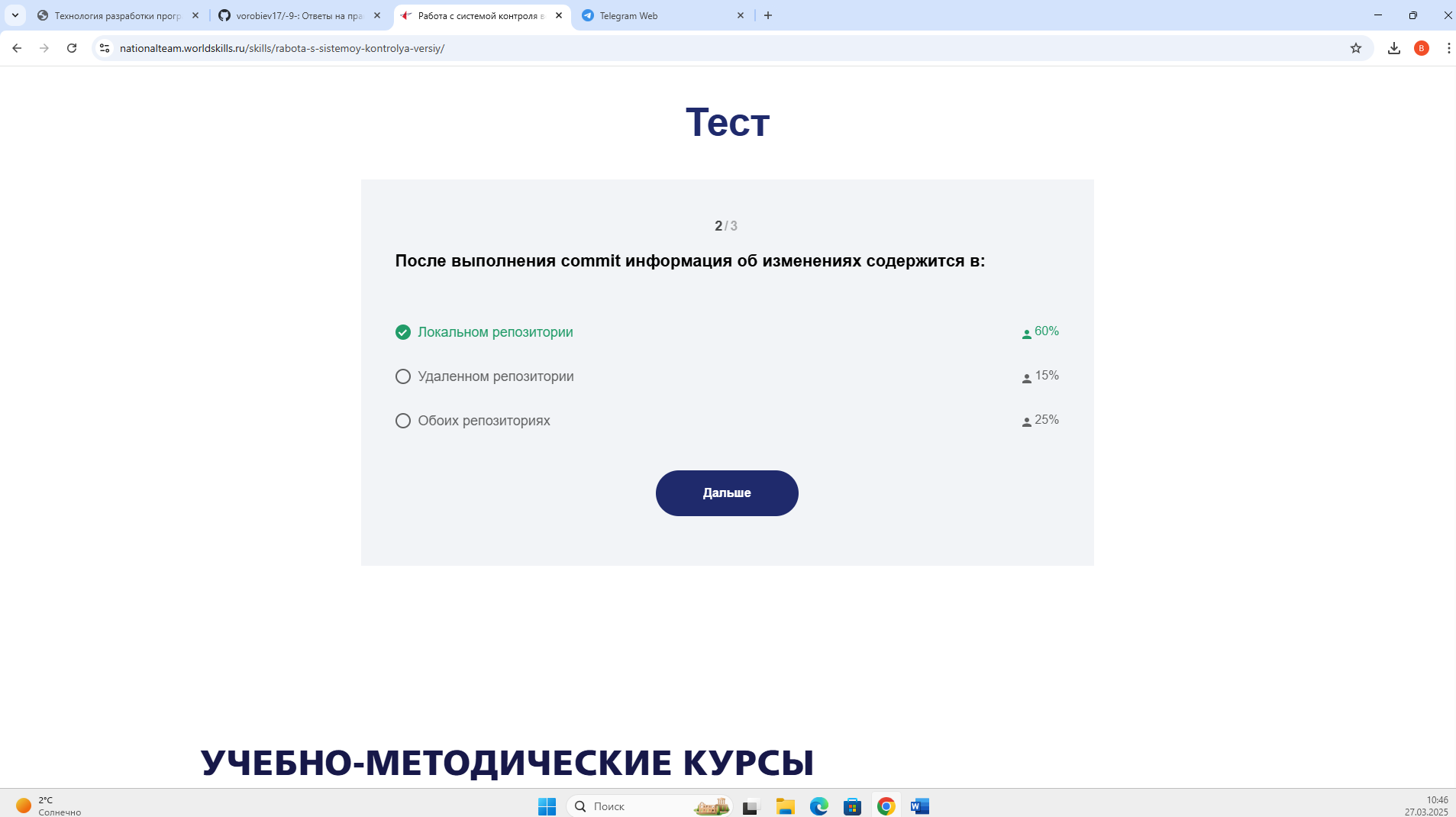


Рисунок 21 – 2 вопрос.

4. Ответы представлены ниже (Рисунок 22).

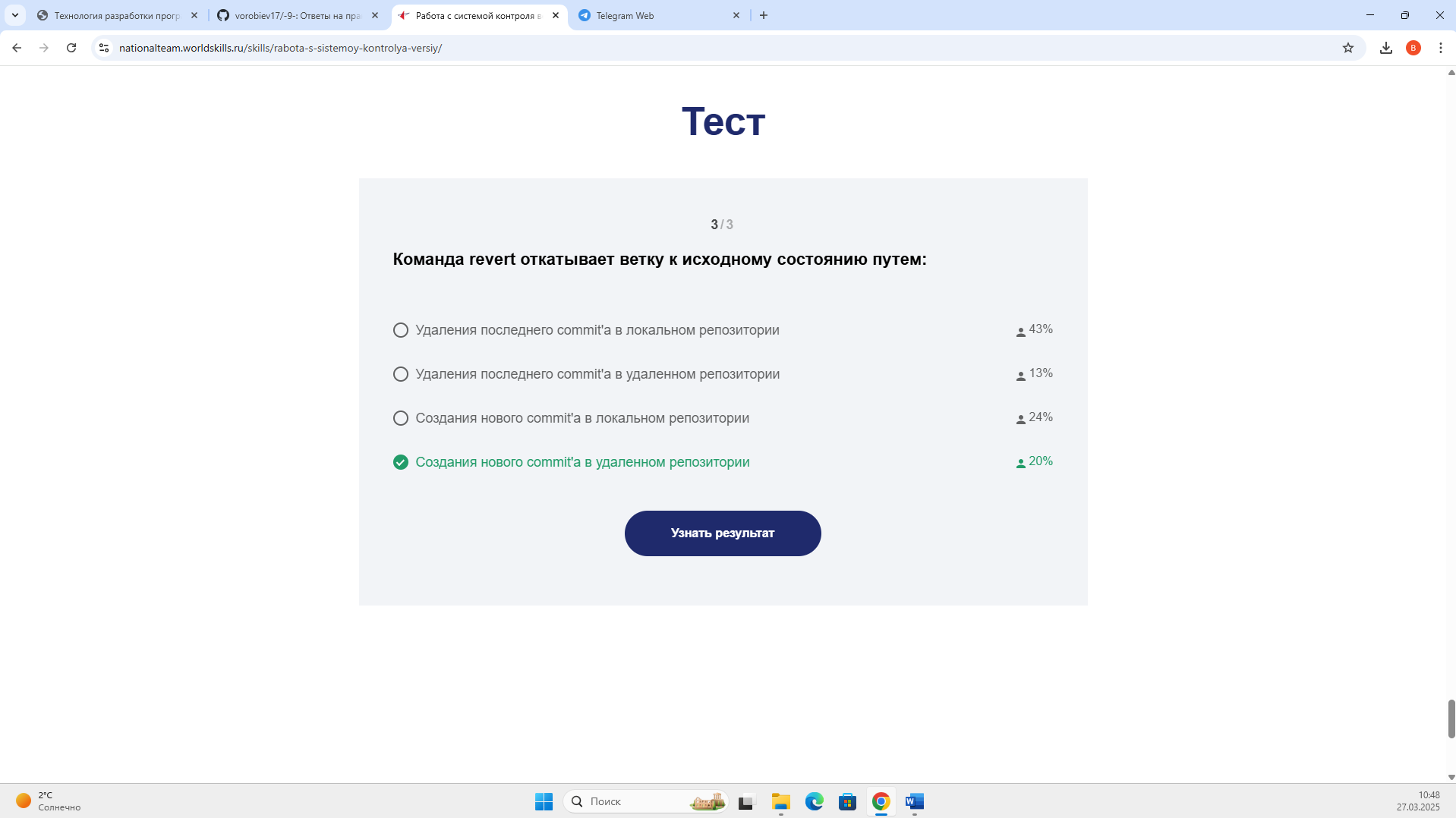


Рисунок 22 – 3 вопрос.

5. Результат (Рисунок 22).

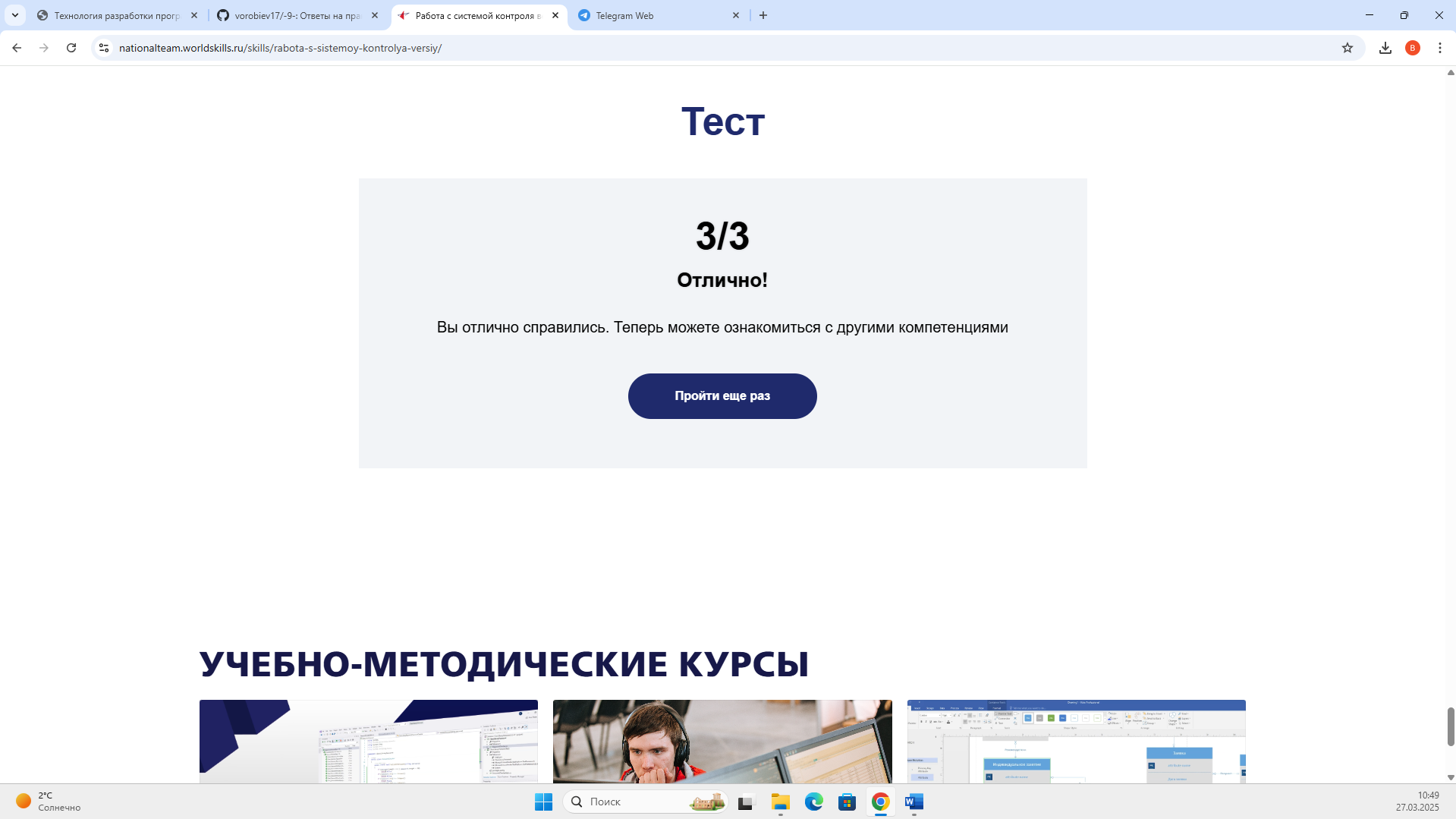


Рисунок 22 – результат

**Вывод:**  в ходе работы я изучила на практике понятия и компоненты систем контроля версии (СКВ или VCS), научилась приемам работы с ними, освоила распространенный сервис для работы с (СКВ или VCS) – GitHub.com.

**Контрольные вопросы**

1. **Что такое системы контроля версий (СКВ) и для решения каких задач они предназначаются?**

Системы контроля версий (СКВ) — это инструменты, которые помогают разработчикам управлять изменениями в файлах проекта. Они позволяют отслеживать изменения, сделанные в коде, сохранять историю изменений и восстанавливать предыдущие версии файлов.

1. **Объясните следующие понятия СКВ и их отношения: хранилище, commit, история, рабочая копия.**

– Хранилище – это как архив, где лежат все версии файлов. Оно хранит всё, что происходило с проектом;

– Commit – это сохранение изменений. Когда делаешь commit, записываешь состояние проекта в определённый момент времени;

– История – это список всех сделанных коммитов. Она помогает увидеть, кто и когда вносил изменения;

– Рабочая копия – это та версия проекта, с которой работаешь прямо сейчас.

1. **Что представляют собой и чем отличаются централизованные и децентрализованные?**

– Централизованные – это когда есть главный сервер, и все изменения отправляются туда. Все работают через этот сервер;

– Децентрализованные (например, Git)– каждый разработчик имеет свою копию проекта, включая всю историю изменений. Можно работать самостоятельно и потом обмениваться изменениями. Это полезно,к примеру, если требуется узнать разные варианты решения задачи от разных разработчиков.

1. **Какие системы контроля версий вы знаете?**

– Git — самая популярная сейчас, децентрализованная;

– BitBucket – это централизированная система управления версиями.

1. **Какие существуют основные операции в системе контроля версий?**

– Commit — сохраняем изменения;

– Update/Checkout — получаем последние изменения из общего хранилища;

– Branching — создаём отдельную ветку для параллельной работы;

– Merge — соединяем изменения из разных веток;

– Revert — откатываемся назад, если что-то пошло не так.

1. **СКВ? Приведите примеры СКВ каждого вида.**

– Централизованные: BitBucket;

– Децентрализованные: Git.

1. **Опишите действия с СКВ при единоличной работе с хранилищем.**
2. Клонировать проект.
3. Внести изменения.
4. Сделать commit.
5. **Опишите порядок работы с общим хранилищем в централизованной СКВ.**
6. Клонировать проект с основного сервера.
7. Внести изменения.
8. Сделать commit.
9. Отправить изменения в общее хранилище (push)
10. **Что такое и зачем может быть нужна разность (diff)?**

Разность (diff) — это процесс сравнения двух версий файлов, позволяющий увидеть изменения, которые были внесены между ними. Это помогает понять, какие изменения были сделаны и почему.

1. **Что такое и зачем может быть нужно слияние (merge)?**

Merge нужен, чтобы соединить изменения из разных веток или версий файла. Например, ты доработал одну фичу, а твой коллега — другую. Теперь надо всё это слить в одну версию.

1. **Что такое конфликты (conflict) и каков процесс их разрешения (resolve)?**

Конфликты возникают, когда два человека меняют одни и те же строки в одном файле. Система не может сама решить, чьё изменение оставить, поэтому тебе приходится вручную выбирать правильное решение.

1. **Поясните процесс синхронизации с общим хранилищем («обновления») в децентрализованной СКВ.**

– Pull — получаешь изменения из удалённого репозитория.

– Push — отправляешь свои изменения туда.

1. **Что такое и зачем могут быть нужны ветви (branches)?**

Ветви позволяют параллельно работать над разными частями проекта. Например, можно создать новую ветку для добавления новой функции, а в это время другие члены команды продолжают работать над основной версией. Позже можно будет слить изменения из этой ветки в основную.

1. **Объясните смысл действия rebase в СКВ Git.**

Rebase позволяет переписать историю коммитов. Например, можно сделать так, чтобы одни коммиты шли после коммитов другого человека, а не перемешивались с ними.

1. **Как и зачем можно игнорировать некоторые файлы при commit?**

Игнорировать файлы при **commit** можно с помощью файла **.gitignore**. Это нужно, чтобы не путать репозиторий временными файлами или файлами, которые не нужны другим разработчикам. Например, можно игнорировать файлы логов или временные файлы IDE.

1. **Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?**

VersionControlSystem **(**VCS**)** — это инструменты, которые помогают разработчикам управлять изменениями в файлах проекта. Они позволяют отслеживать изменения, сделанные в коде, сохранять историю изменений и восстанавливать предыдущие версии файлов.

1. **Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.**

– Хранилище – это как архив, где лежат все версии файлов. Оно хранит всё, что происходило с проектом;

– Commit – это сохранение изменений. Когда делаешь commit, записываешь состояние проекта в определённый момент времени;

– История – это список всех сделанных коммитов. Она помогает увидеть, кто и когда вносил изменения;

– Рабочая копия – это та версия проекта, с которой работаешь прямо сейчас.

1. **Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида?**

– **Централизованные** – это когда есть главный сервер, и все изменения отправляются туда. Все работают через этот сервер;

– **Децентрализованные (например, Git)** – каждый разработчик имеет свою копию проекта, включая всю историю изменений. Можно работать самостоятельно и потом обмениваться изменениями. Это полезно, к примеру, если требуется узнать разные варианты решения задачи от разных разработчиков.

1. **Опишите действия с VCS при единоличной работе с хранилищем.**
2. Клонируешь проект.
3. Работаешь, вносишь изменения.
4. Делаешь commit.
5. Повторяешь, если надо.
6. **Опишите порядок работы с общим хранилищем VCS.**
7. Клонировать проект с основного сервера.
8. Внести изменения.
9. Сделать commit.
10. Отправить изменения в общее хранилище (push)
11. **Каковы основные задачи, решаемые инструментальным средством git?**
12. Управление историей изменений.
13. Поддержание различных ветвей разработки.
14. Удобство совместной работы команды.
15. Возможность восстановления предыдущих версий.
16. **Назовите и дайте краткую характеристику командам git.**

– gitinit – инициализация нового репозитория;

– gitclone– клонирование существующего репозитория;

– gitpush – Команда «gitpush» используется для установления связи с удалённым репозиторием, вычисления локальных изменений отсутствующих в нём, и собственно их передачи в вышеупомянутый репозиторий. Этой команде нужно право на запись в репозиторий, поэтому она использует аутентификацию;

– gitadd – добавление изменений в индекс;

– gitcommit – фиксация изменений данных, которые были добавлены в индекс ранее;

– gitpush – отправка изменений на удаленный сервер;

– gitpull – Команда «gitpull» работает как комбинация команд «gitfetch» и «gitmerge», т. е. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой;

– gitbranch – создание новых веток;

– gitmerge– слияние веток.

1. **Приведите примеры использования при работе с локальным и удалённым репозиториями.**

При работе с локальными репозиториями:

– Создаем новый проект: gitinit;

– Добавляем файлы: gitadd…;

– Фиксируем изменения: gitcommit-m"Initialcommit".

При работе с удаленными репозиториями:

– Клонируем проект: gitclonehttps://github.com/user/repo.git;

– Отправляем изменения: gitpushoriginmain;

– Получаем обновления: gitpulloriginmain.

1. **Что такое и зачем могут быть нужны ветви (branches)?**

Ветви позволяют параллельно работать над разными частями проекта. Например, можно создать новую ветку для добавления новой функции, а в это время другие члены команды продолжают работать над основной версией. Позже можно будет слить изменения из этой ветки в основную.

1. **Как и зачем можно игнорировать некоторые файлы при commit?**

Игнорировать файлы при commit можно с помощью файла .gitignore. Это нужно, чтобы не путать репозиторий временными файлами или файлами, которые не нужны другим разработчикам. Например, можно игнорировать файлы логов или временные файлы IDE.