«Утверждаю»

Преподаватель спец дисциплины

«Технология разработки программного обеспечения»

Абдрахманова Зульфия Амуровна

«27» Марта 2025г.

**Лабораторная работа №9**

**Изучение работы в системе контроля версий**

**Выполнил:**

Студент группы ПР-273, Твердохлебова Лидия

Челябинск, 2025

**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.

**Задания:**

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git и автоматические подписи git.
5. Зарегистрироваться на Github.
6. Создать локальный каталог для выполнения заданий по предмету.
7. Настройка gh.

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

Так как я выполняю практическую работу на своем домашнем компьютере, где уже установлен и настроен GitHub, то базовую конфигурацию настраивать снова не нужно. Базовая конфигурация представлена ниже (Рисунок 1).

Команда: git config --list

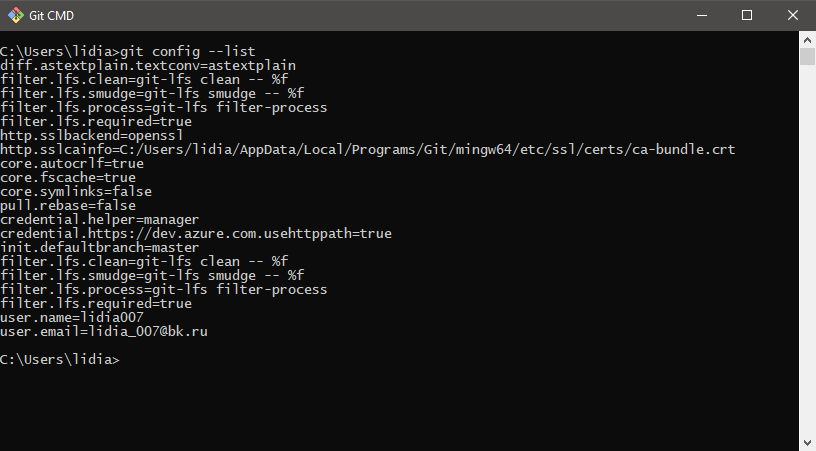


Рисунок 1 - Базовая конфигурация

**2. Создать ключ SSH.**

Команда: ssh-keygen -t ed25519 -C "lidia\_007@bk.ru"

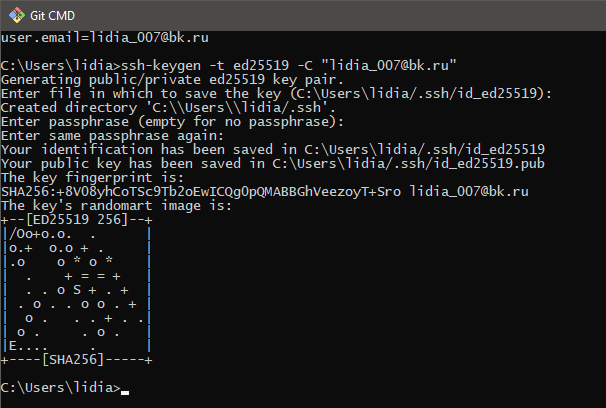
****

Рисунок 2 - Создание ключа SSH

Теперь добавляем SSH ключ в учетную запись GitHub.

Команда: git-gui

****

Рисунок 3 *–* Команда

Открывается приложение Git Gui. Во вкладке «Help» находим «Show SSH key» и нажимаем. Нам выводится наш ключ. Копируем.

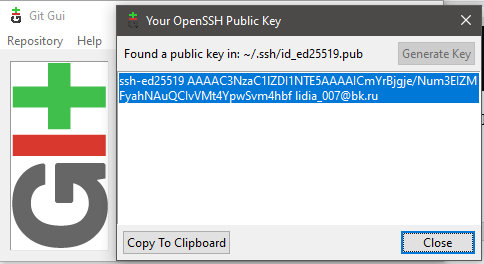
****

Рисунок 4 *–* КопируемSSH ключ

Затем заходим в Git Hub -> «настройки» -> «Ключи SSH и GPG» и вставляем скопированное:

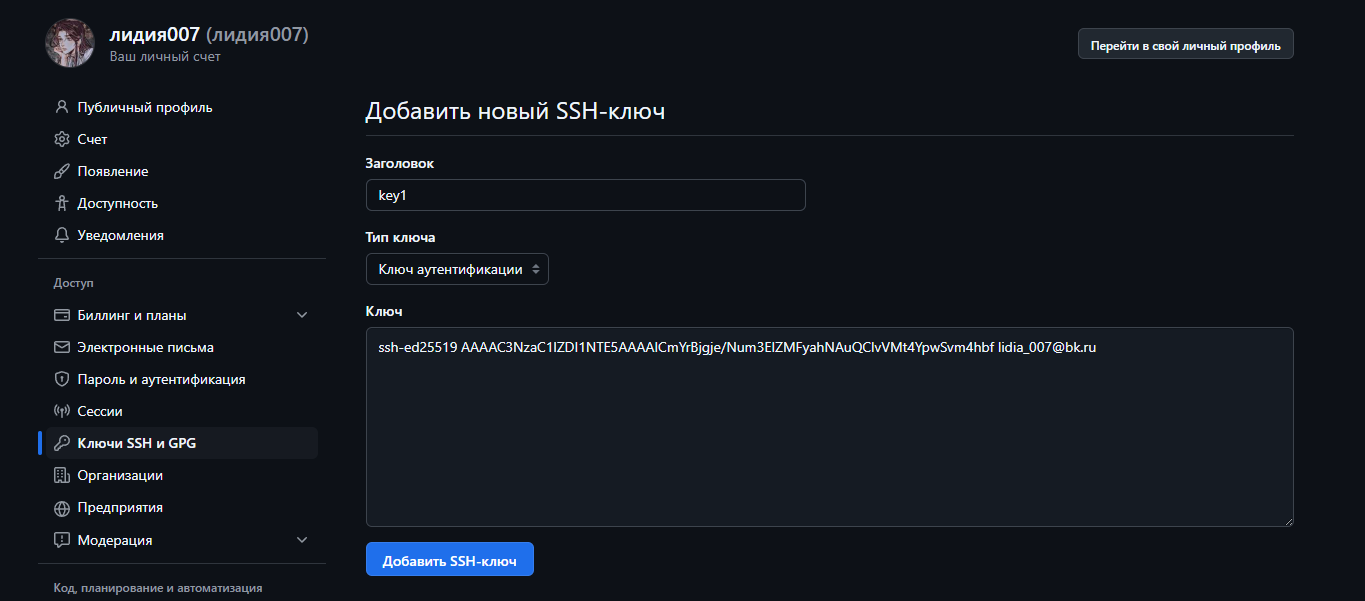
****

Рисунок 5 *–* Вставляем ключ

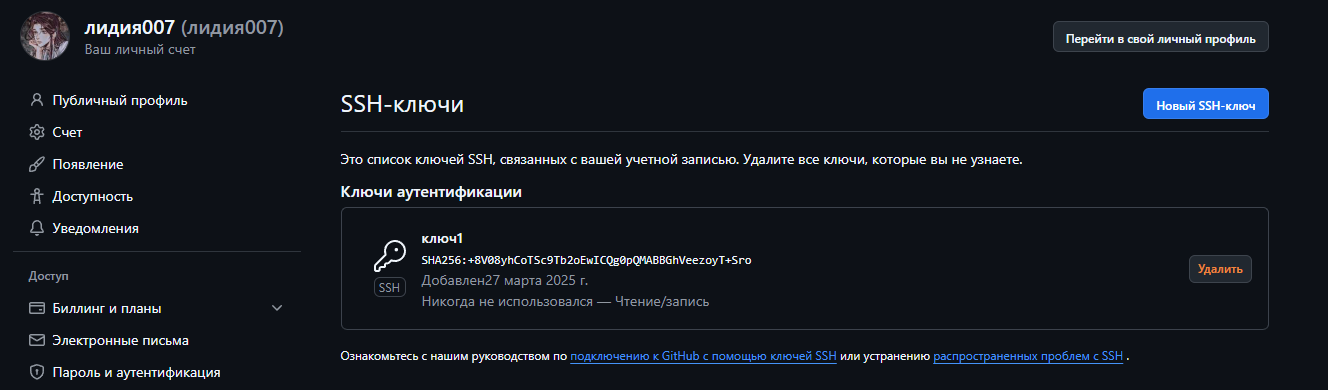
****

Рисунок 6 *–* Ключ добавлен

**3. Создать ключ PGP.**

Для этого нужно установить GnuPG:



Рисунок 7 - установщик GnuPG

Для создания ключа вводим команду: gpg --full-generate-key

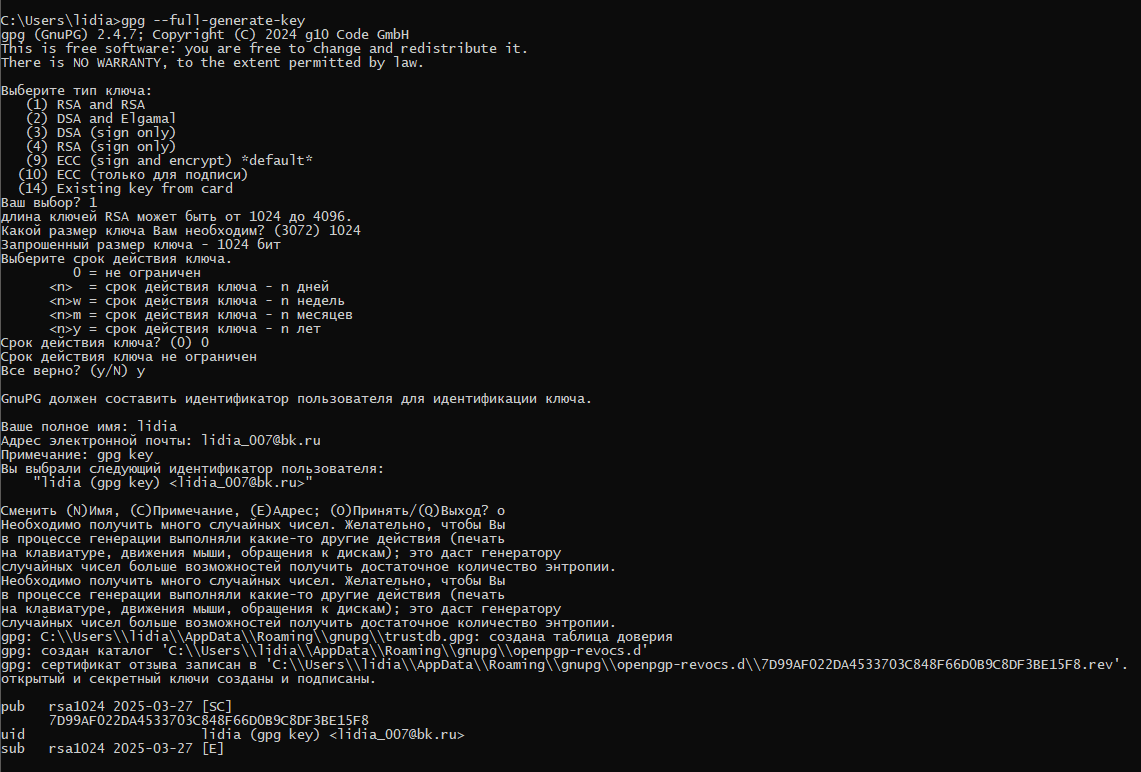


Рисунок 8 – Создание ключа

Далее добавляем ключ в Git Hub, для того чтобы получит полный ключ вводим следующие команды:

gpg --list-secret-keys --keyid-format=long

gpg --armor --export 6D0B9C8DF3BE15F8

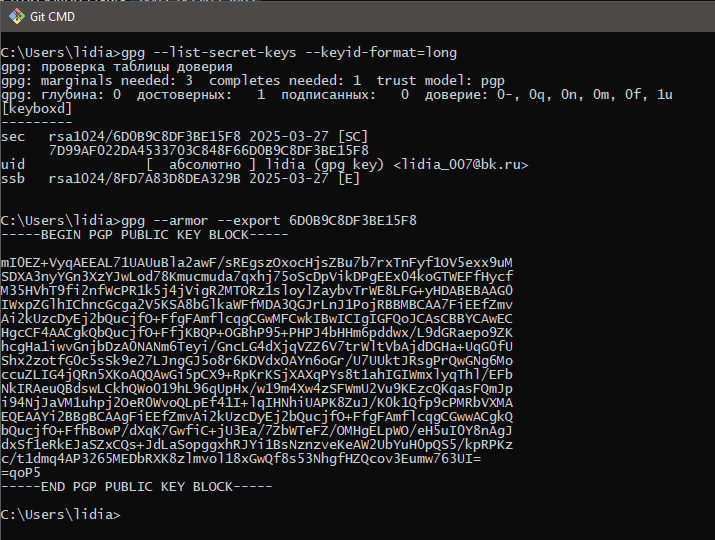


Рисунок 9 – КопируемGPG ключ

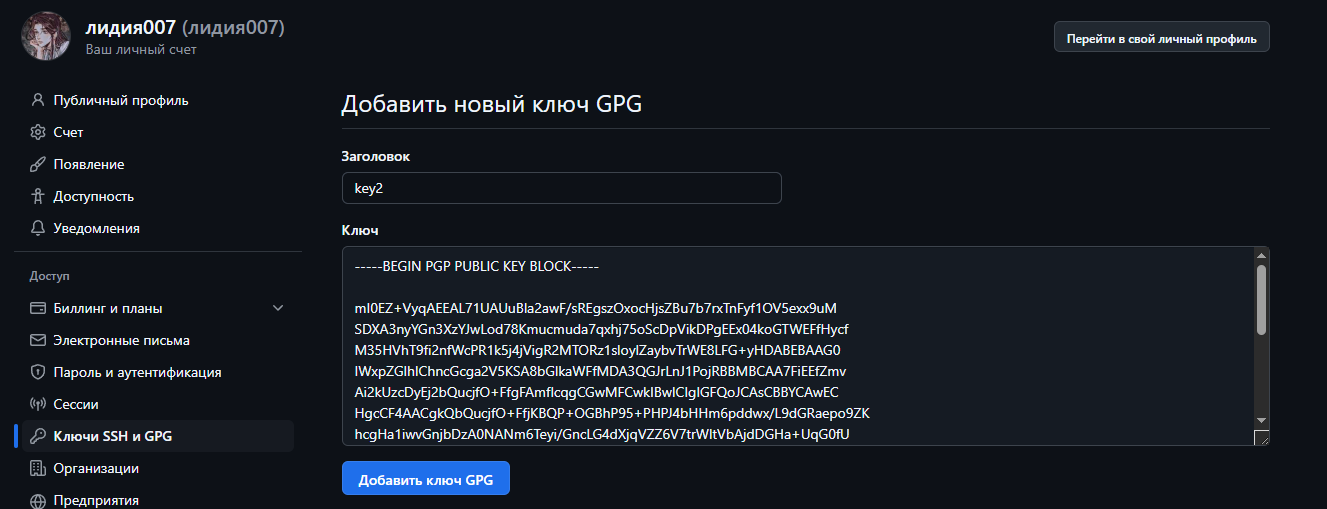


Рисунок 10 – Вставляем ключ

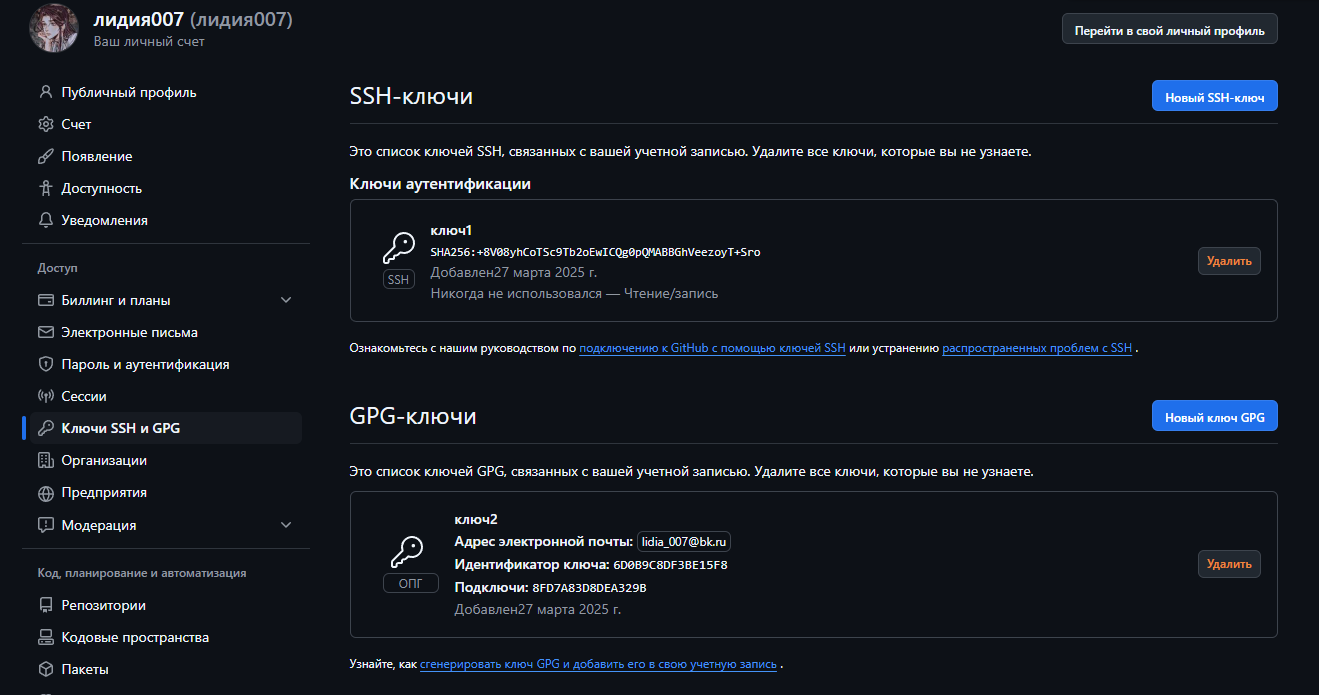


Рисунок 11 – Ключ добавлен

**4. Настроить подписи git.**

Устанавливаем ключ GPG по умолчанию. Включаем обязательную подпись для всех коммитов:

Команды:

git config --global user.signingkey 6D0B9C8DF3BE15F8

git config --global commit.gpgsing true



Рисунок 12 – Команды ключа по умолчанию и обязательной подписи коммитов

**5. Зарегистрироваться на Github.**

Так как у меня уже есть аккаунт, я просто в него зашла (Рисунок 13).

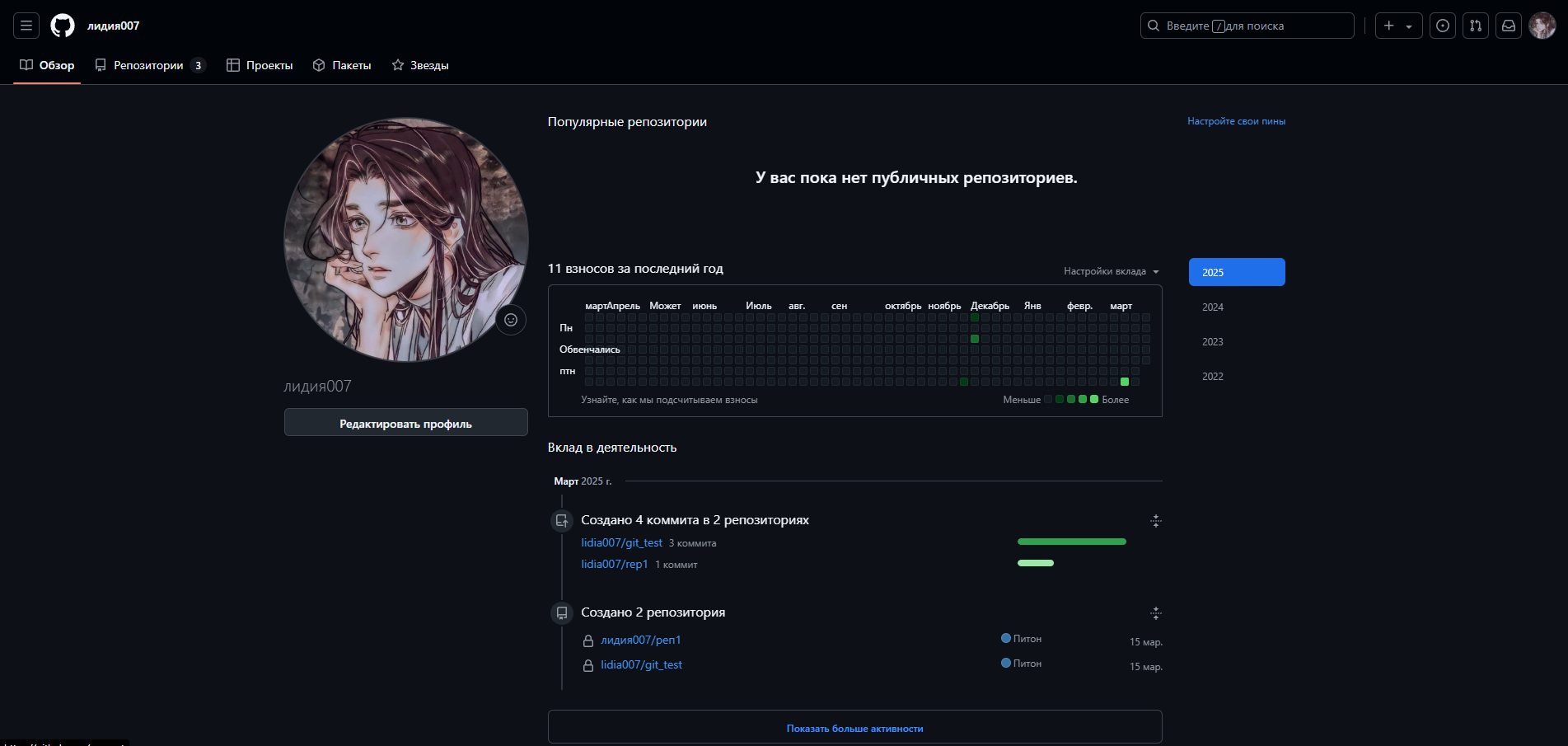


Рисунок 13 – Профиль аккаунта

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

Создаем новый репозиторий: «Мои репозитории» -> «Новое»

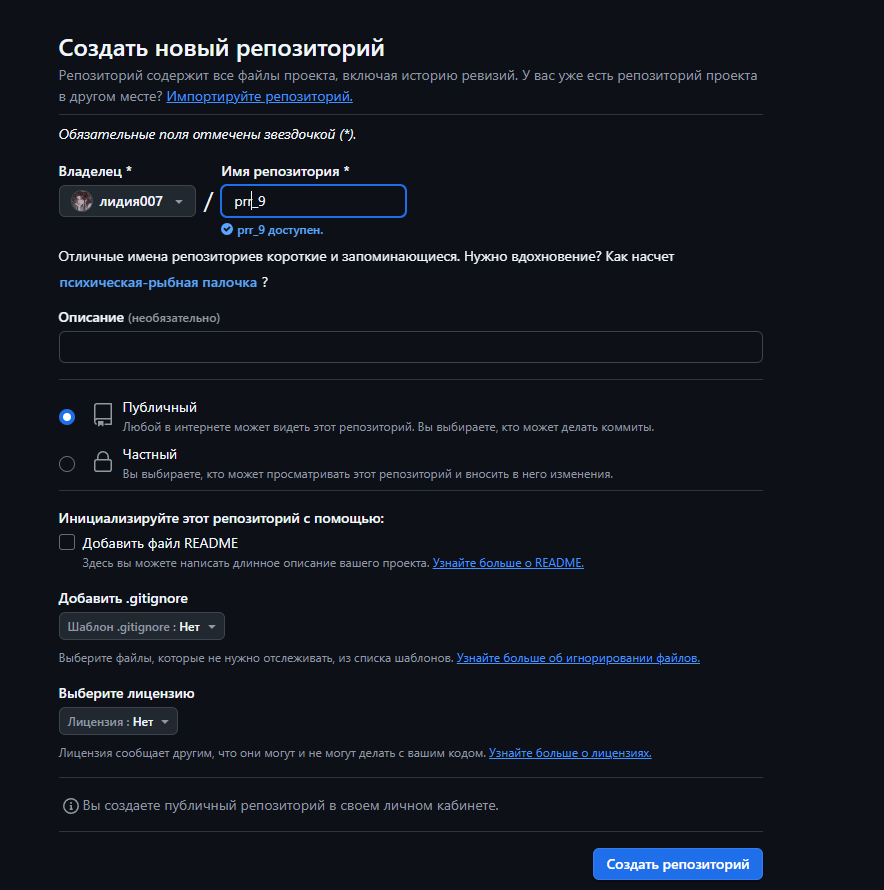
****

Рисунок 14 – Создание репозитория

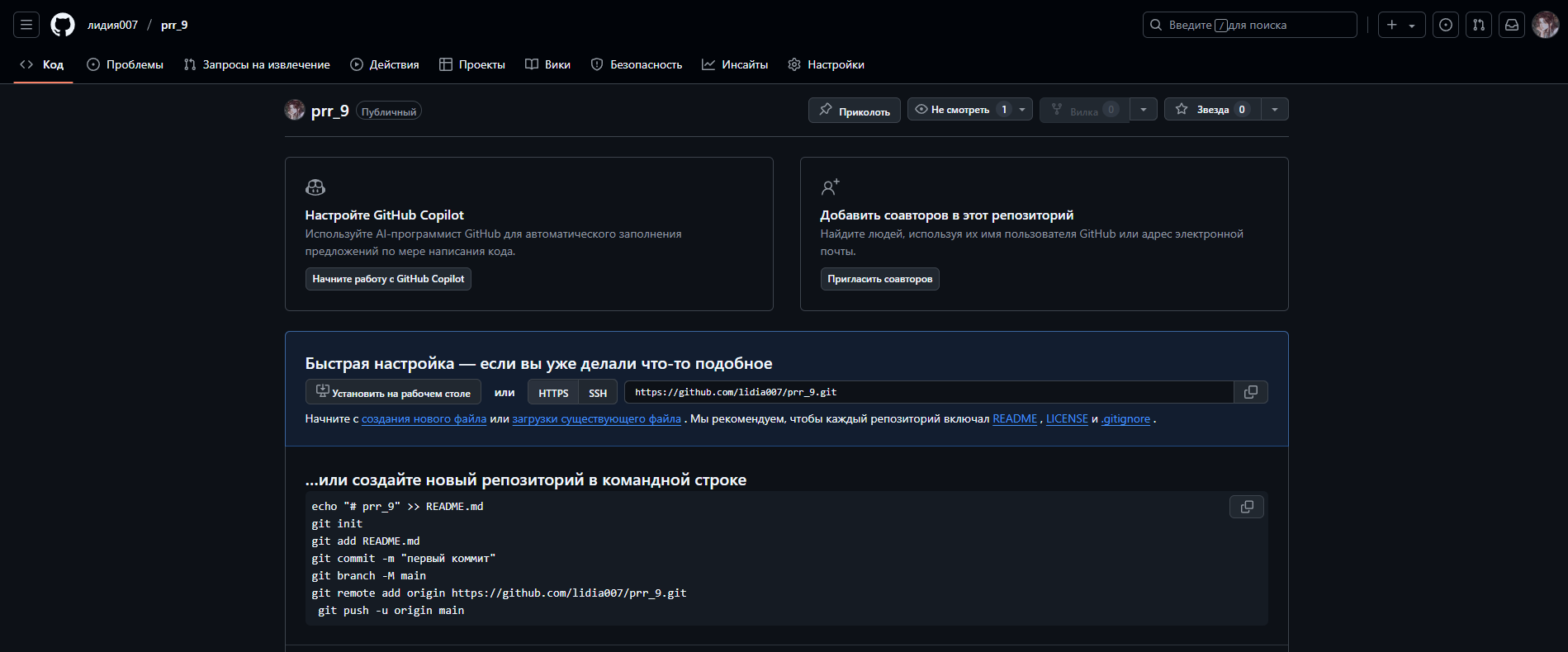
****

Рисунок 15 – Репозиторий создан

Помещаем файл «Контрольные вопросы» в репозиторий с пометкой «Контрольные вопросы без отчета»:

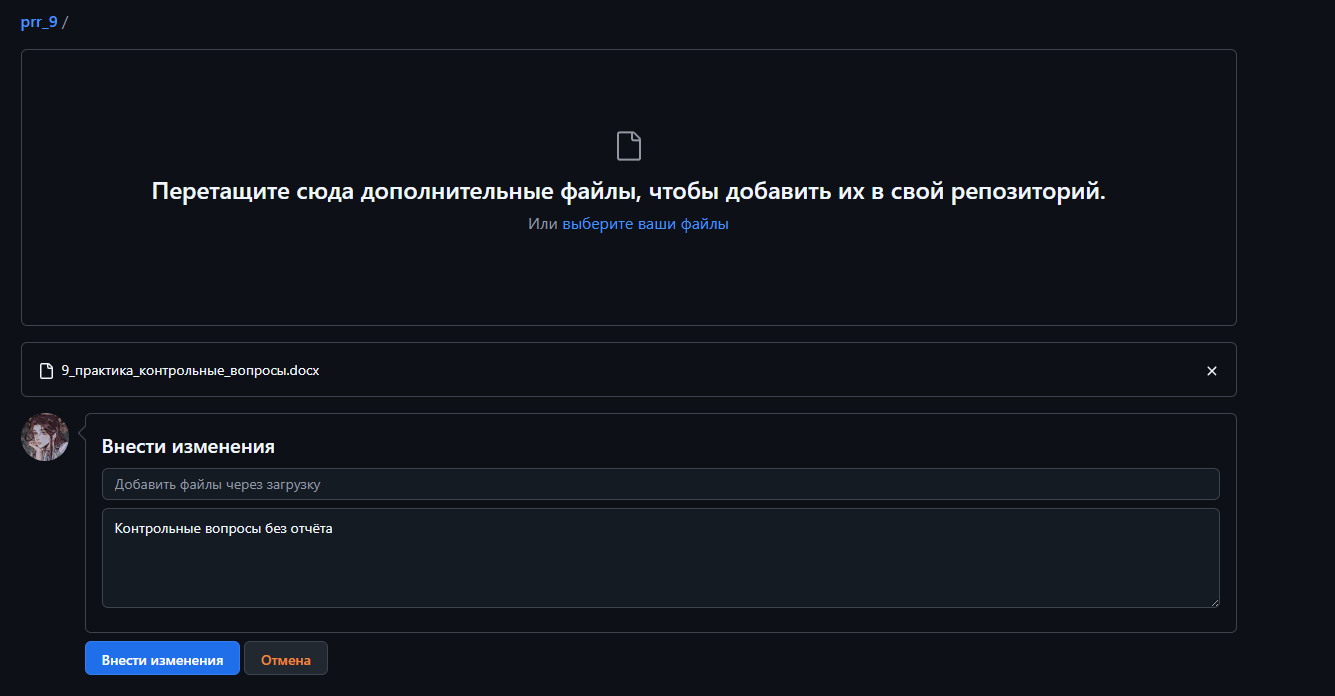


Рисунок 16 – Добавляем файл

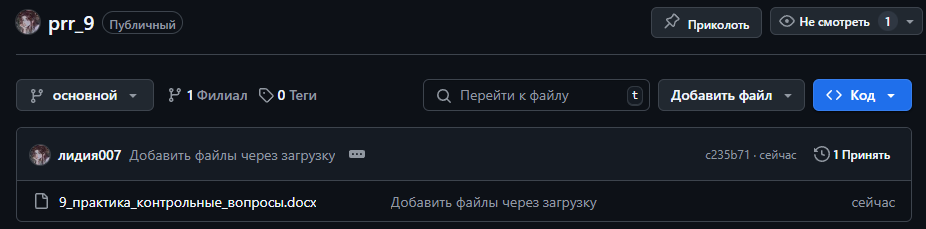


Рисунок 17 – Файл добавлен

Переходим в Git Brush:

Добавляем удаленный репозиторий с помощью команды:

git clone https://github.com/lidia007/TRPO

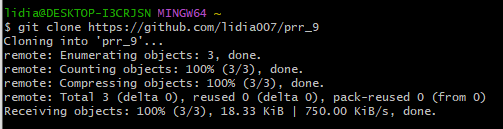


Рисунок 18 – Подключение удаленного репозитория

Создается папка prr\_9, где хранятся все файлы.

Переходим в директорию нового репозитория и инициализируем новый репозиторий:

Команда: cd C:/Users/lidia/TRPO

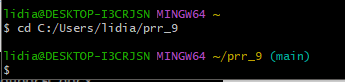


Рисунок 19 – Переходим в папку проекта

Команда: git init



Рисунок 20 – Новая директория и инициализация

И создаём новую ветку:

Команда: git checkout -b otchet

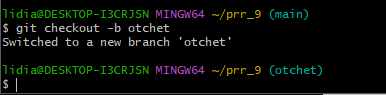


Рисунок 21 – Новая ветка «Otchet»

Кидаем файл с отчетом в папку с репозиторием:

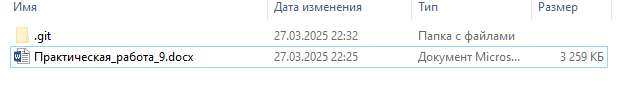


Рисунок 22 – Файл с отчетом в папке репозитория

Команда: git add Практическая\_работа\_9.docx



Рисунок 23 – Добавляем файл в ветку

Команда: git commit –m “Добавлен отчет”

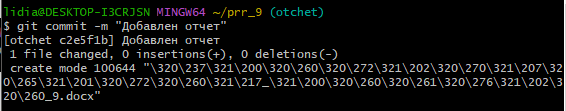


Рисунок 24 – Фиксируем изменения

Команда: git push –u origin otchet

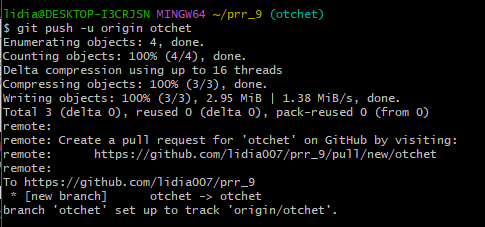


Рисунок 25 – Отправляем изменения



Рисунок 26 – Принимаем

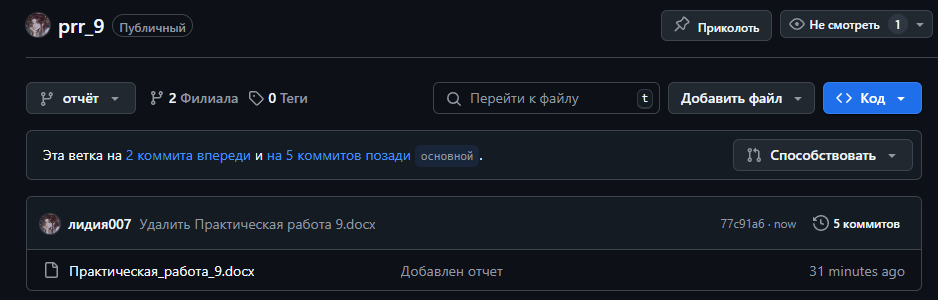


Рисунок 27 – Готово

Ссылка на репозиторий: https://github.com/lidia007/prr\_9.git

**Интерактивное задание на сайте:**

1. Нажимаем на знак плюса в правом верхнем углу, заполняем данные и жмем на кнопку «Создать репозиторий» (рисунок 28, рисунок 29)

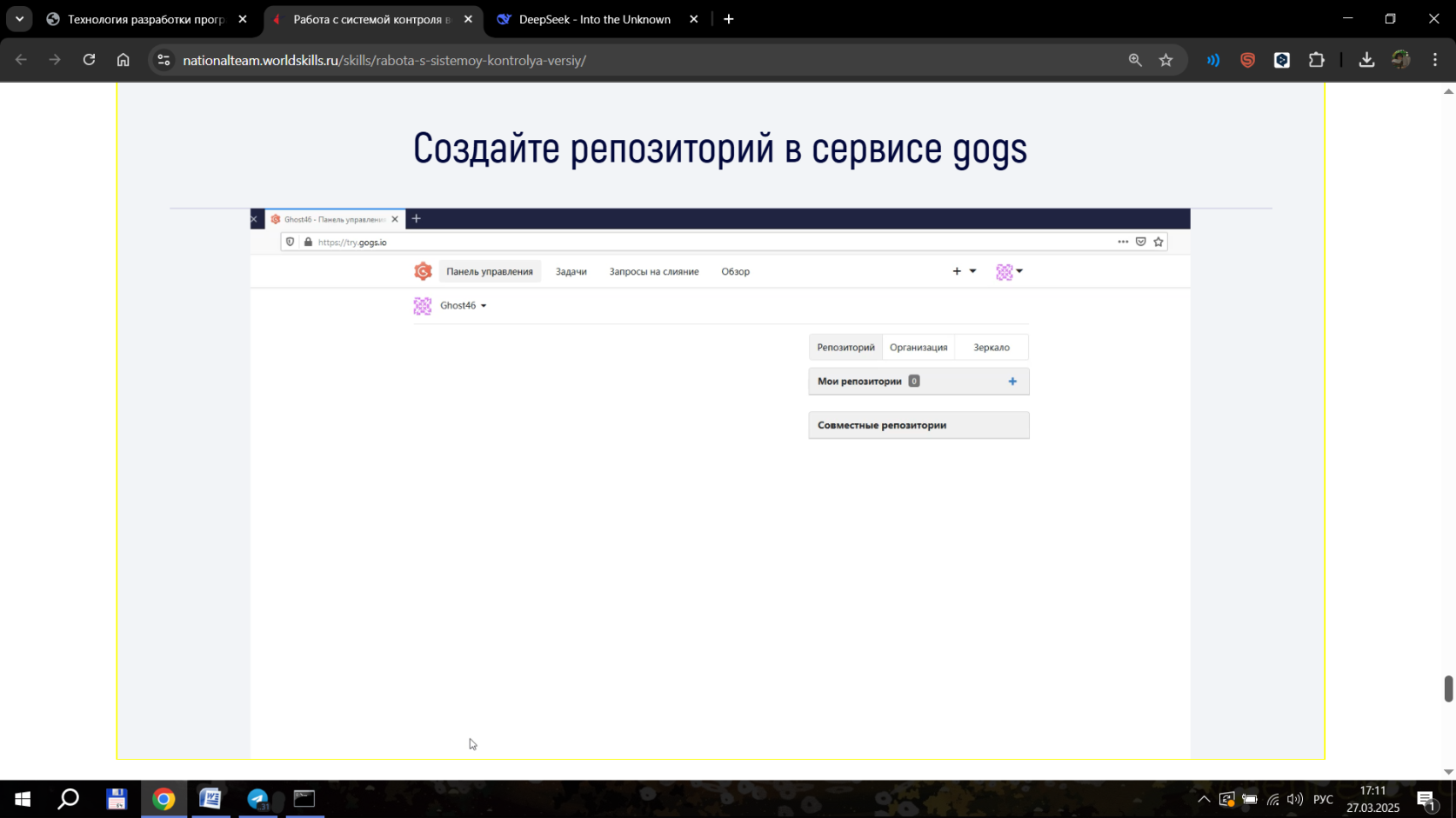


Рисунок 28 – интерактивное задание.

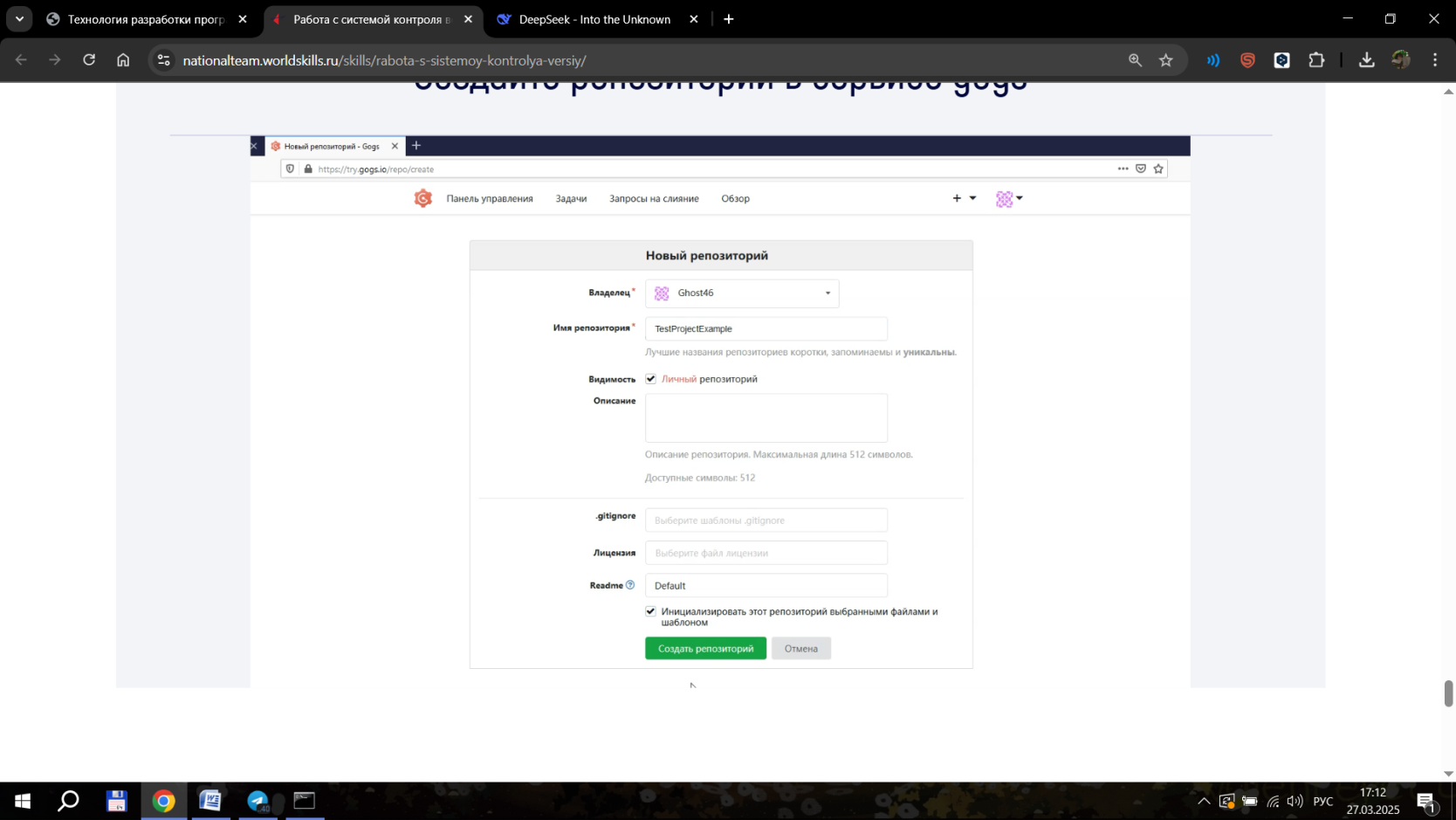


Рисунок 29 – интерактивное задание.

3. Задание пройдено (Рисунок 30).

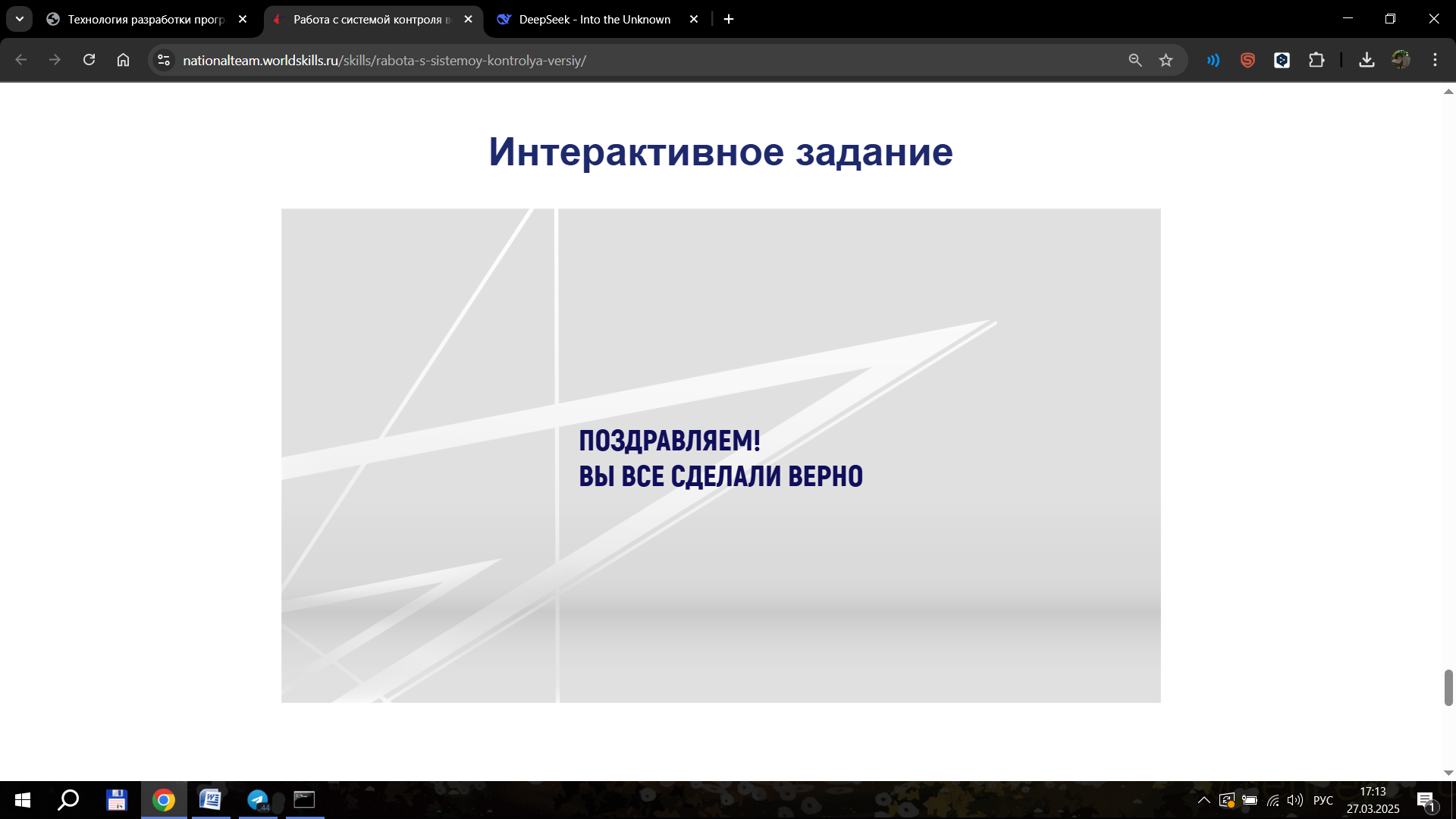


Рисунок 30 - интерактивное задание.

**Проходим тест:**

1. Нажимаем на кнопку «Стартуем!» (Рисунок 31)

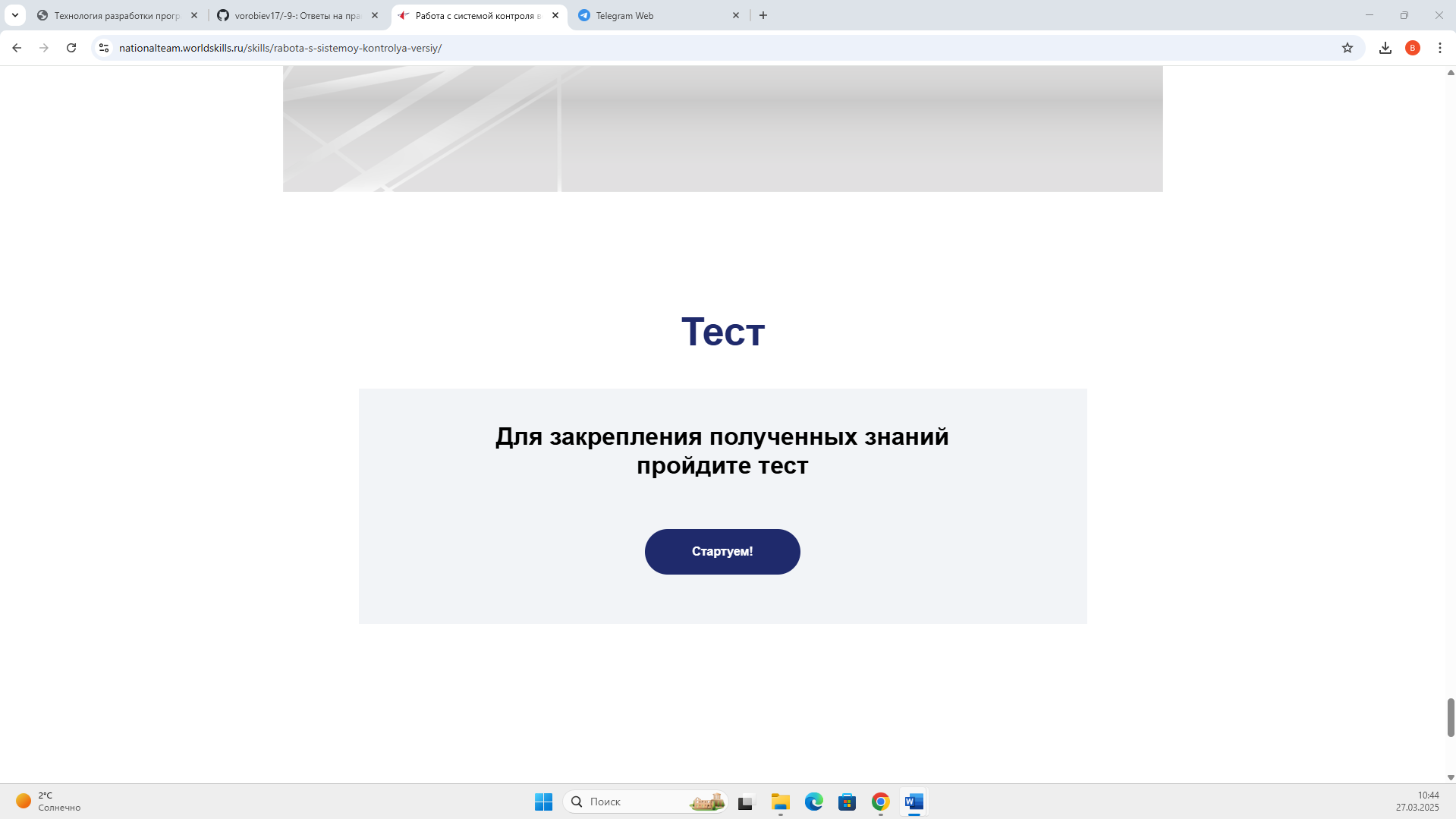


Рисунок 31 – тест.

2. Ответы представлены ниже (Рисунок 32)

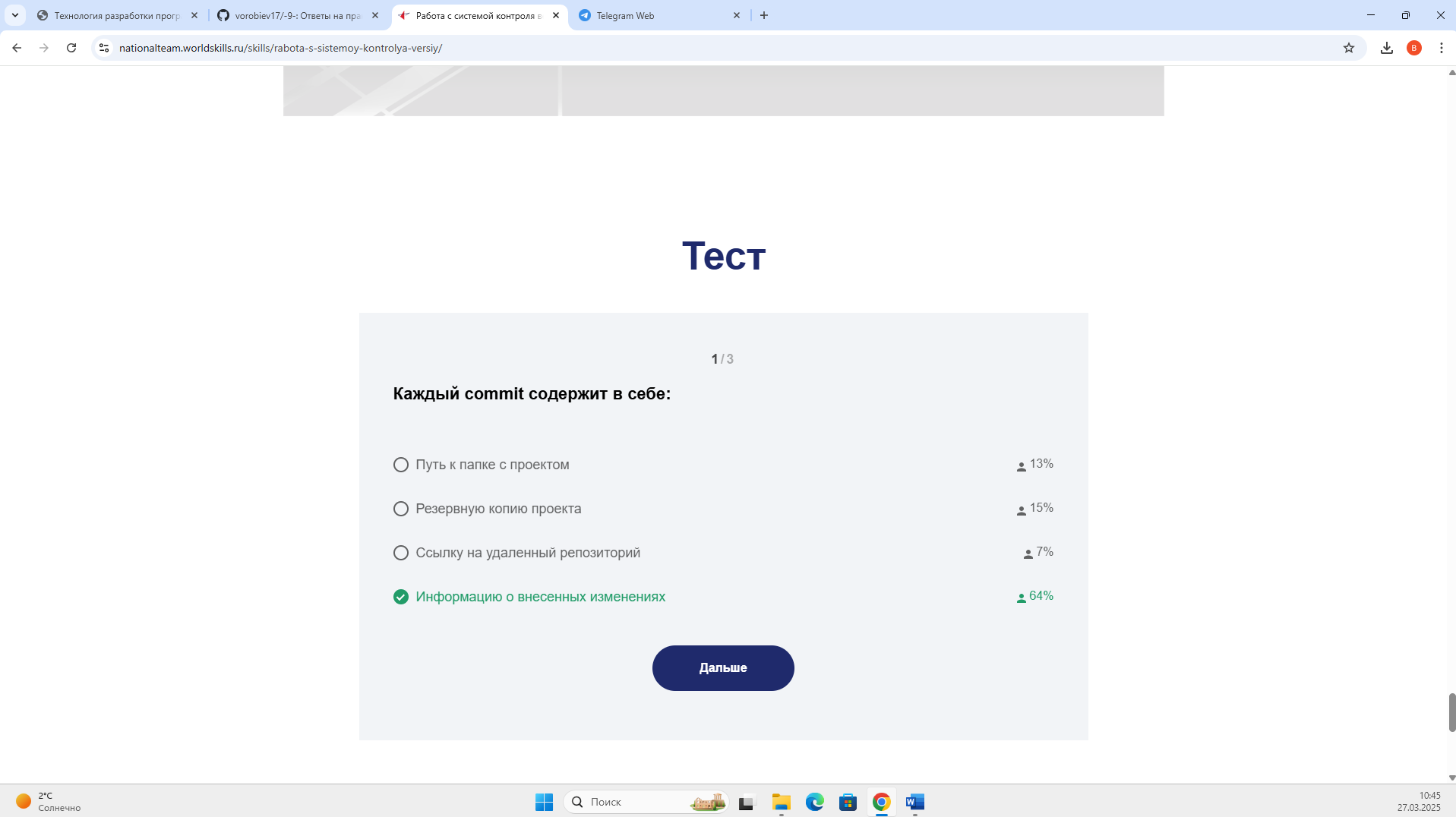


Рисунок 32 – 1 вопрос.

3. Ответы представлены ниже (Рисунок 33)

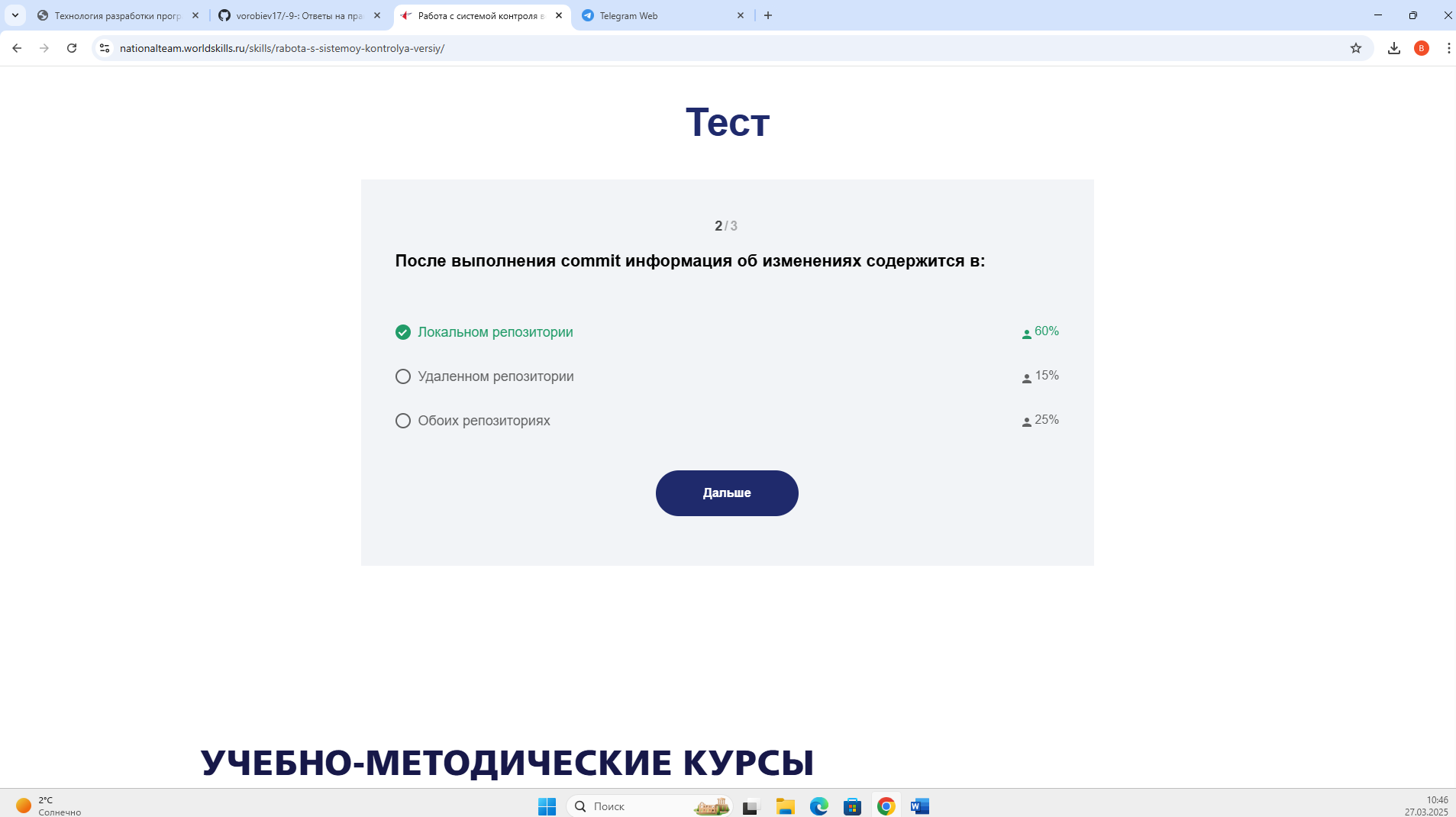


Рисунок 33 – 2 вопрос.

4. Ответы представлены ниже (Рисунок 34).

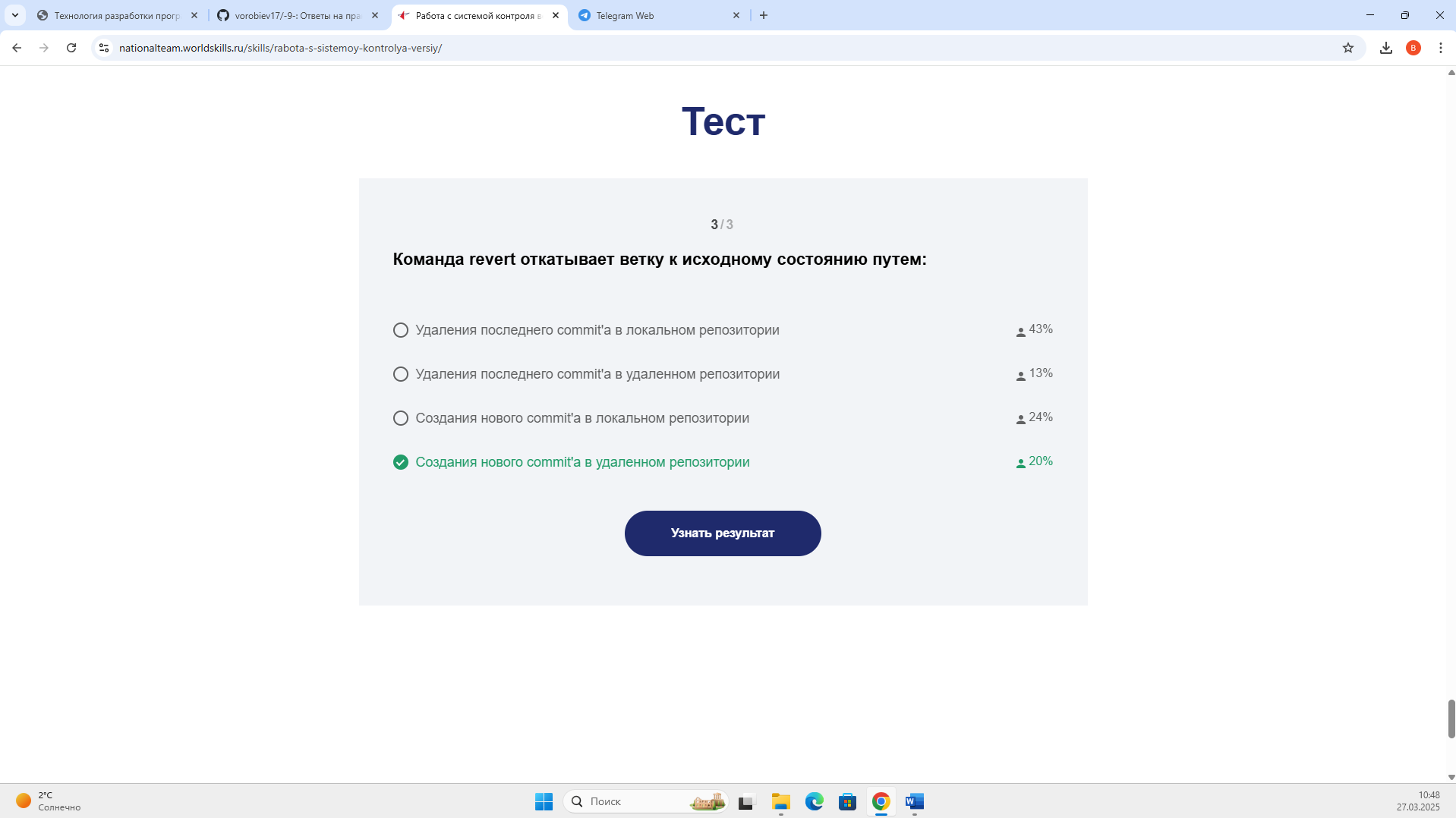


Рисунок 34 – 3 вопрос.

5. Результат (Рисунок 35).

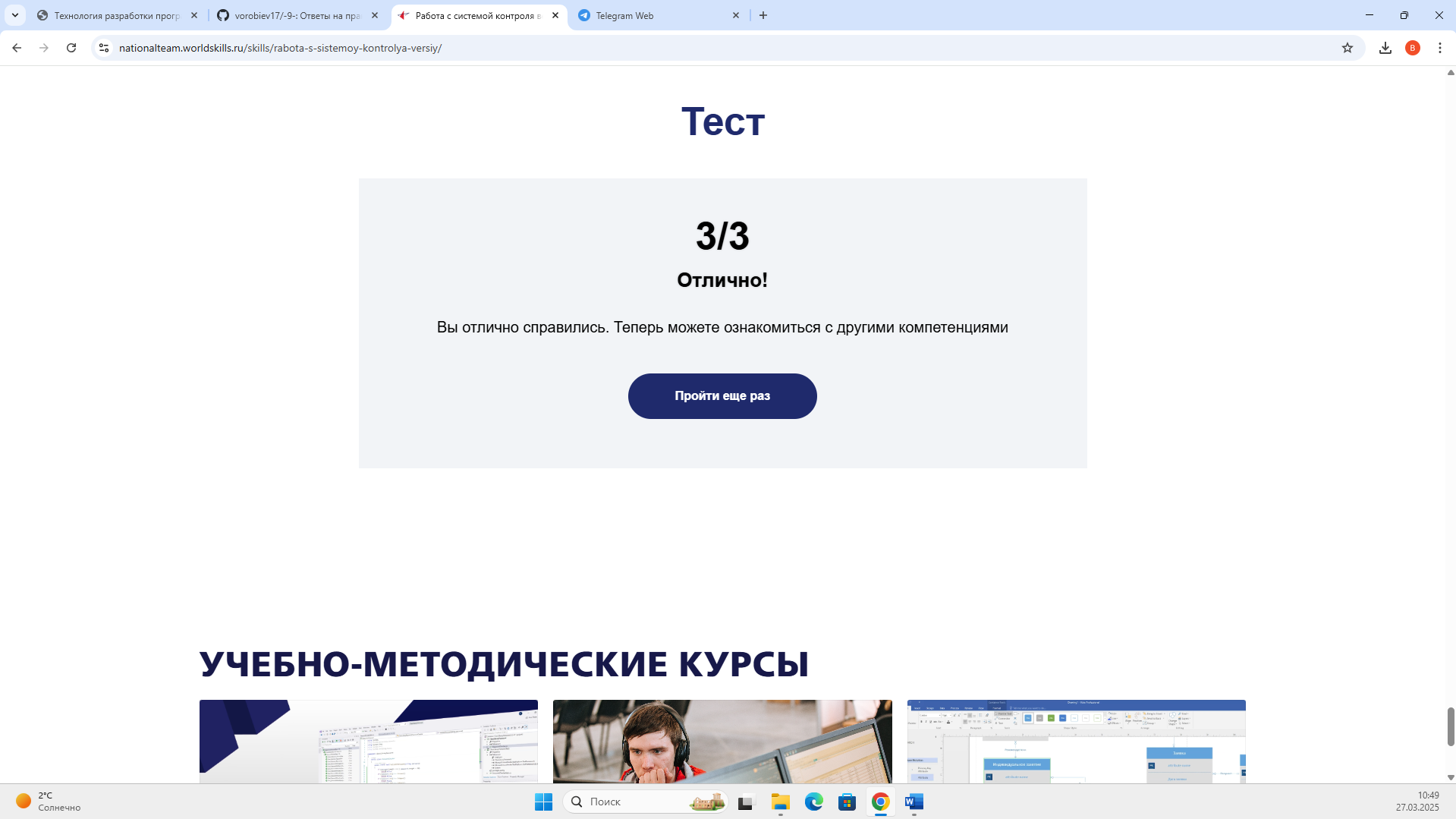


Рисунок 35 – результат

**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.