«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:** Матвеева Анжелика, студент ГБПОУ «ЧРТ» группы ПР-273

**Ход работы:**

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.
6. Зарегистрироваться на Github.
7. Создать локальный каталог для выполнения заданий по предмету.
8. **Заполните основные данные на**[**https://github.com**](https://github.com/).

Поскольку аккаунт уже был создан ранее, просто захожу в него.

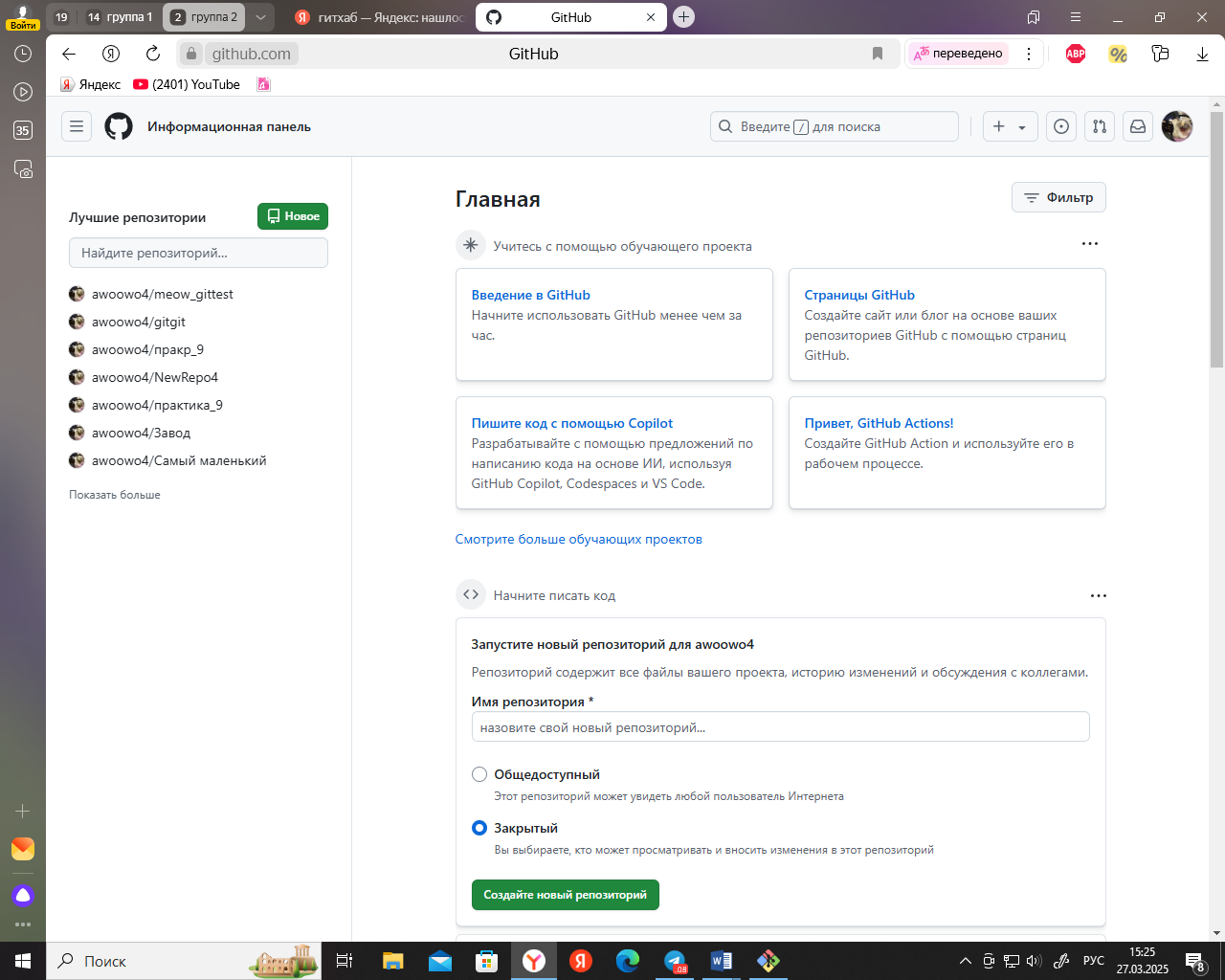


Рисунок 1 – Главная страница GitHub

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

Скачиваем Git:

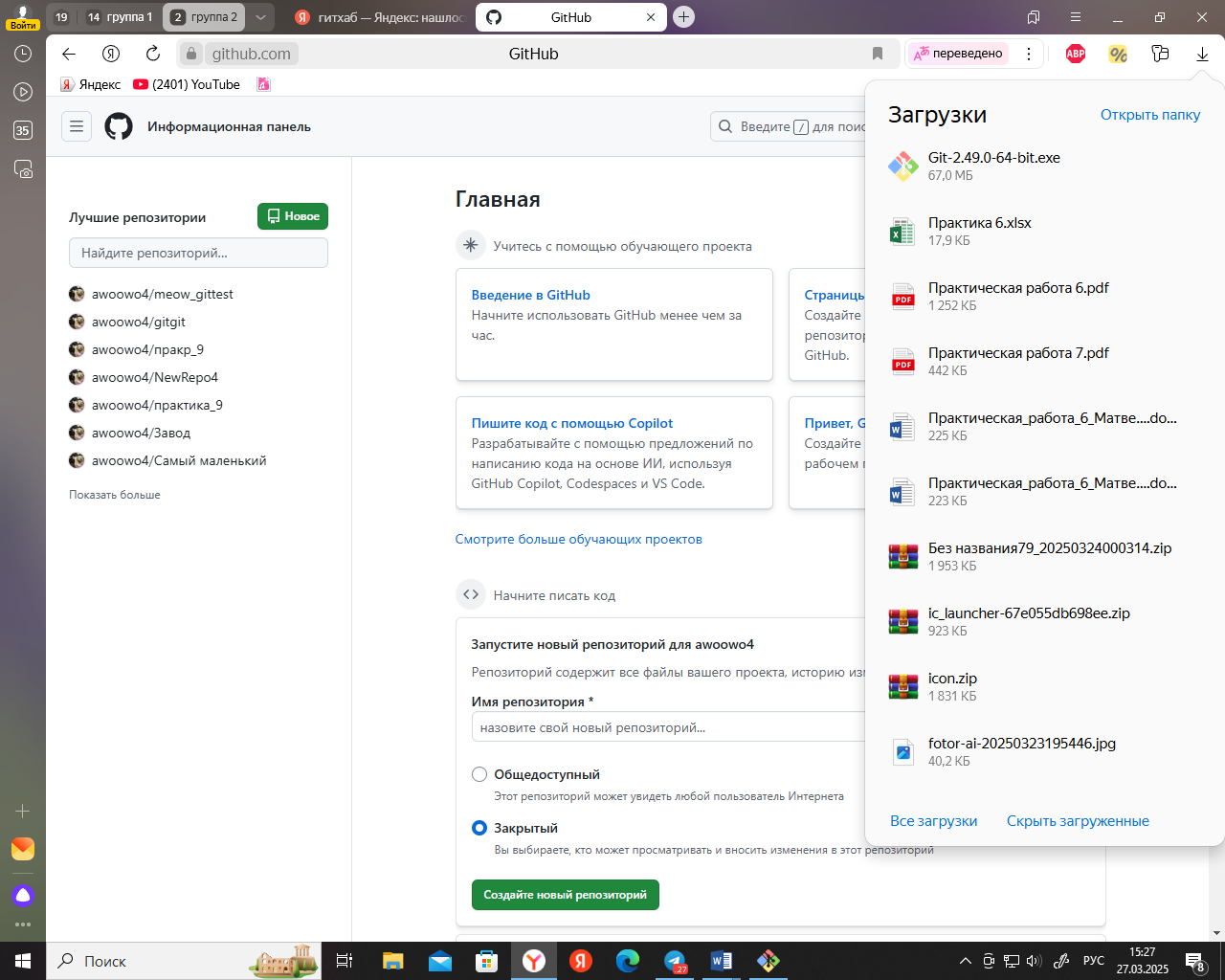


Рисунок 2 - Установщик

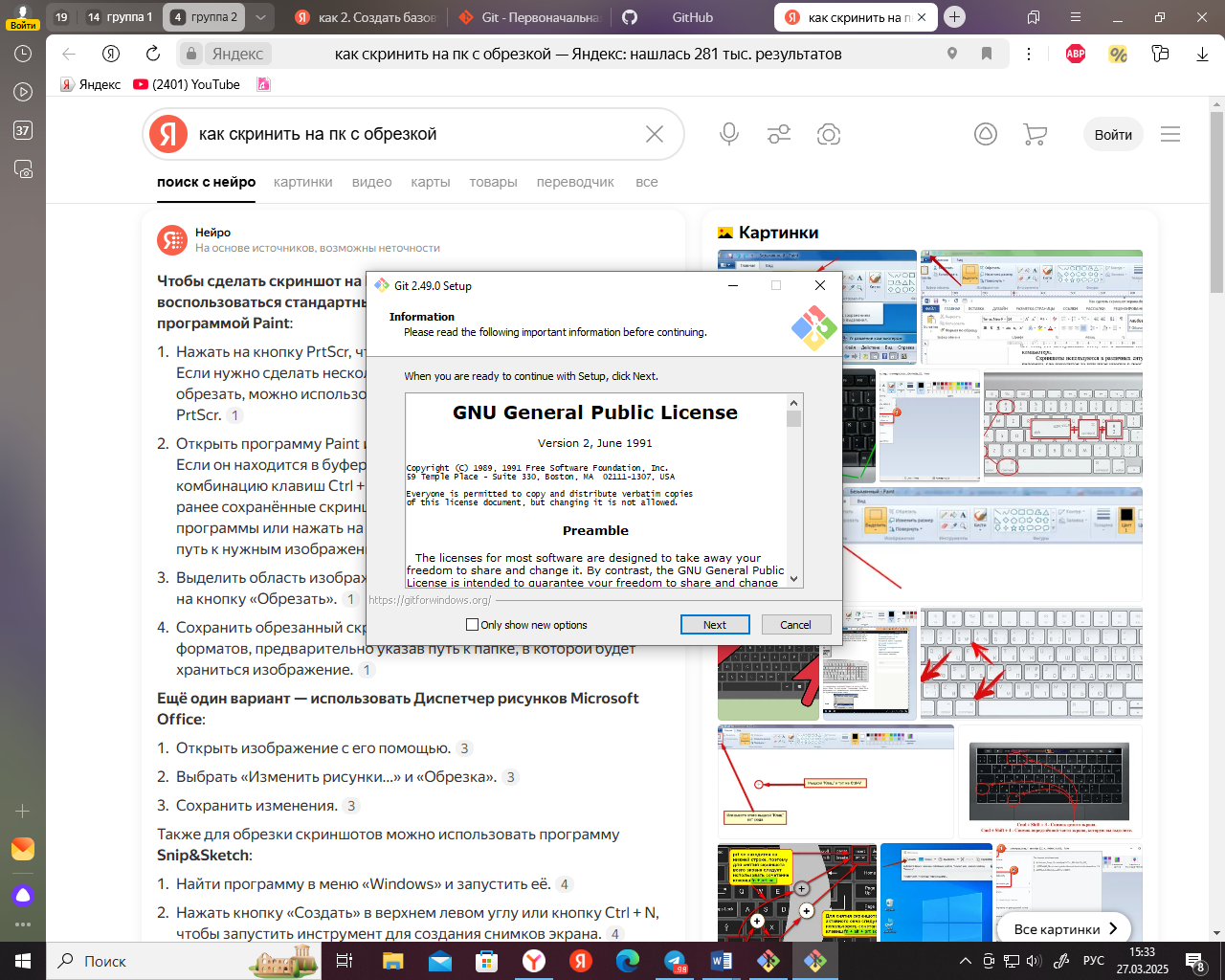


Рисунок 3 – Установщик

Открываем git cmd и работаем с конфигурацией.

Конфигурация git уже автоматически создана. Для ее просмотра вводим в консоль команду:

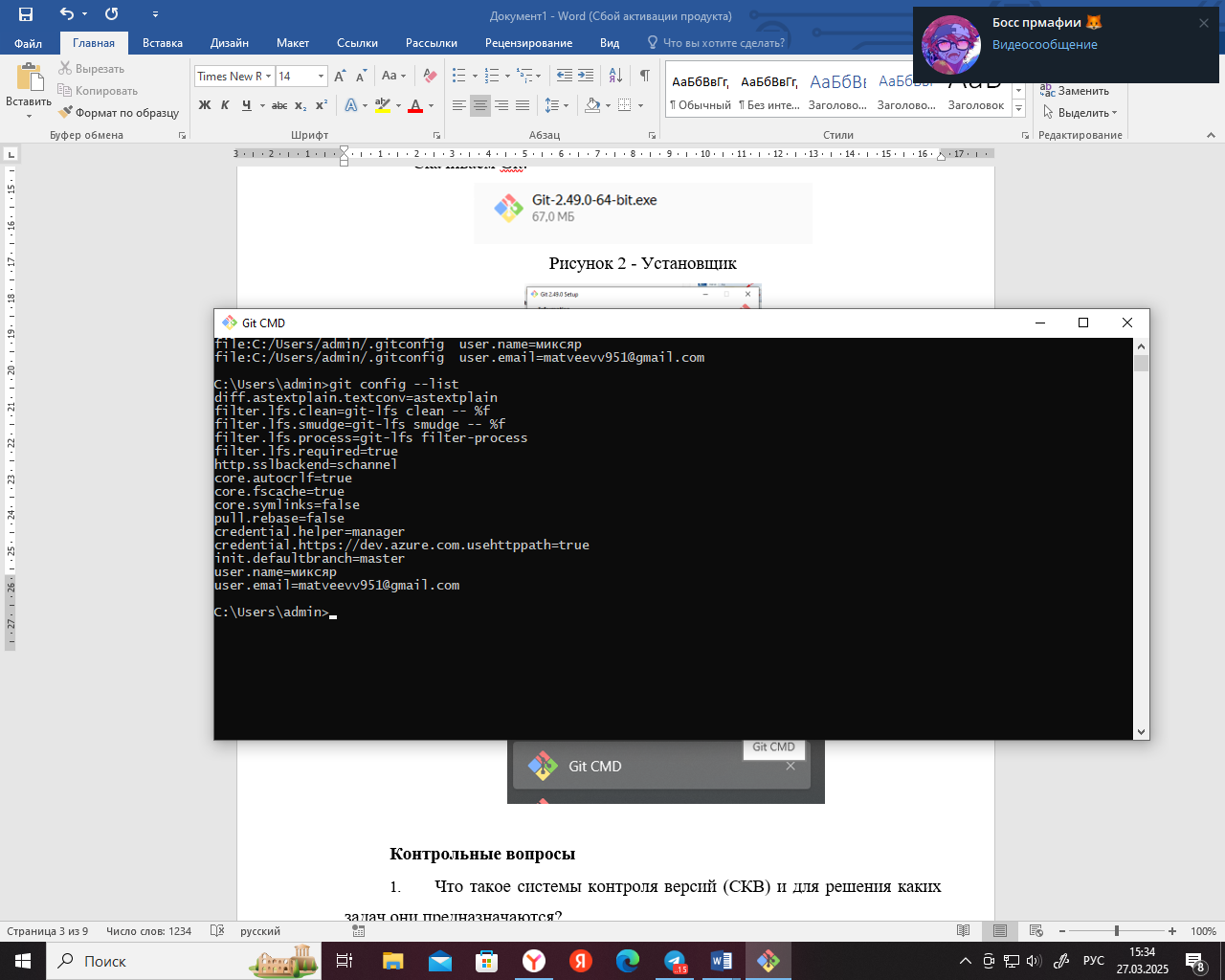


Рисунок 4 – Команда для просмотра конфигурации

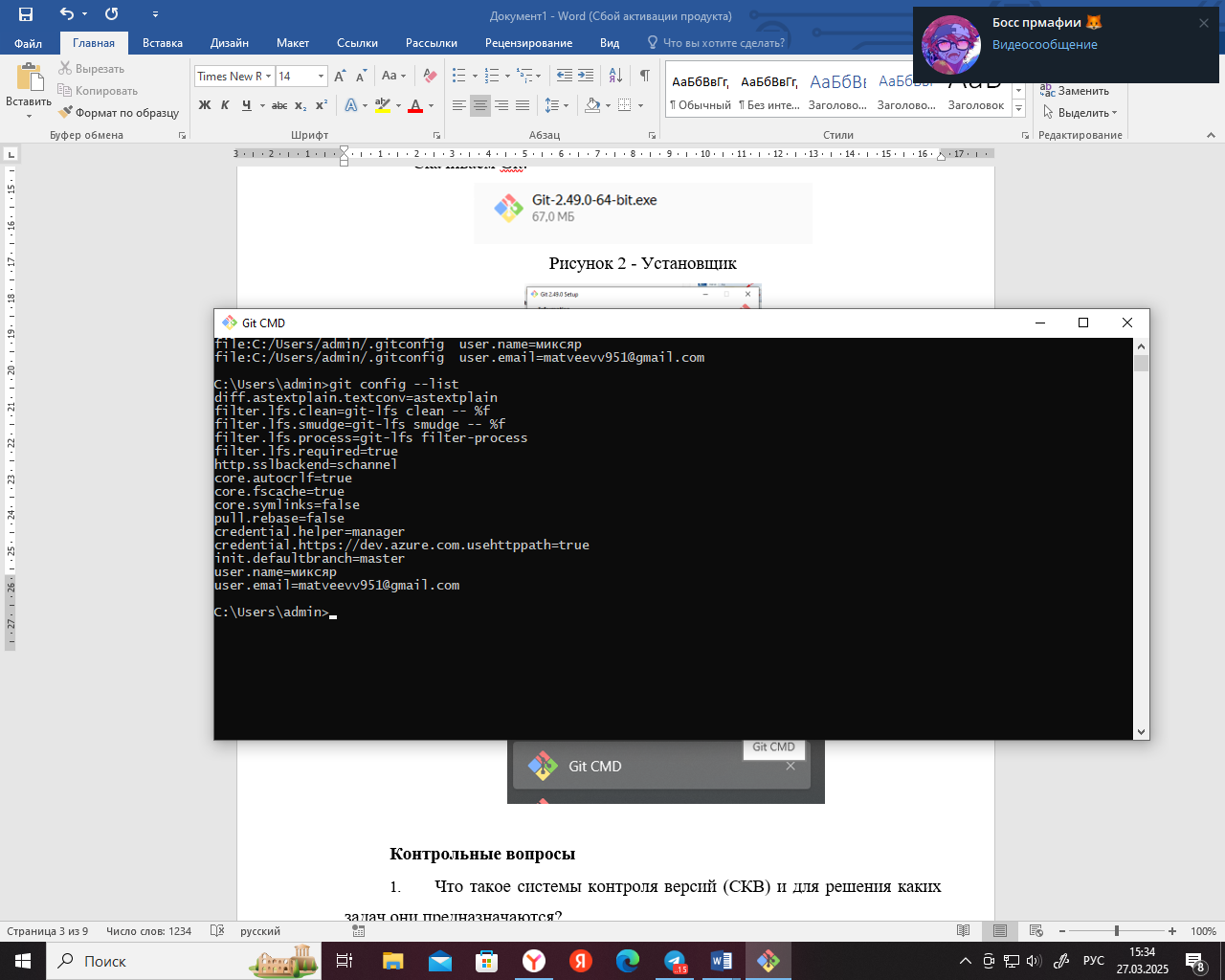


Рисунок 5 – Настроенная конфигурация

1. **Создать ключ SSH.**

Создается ключ с помощью команды:

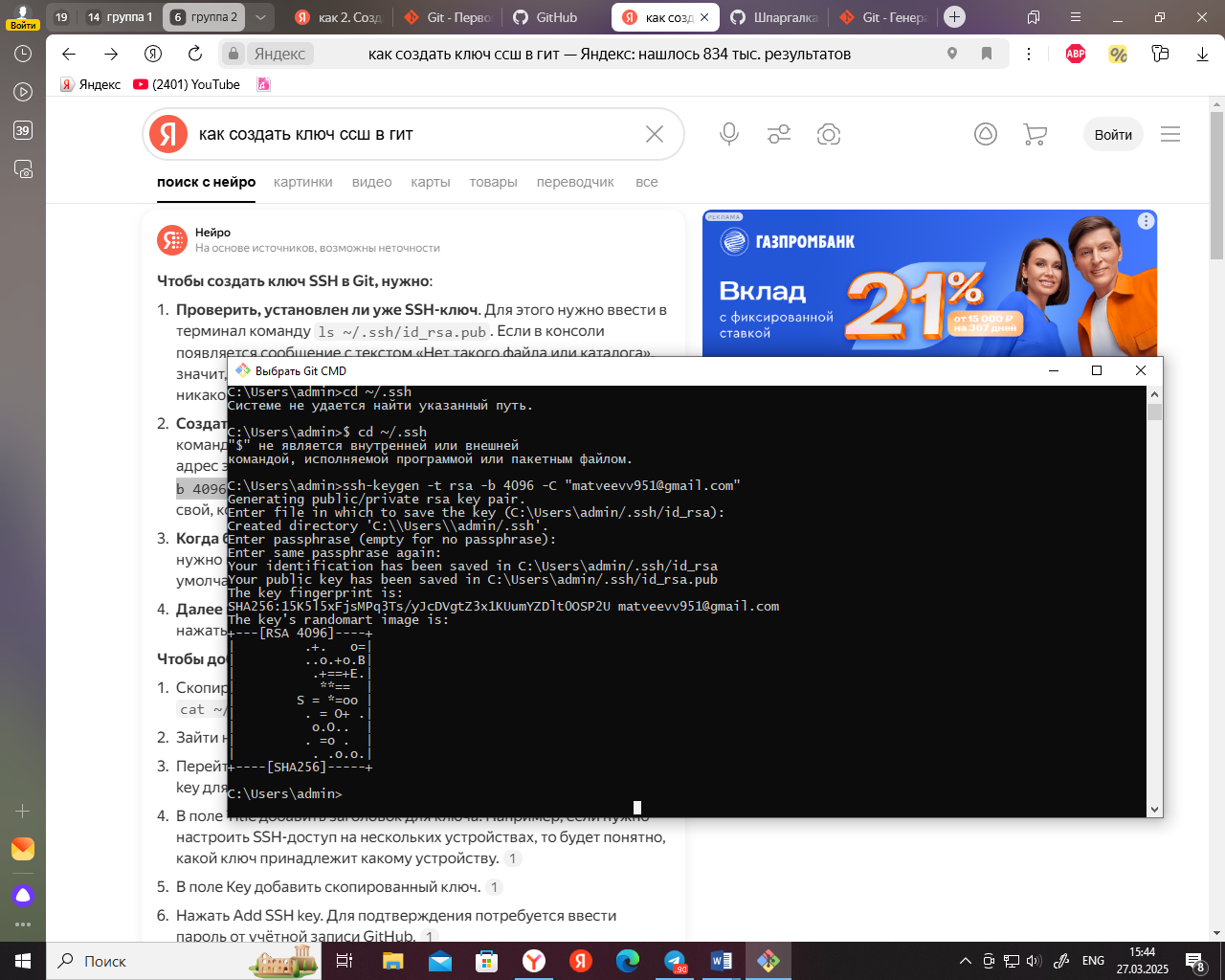


Рисунок 6 – Команда создания ключа SSH

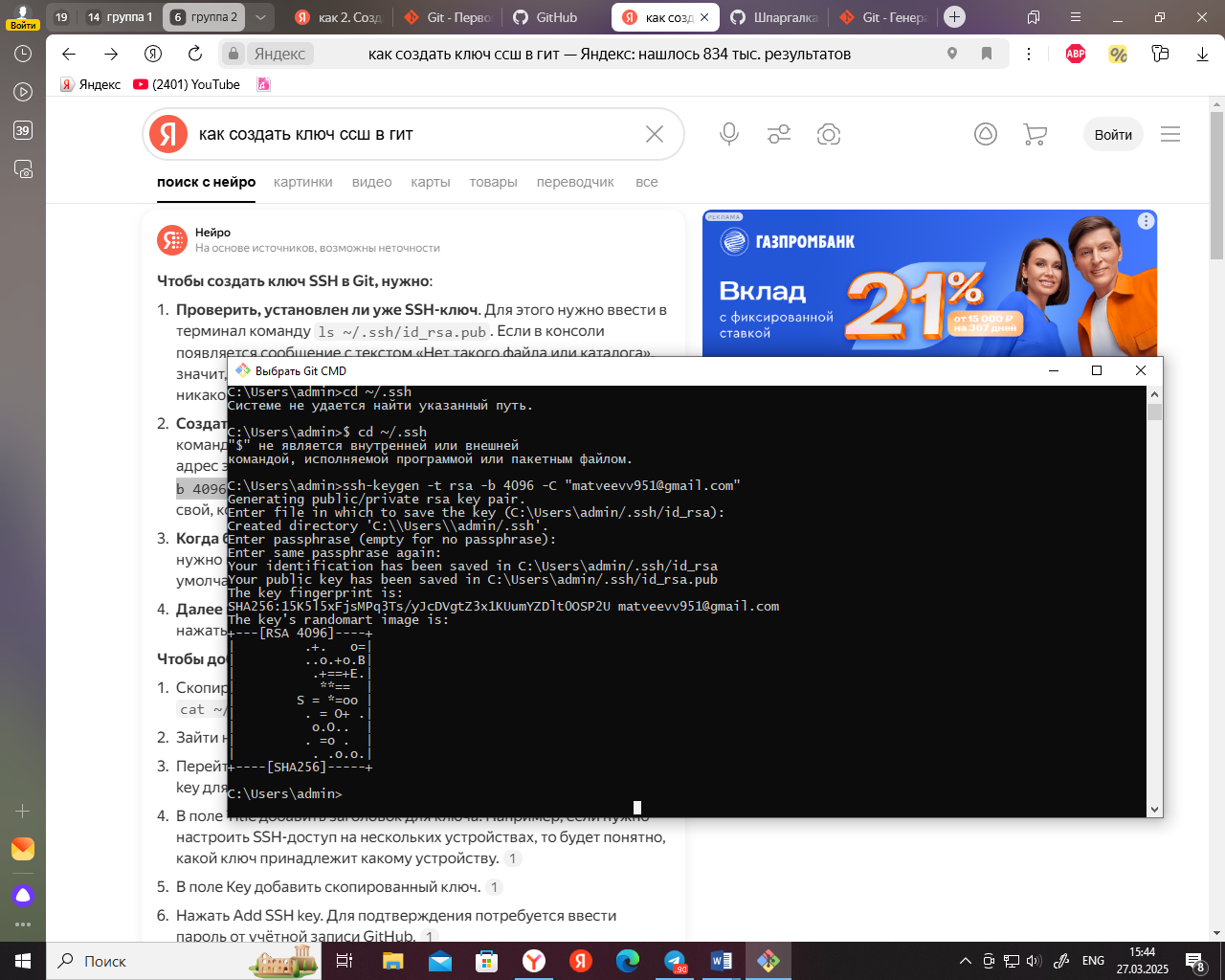


Рисунок 7 – Создание ключа SSH

Теперь добавляем SSH ключ в учетную запись GitHub. Для этого вводим в терминал команду:

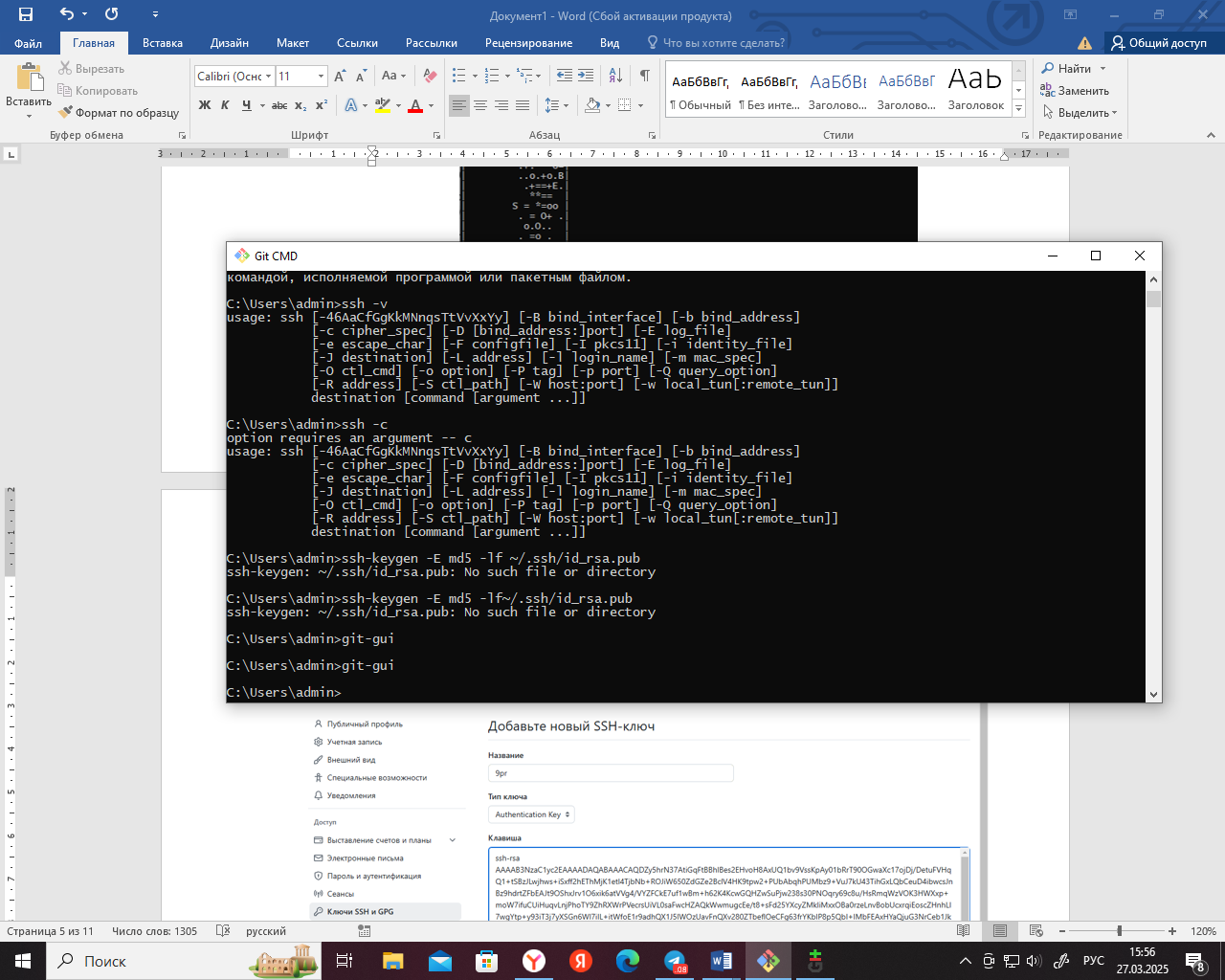


Рисунок 8 – Команда

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

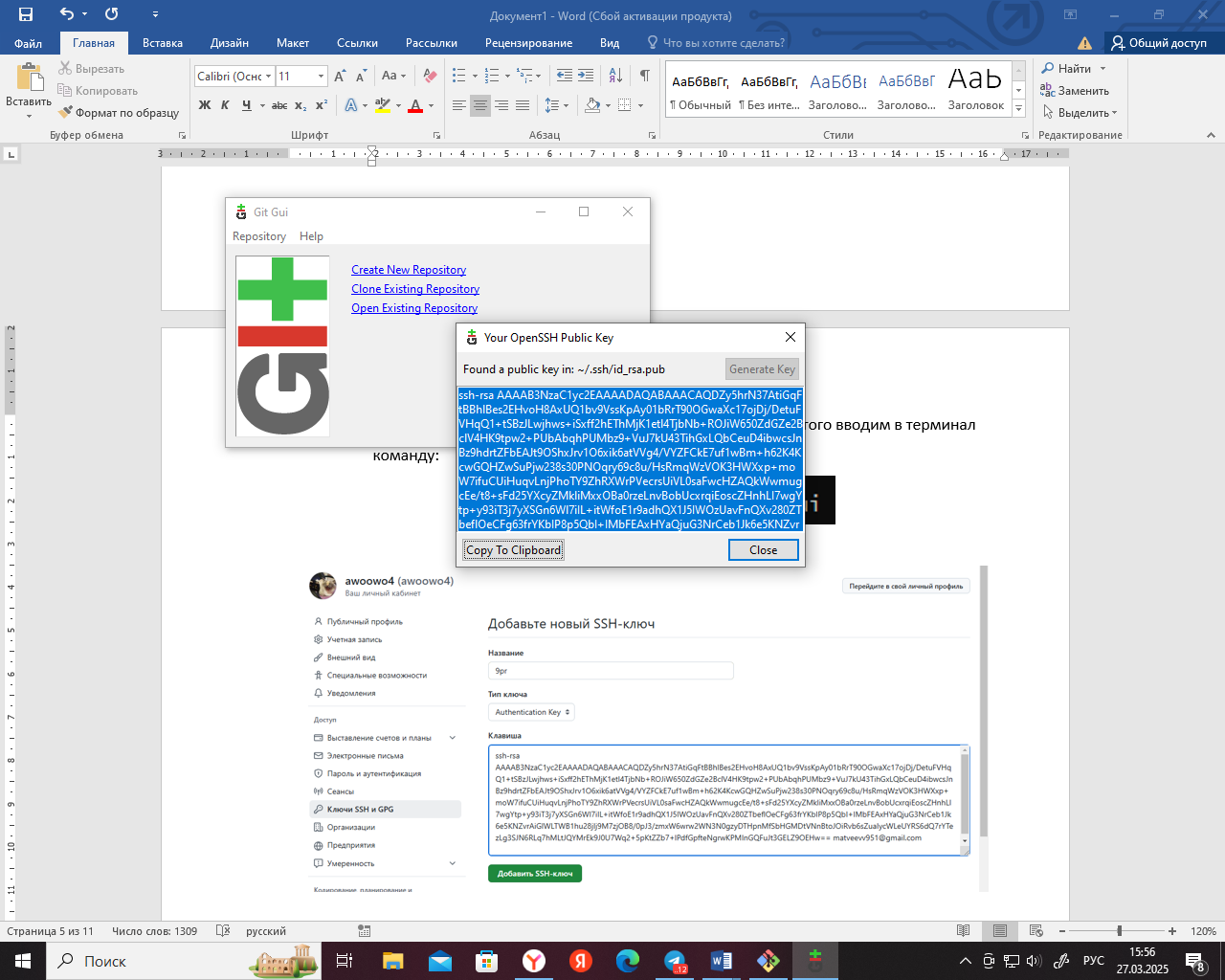


Рисунок 9 – Копируем ключ

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

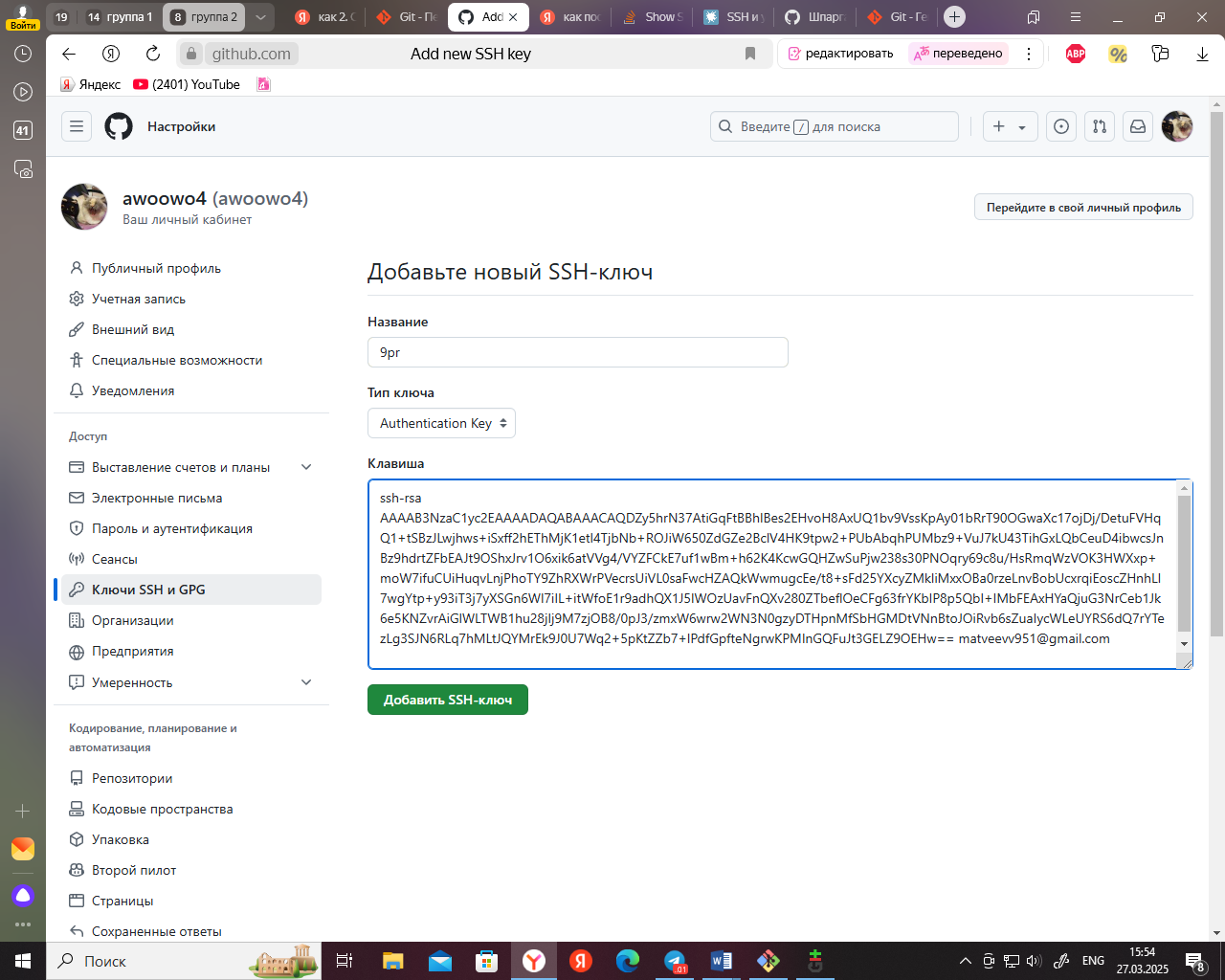


Рисунок 10 – Вставляем ключ

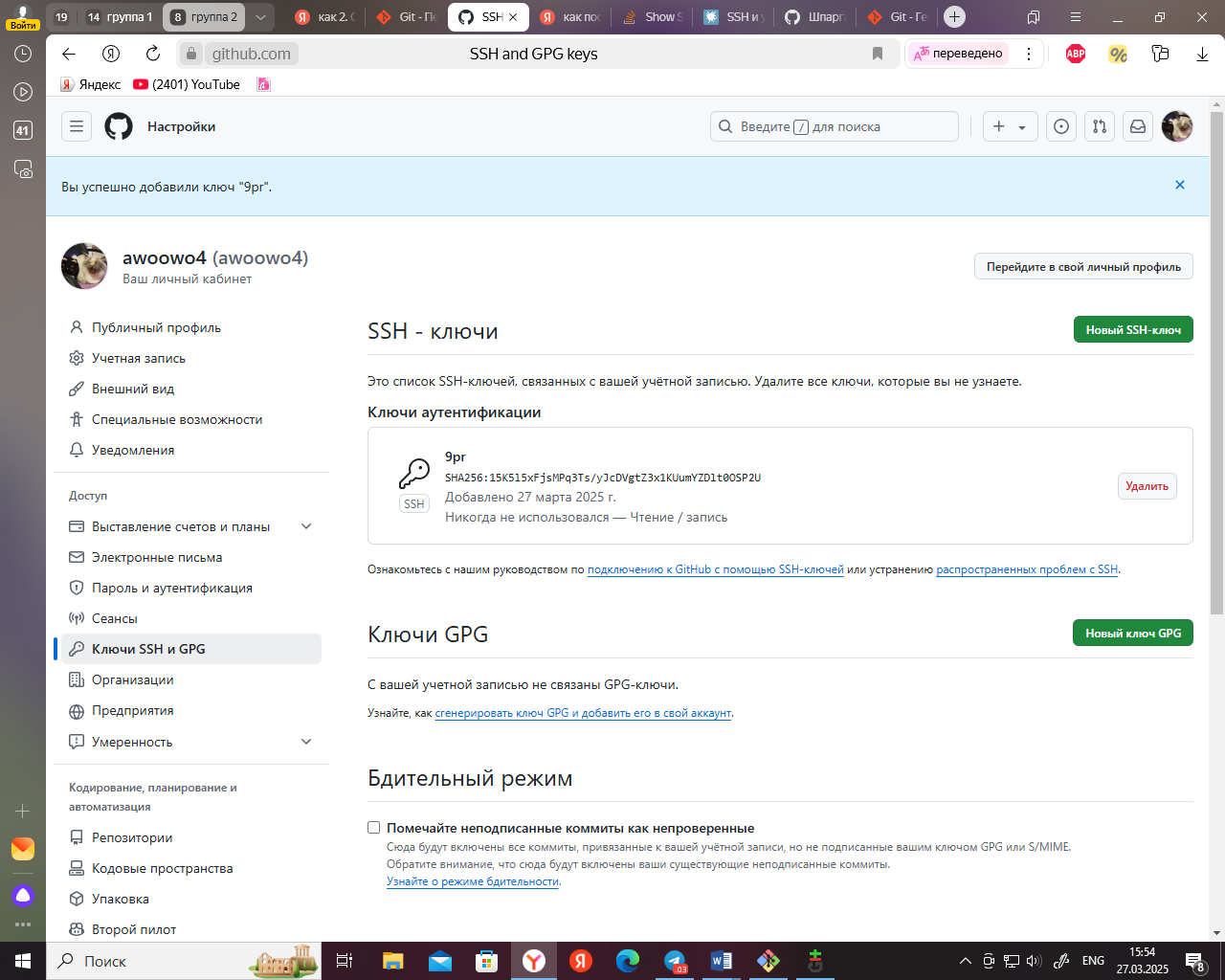


Рисунок 11 – Ключ создан

1. Создать ключ GPG.

Скачиваем GnuPG:

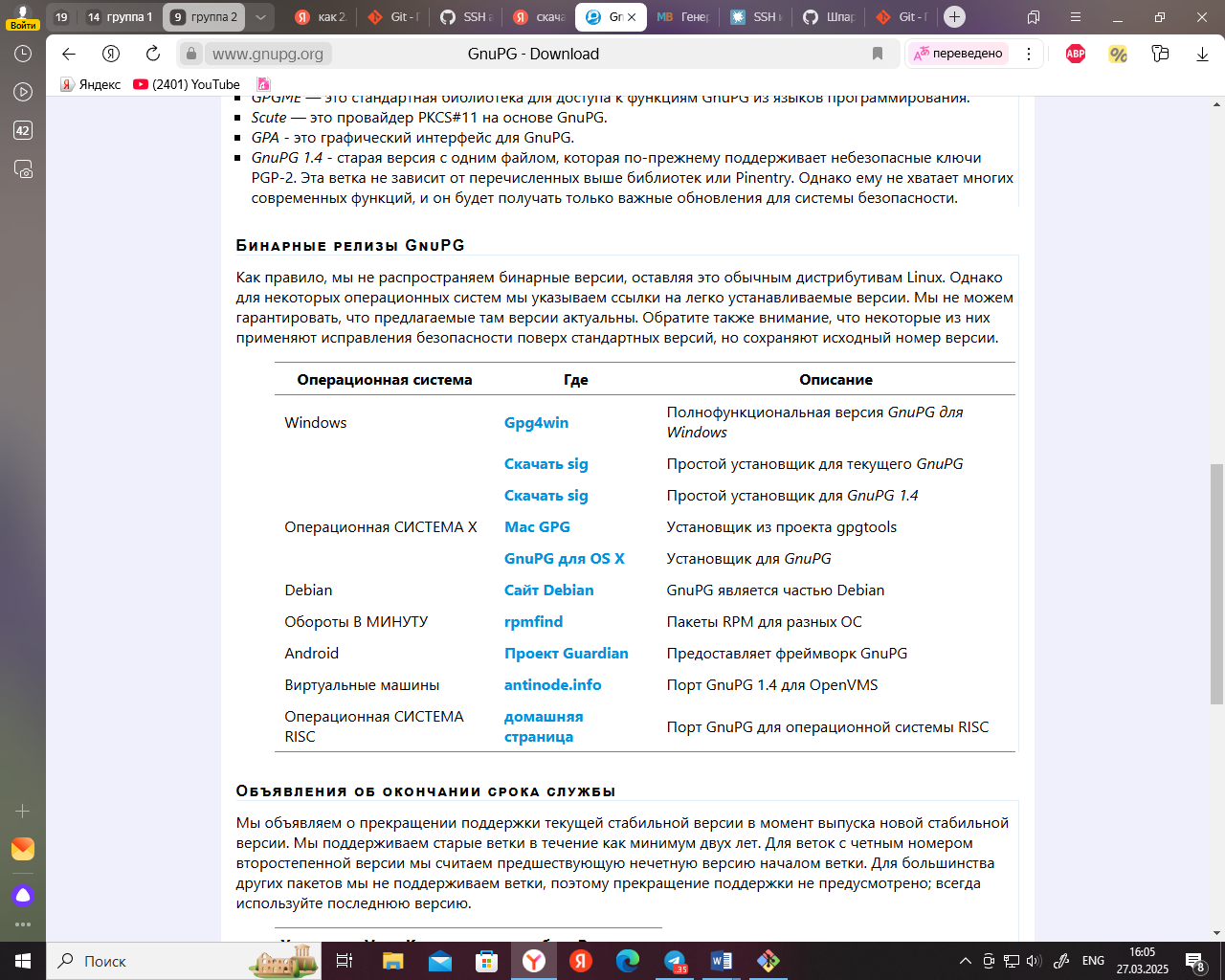


Рисунок 12 - установщик GnuPG

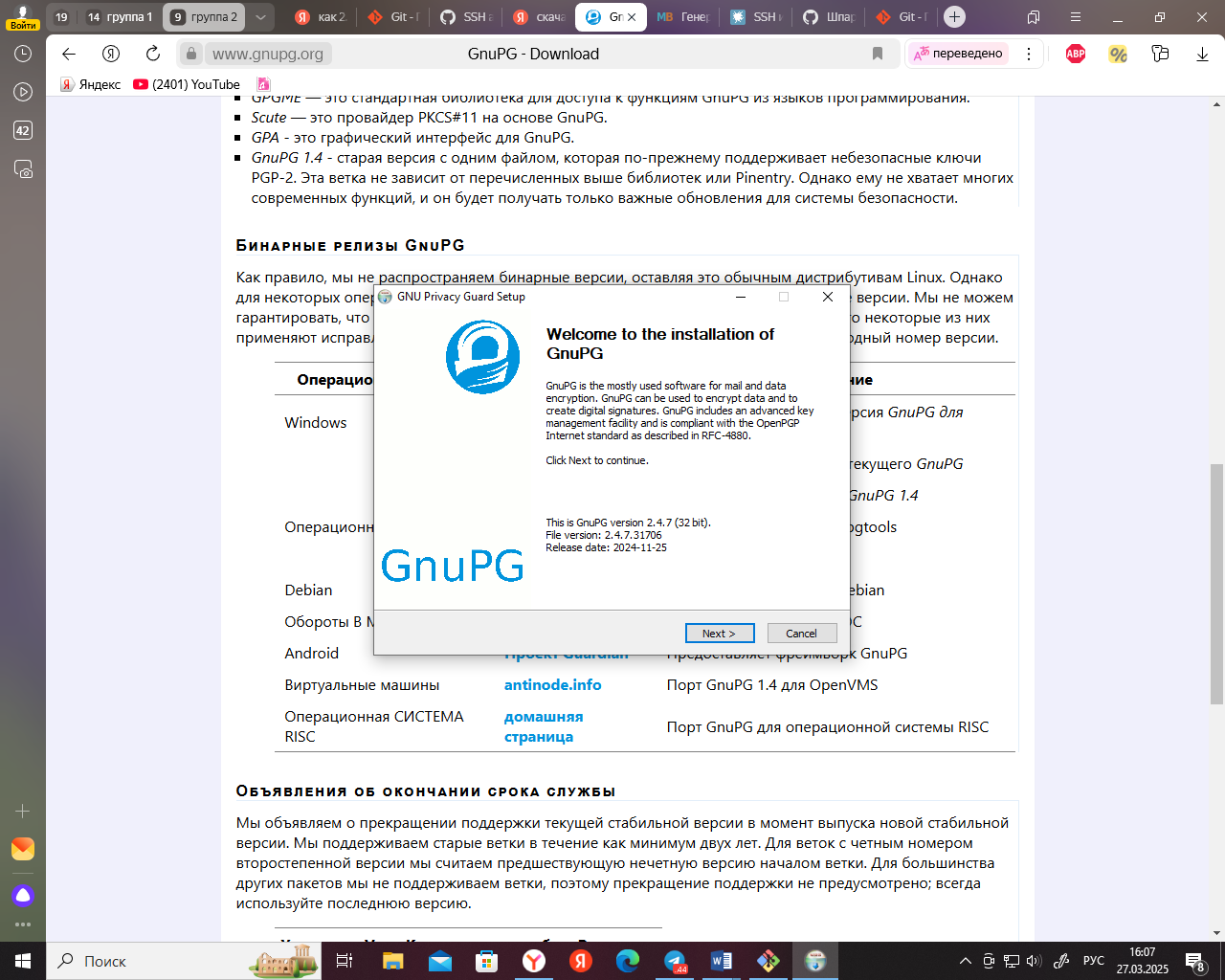


Рисунок 13 - установщик GnuPG

Для создания ключа вводим команду:

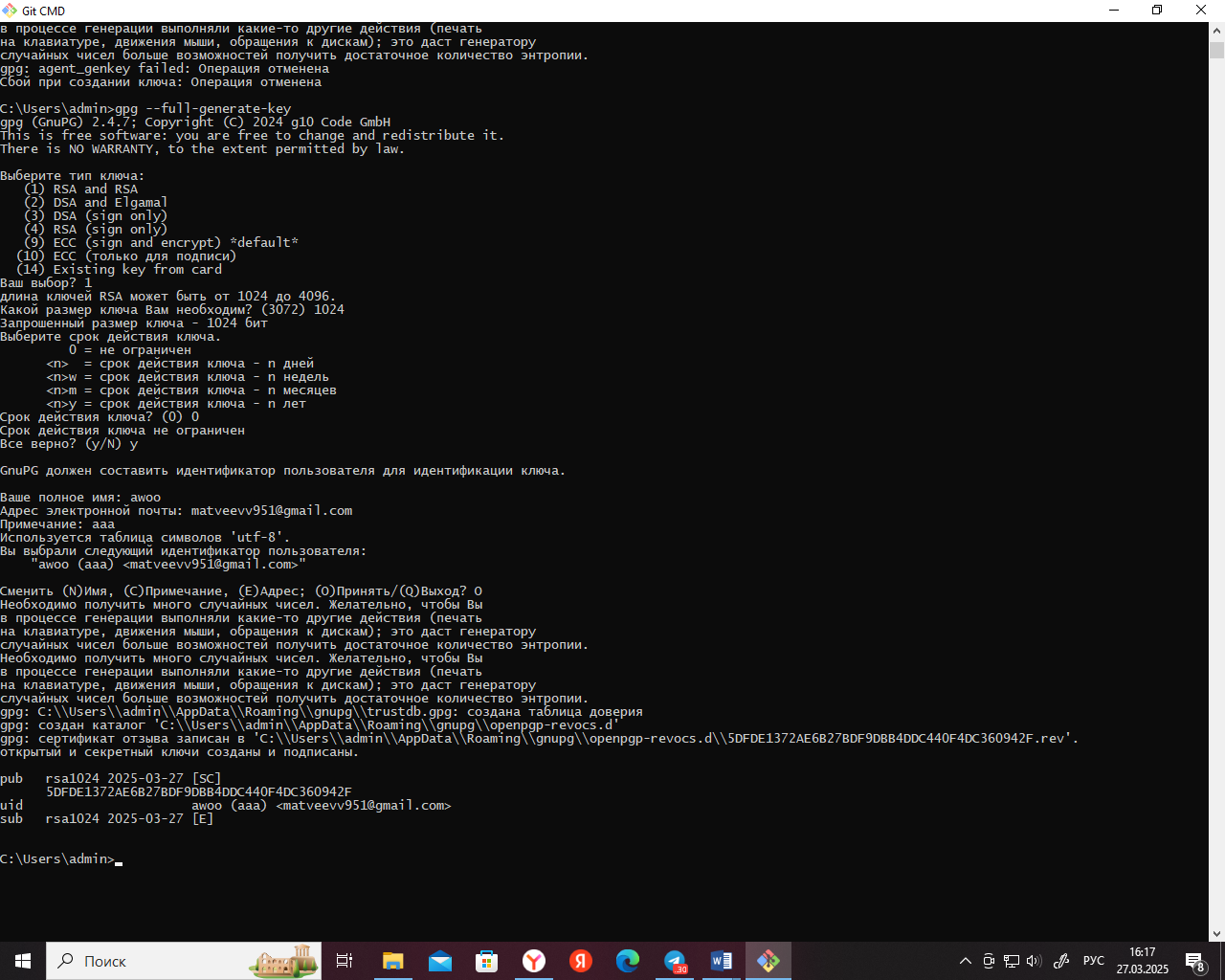


Рисунок 14 – Команда генерации GPG ключа

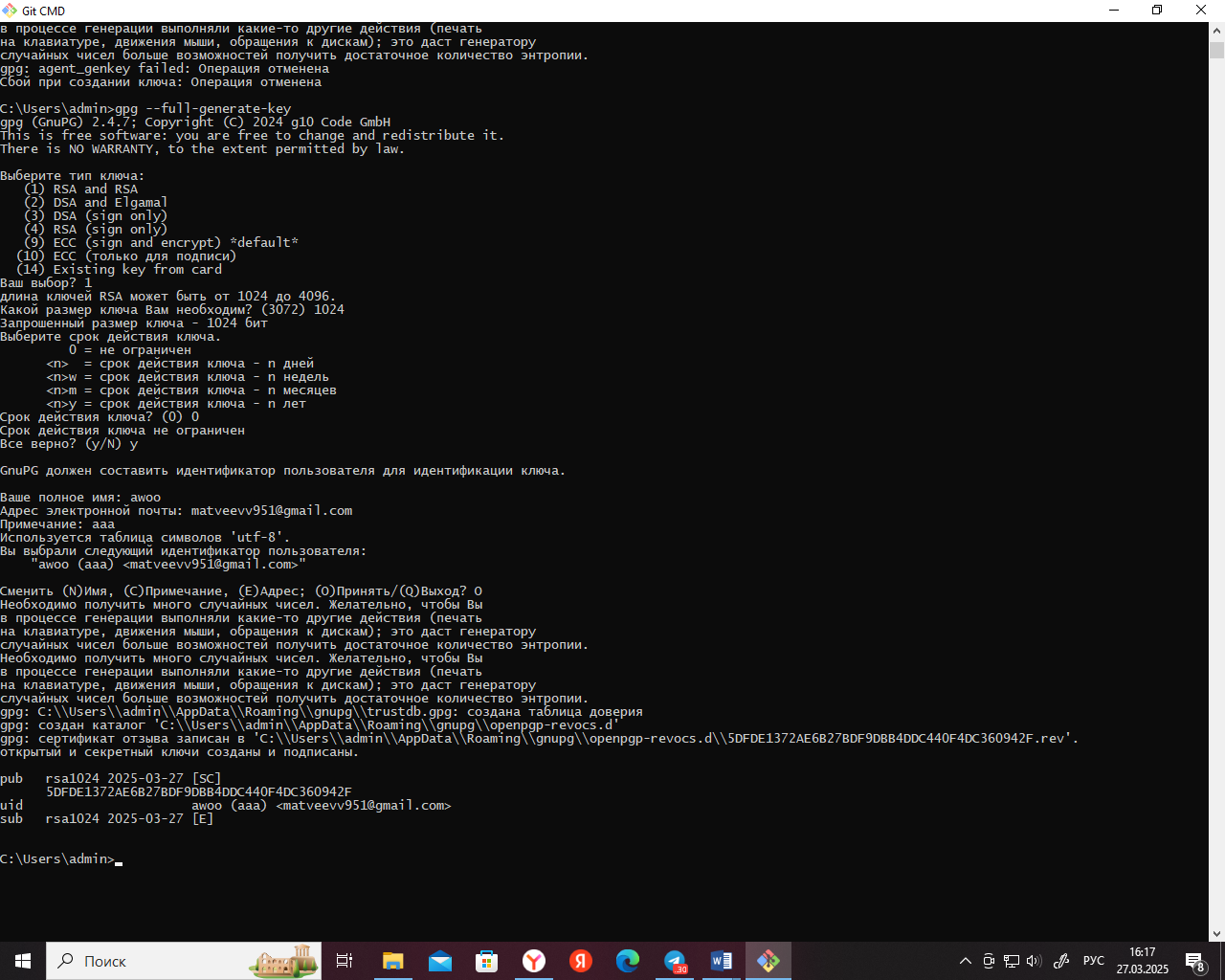


Рисунок 15 – Создание ключа

Чтобы добавить ключ GPG в учетную запись ГитХаба, необходимо получить GPG-ключ из терминала с помощью команд:

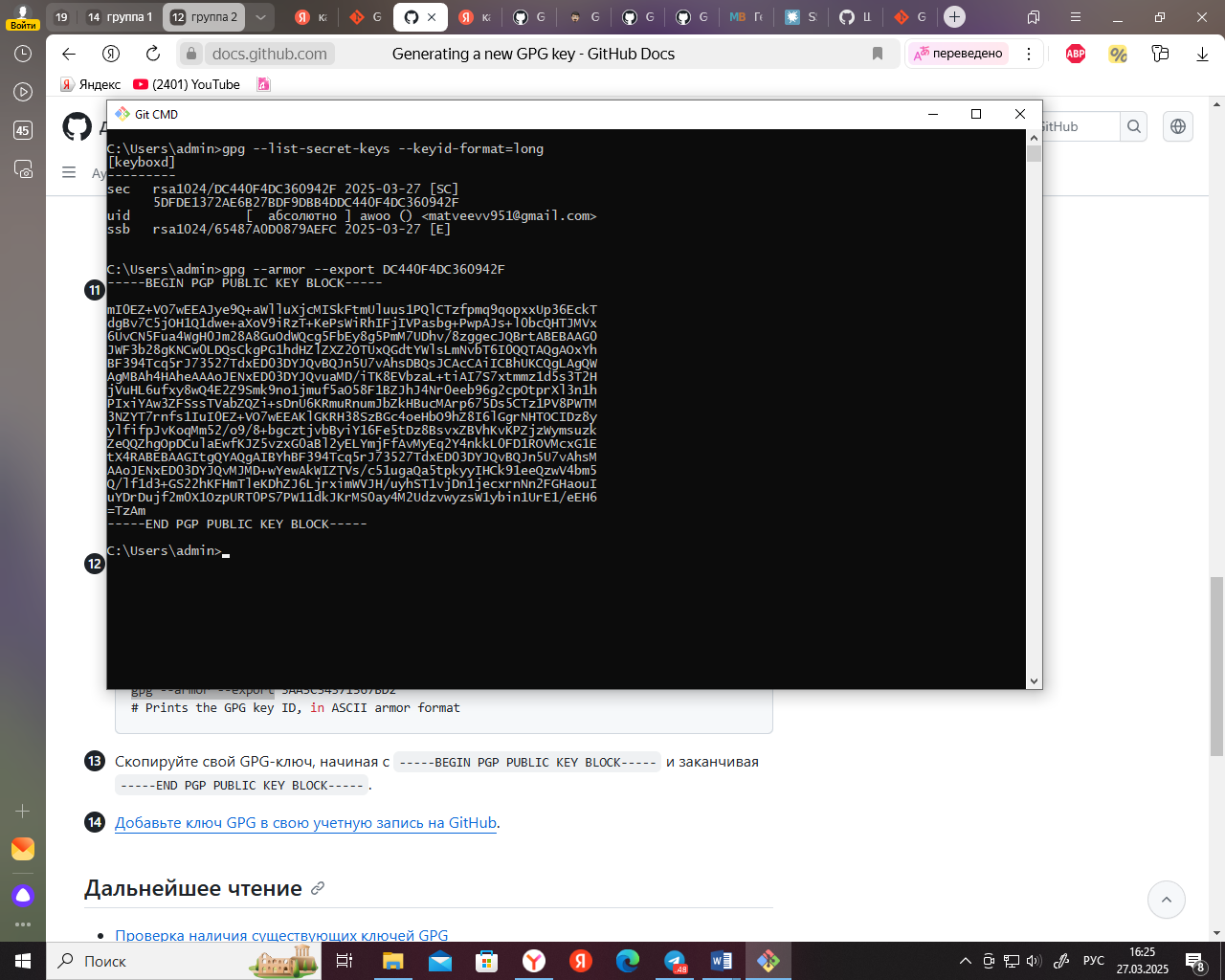


Рисунок 16 – Полный ключ

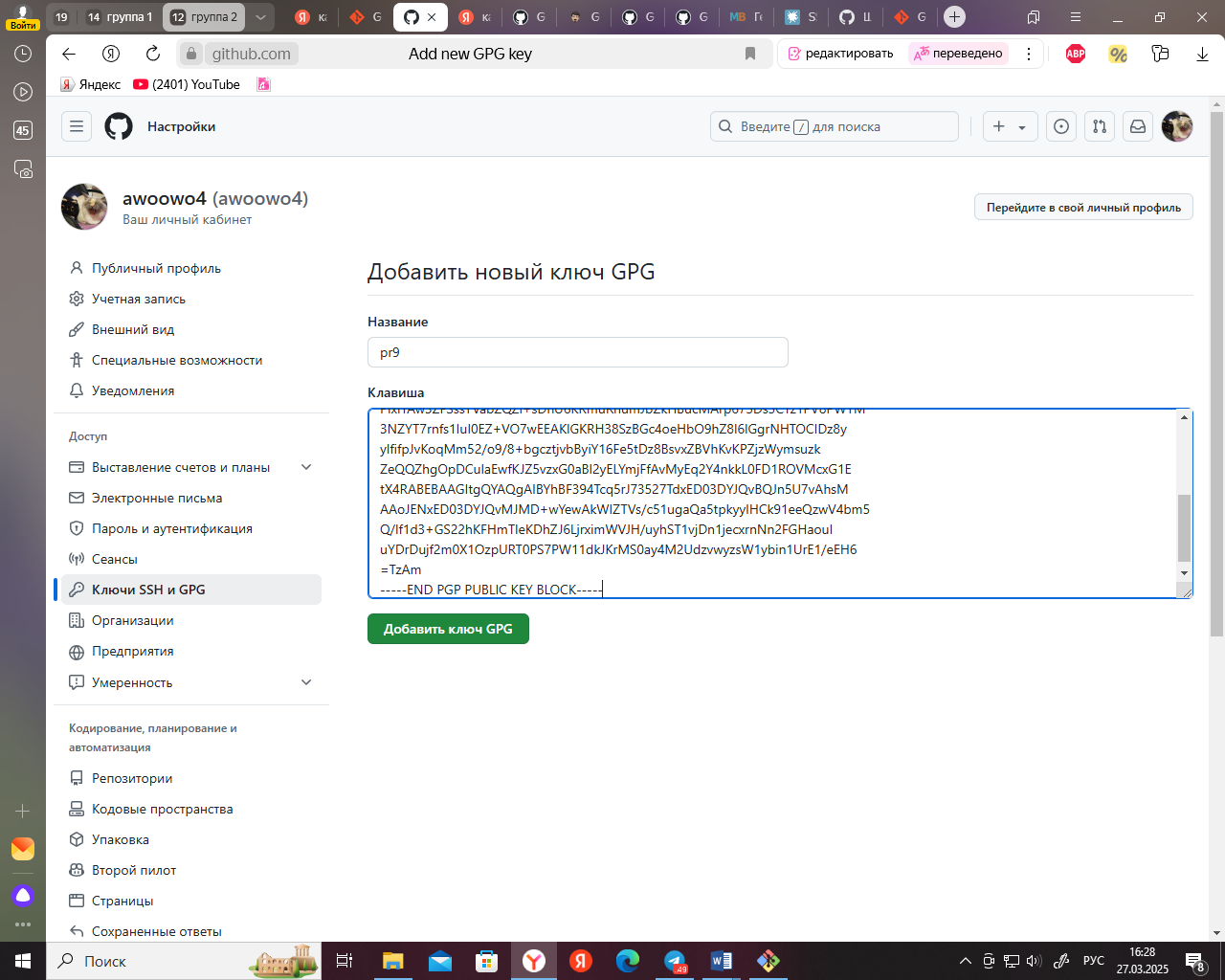


Рисунок 17 – Вставляем ключ

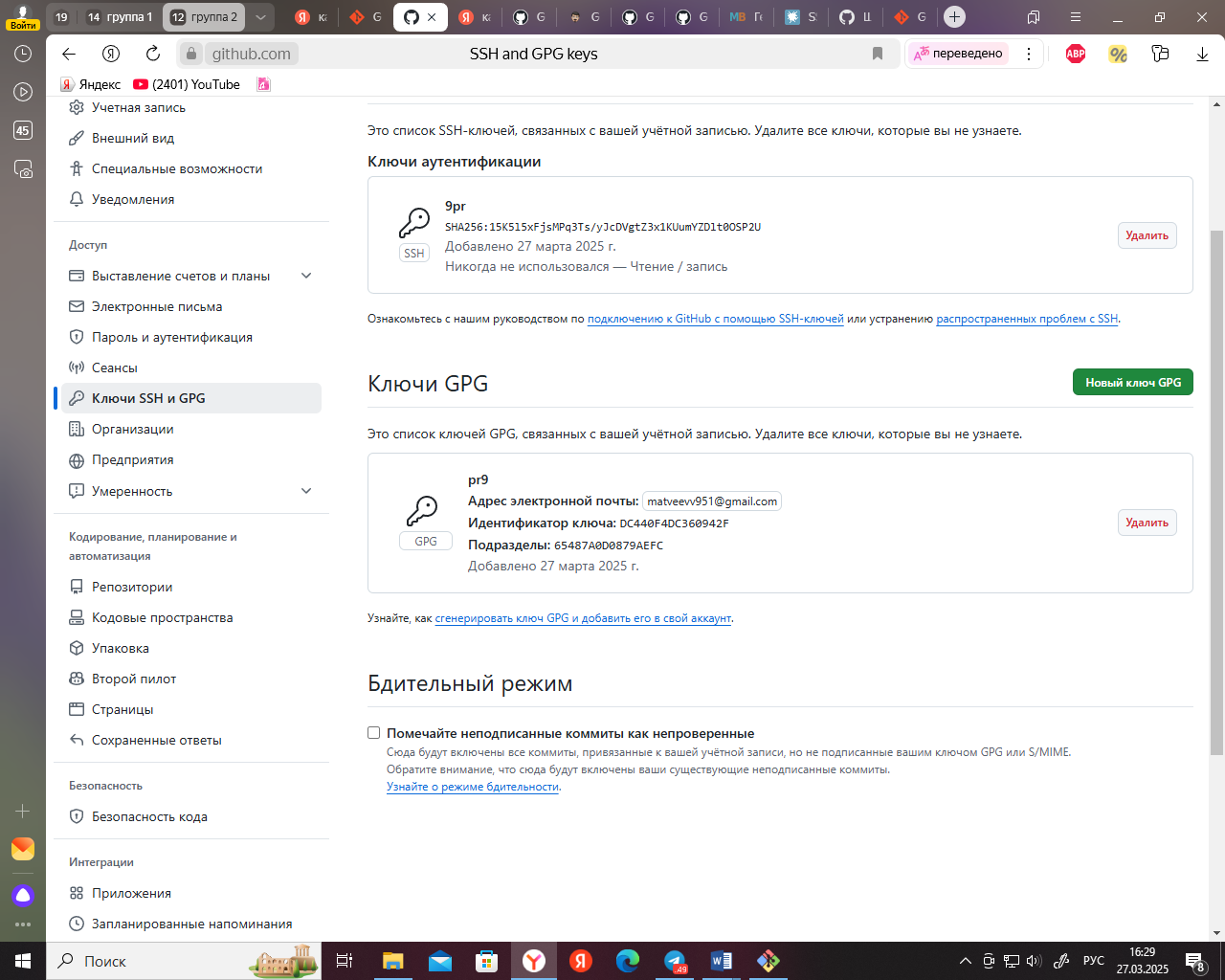


Рисунок 18 – Ключ создан

1. Настроить подписи git.

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

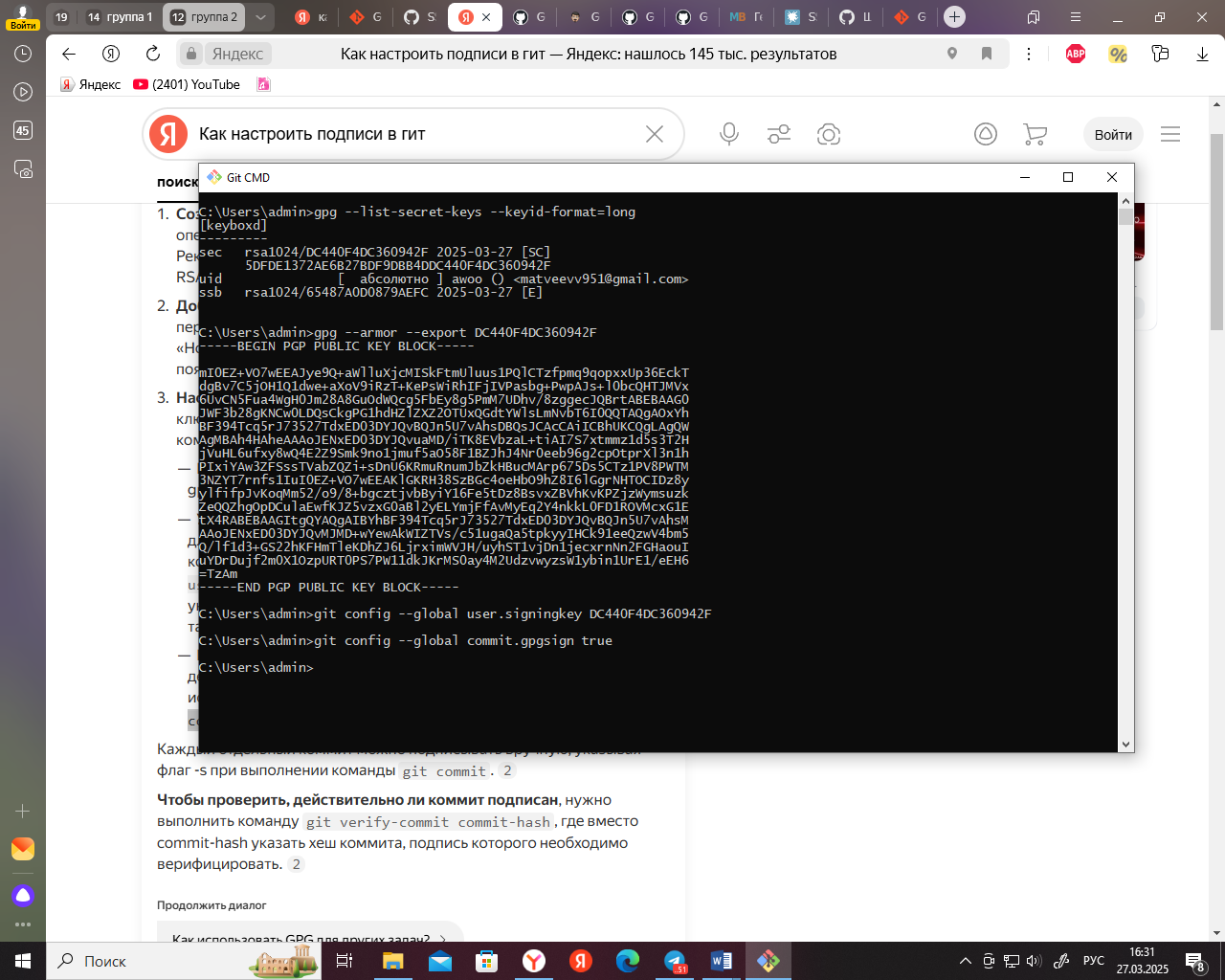


Рисунок 19 – Команды ключа по умолчанию и обязательной подписи коммитов

1. Зарегистрироваться на Github.

Уже зарегистрирована.

1. Создать локальный каталог для выполнения заданий по предмету.

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

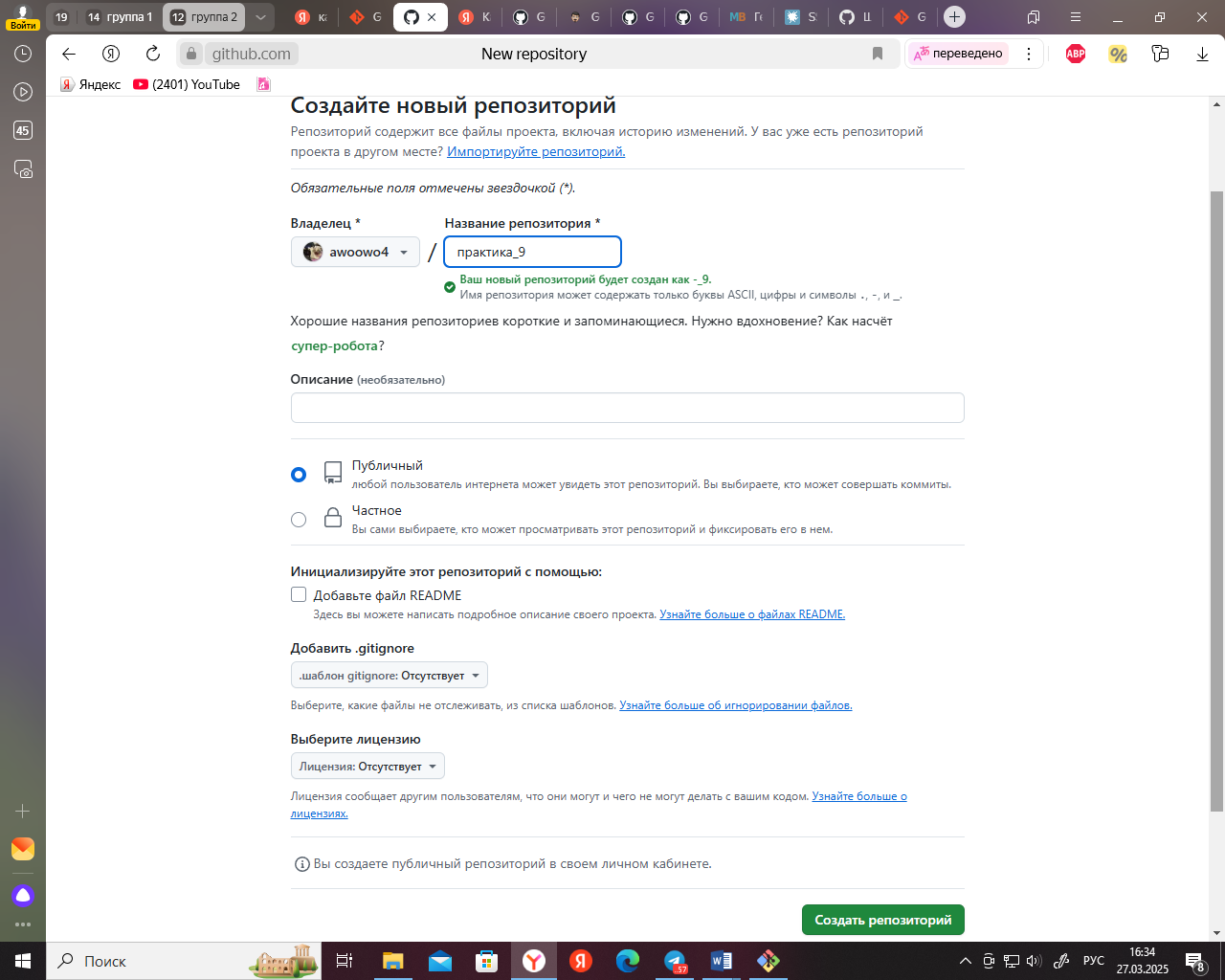


Рисунок 20 – Создание репозитория

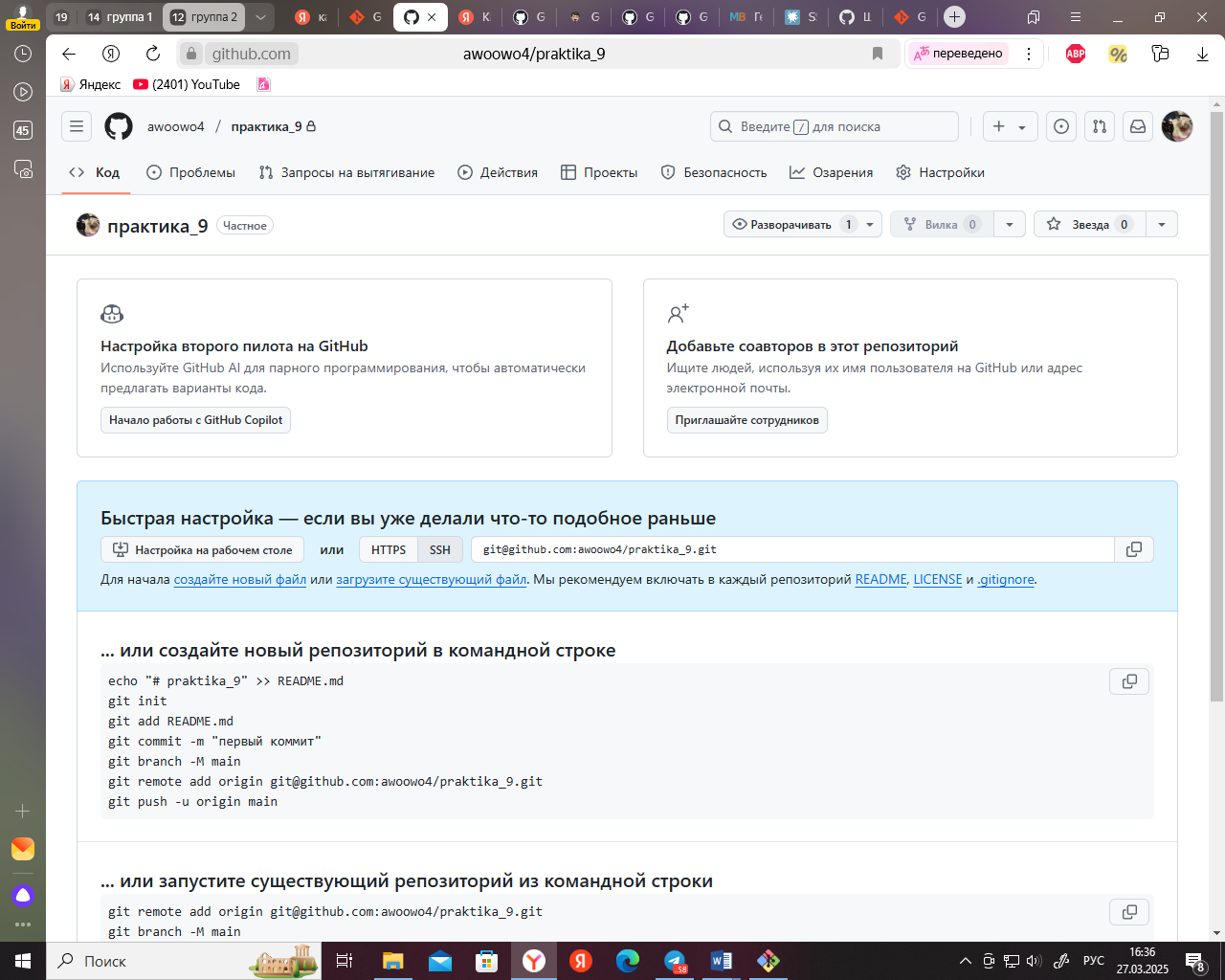


Рисунок 21 – Репозиторий создан

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

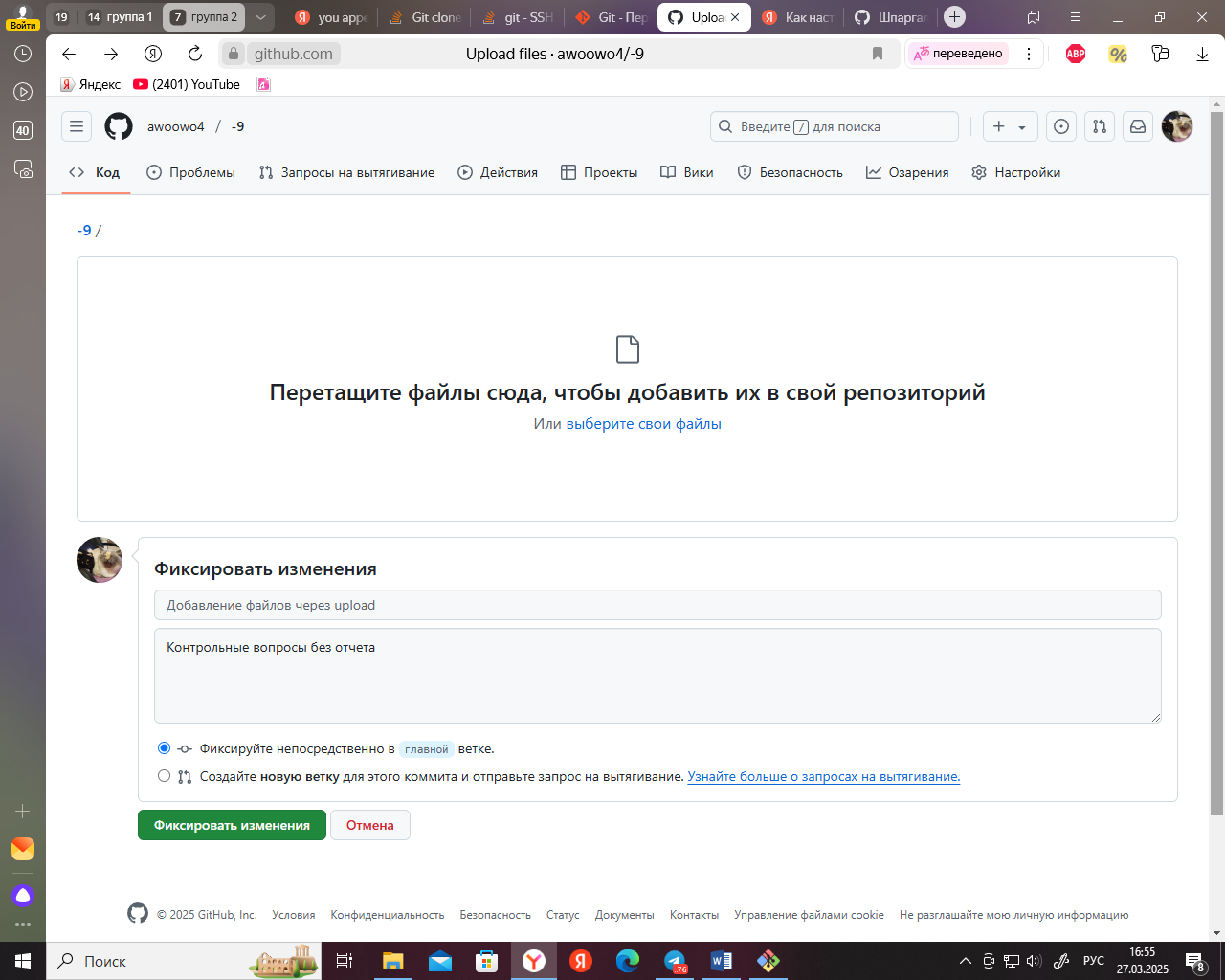


Рисунок 22 – Добавляем файл

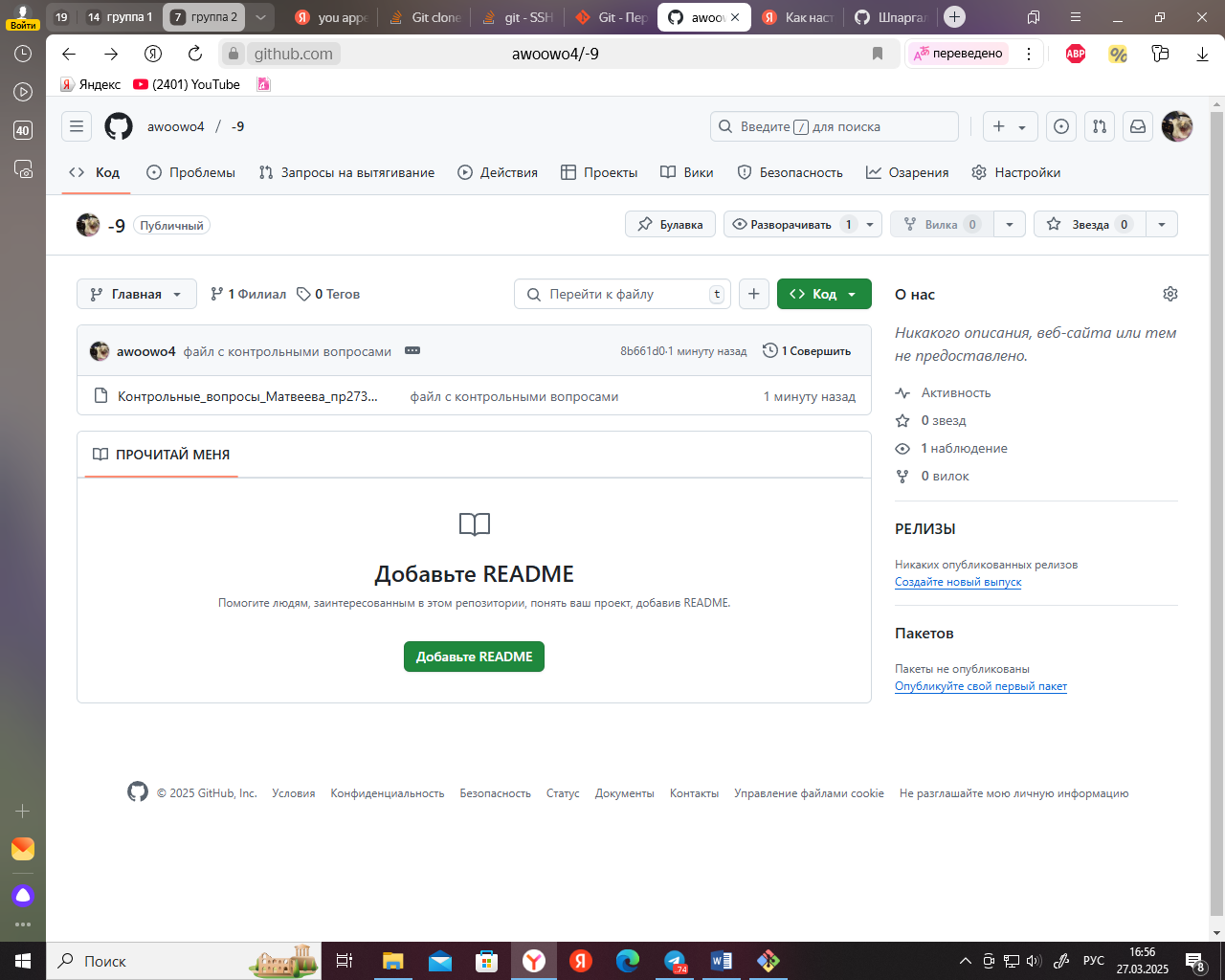


Рисунок 23 – Файл добавлен

Переходим в Git Brush:

Добавляем удаленный репозиторий с помощью команды:

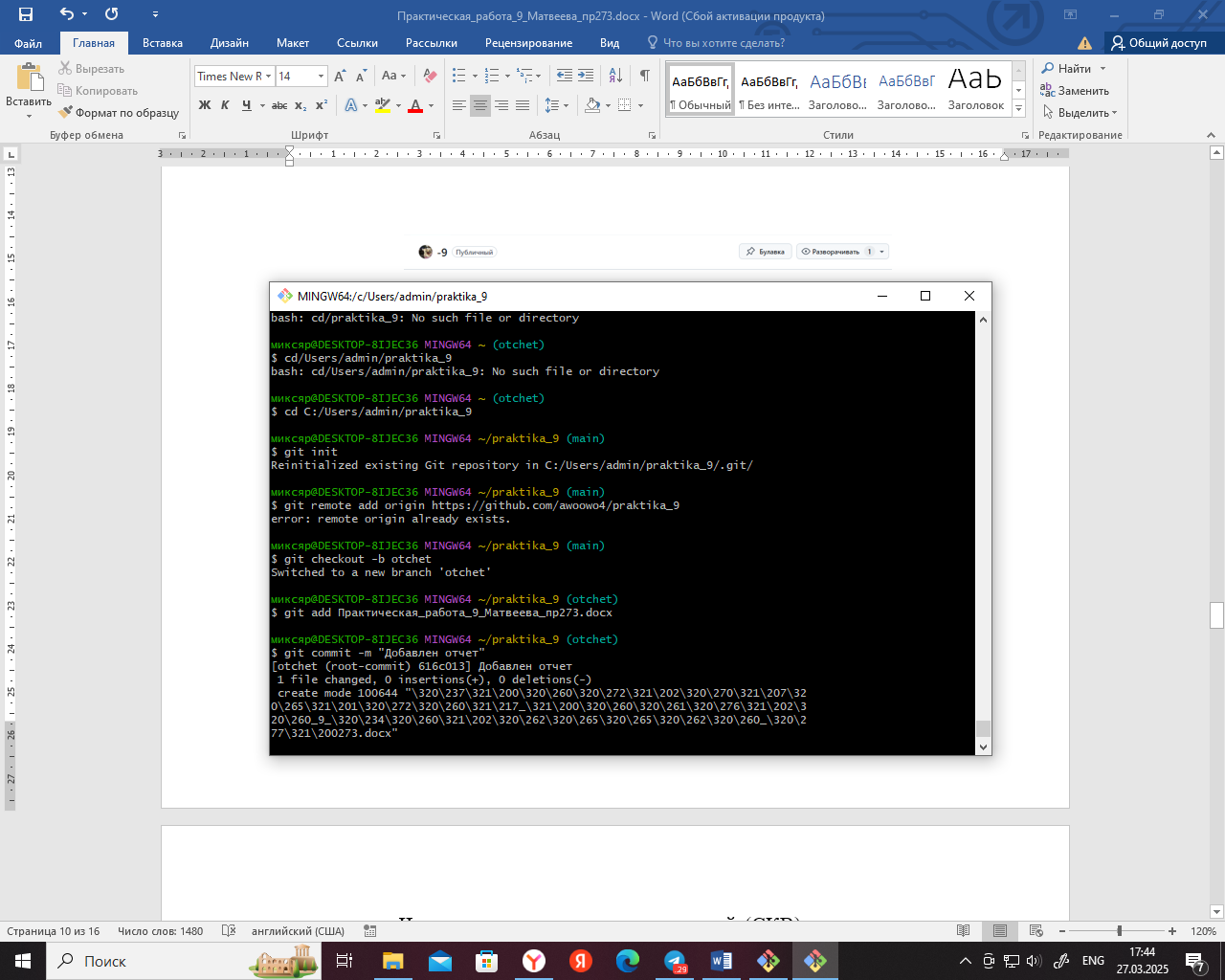


Рисунок 24 – Подключение удаленного репозитория

Создается папка praktika\_9, где хранятся все файлы.

Переходим в директорию нового репозитория и инициализируем новый репозиторий:

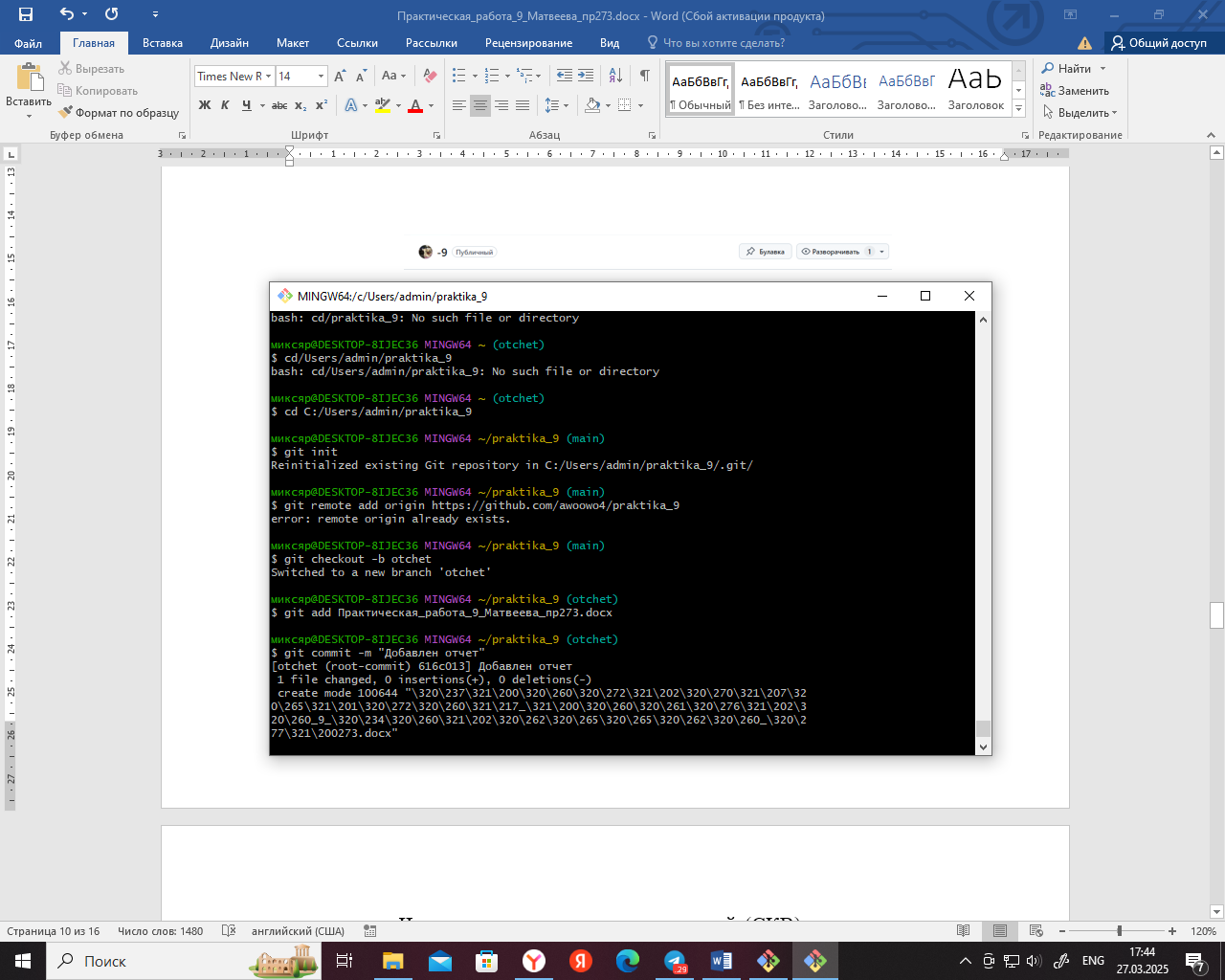


Рисунок 25 – Новая директория и инициализация

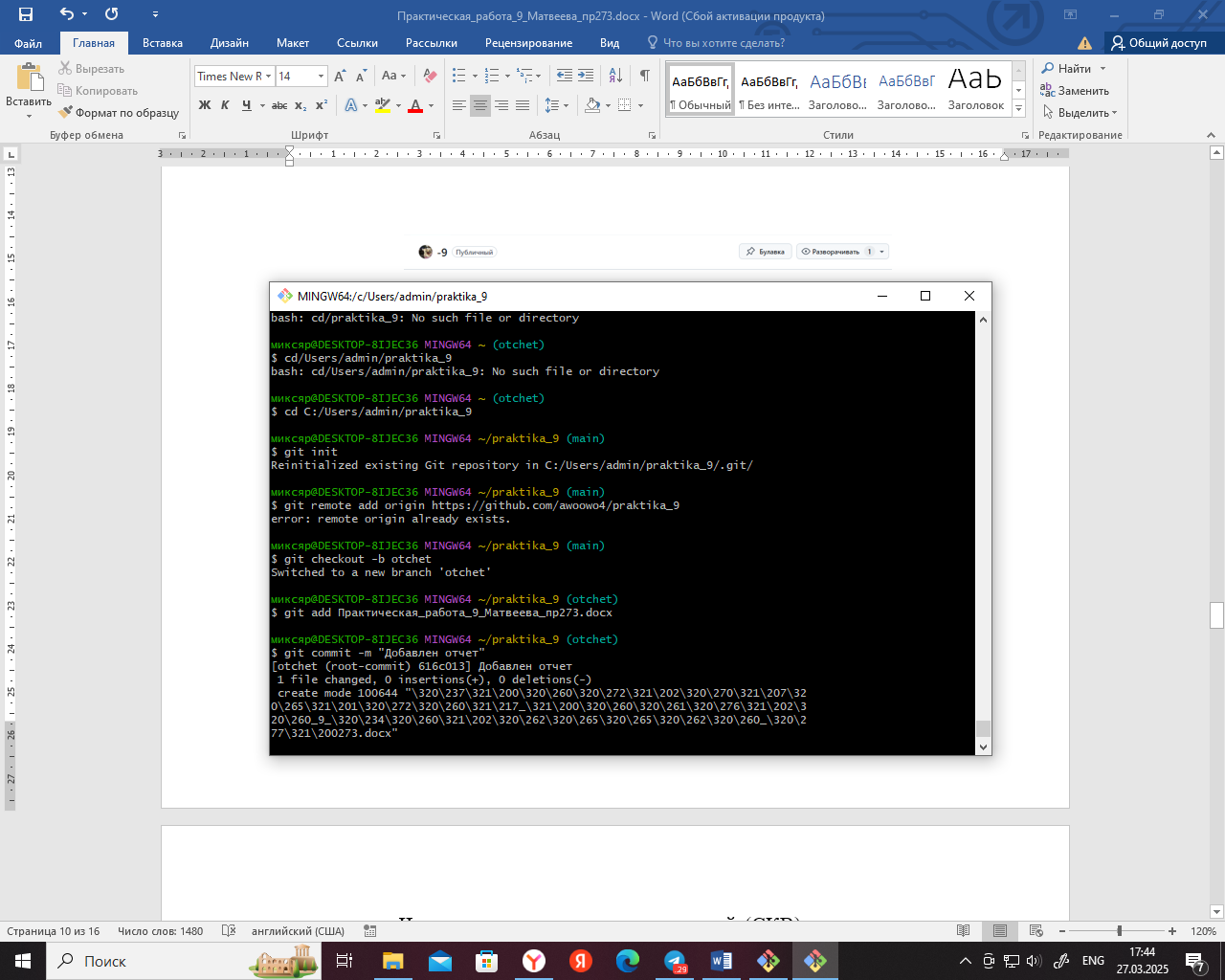


Рисунок 26 – Новая ветка «Otchet»

Кидаем файл с отчетом в папку с репозиторием:

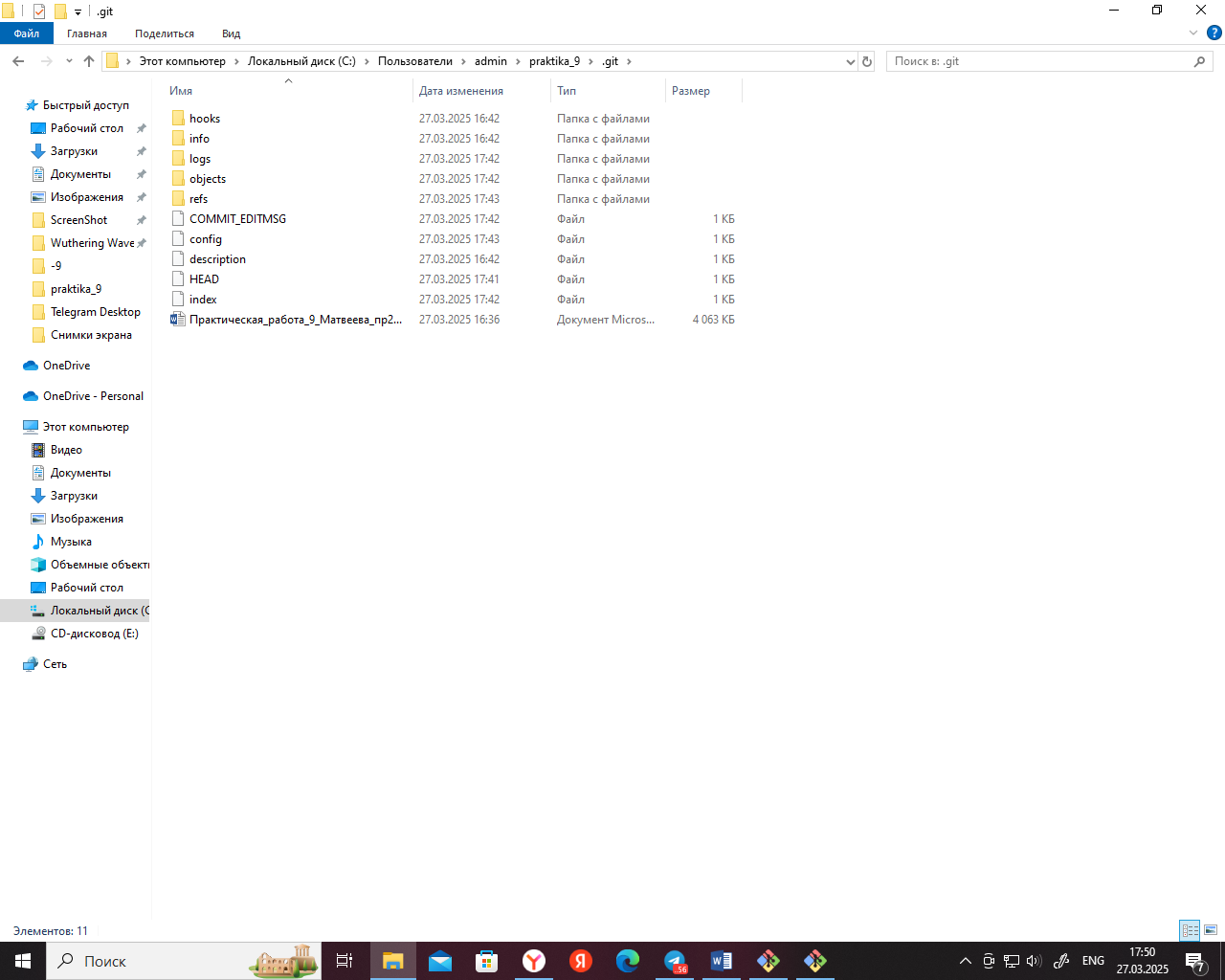


Рисунок 27 – Файл с отчетом в папке репозитория

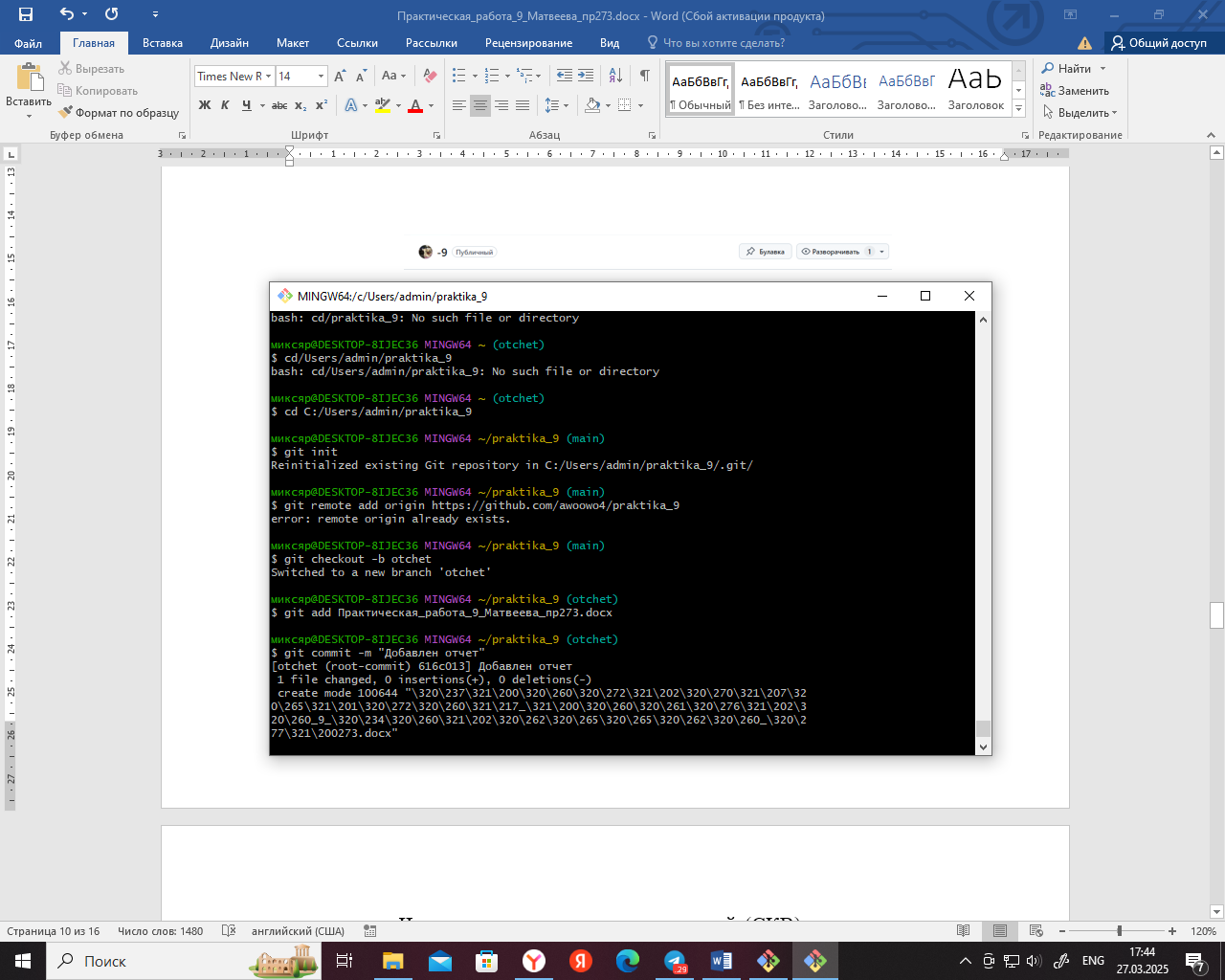


Рисунок 28 – Добавляем файл в ветку

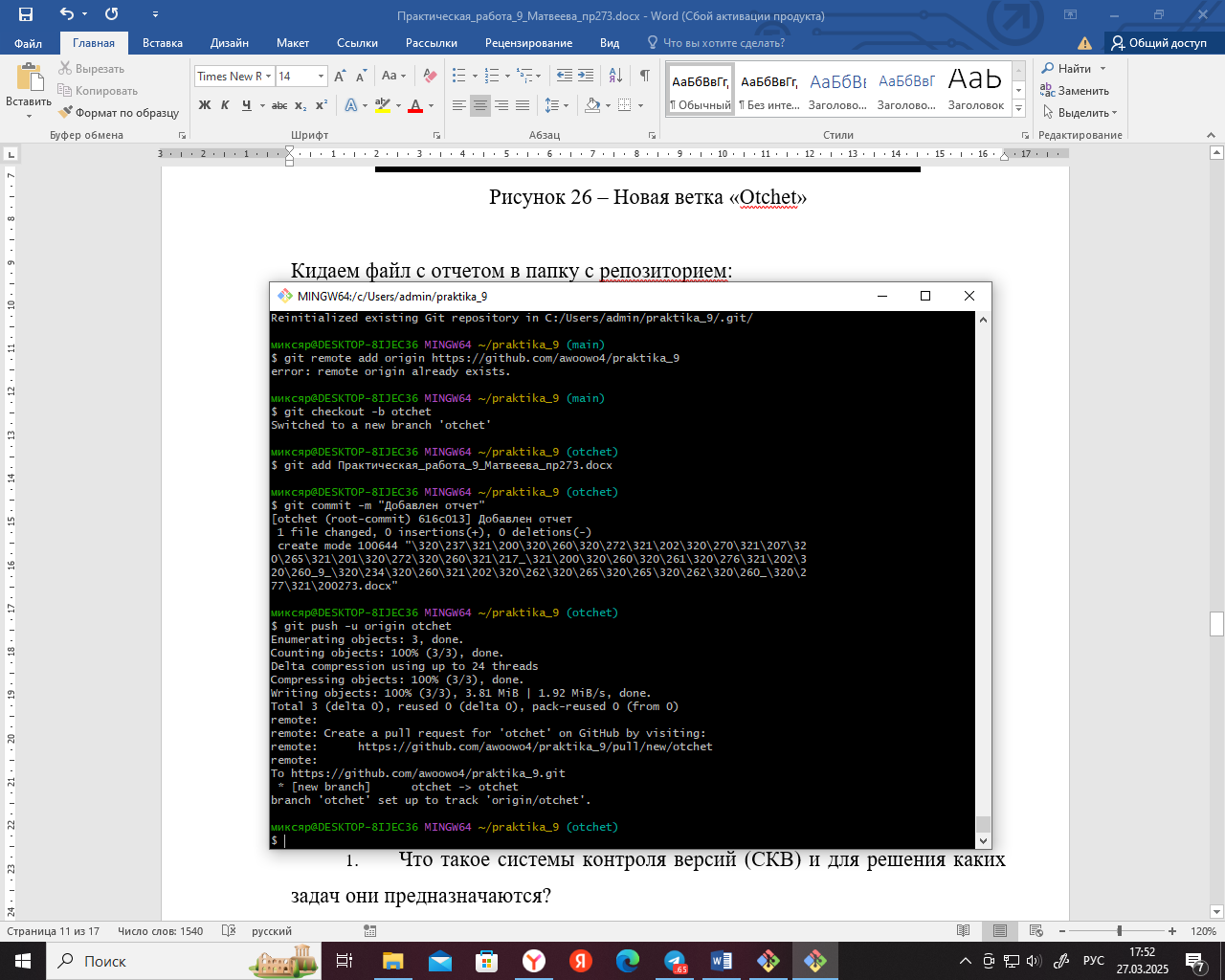


Рисунок 29 – Фиксируем изменения

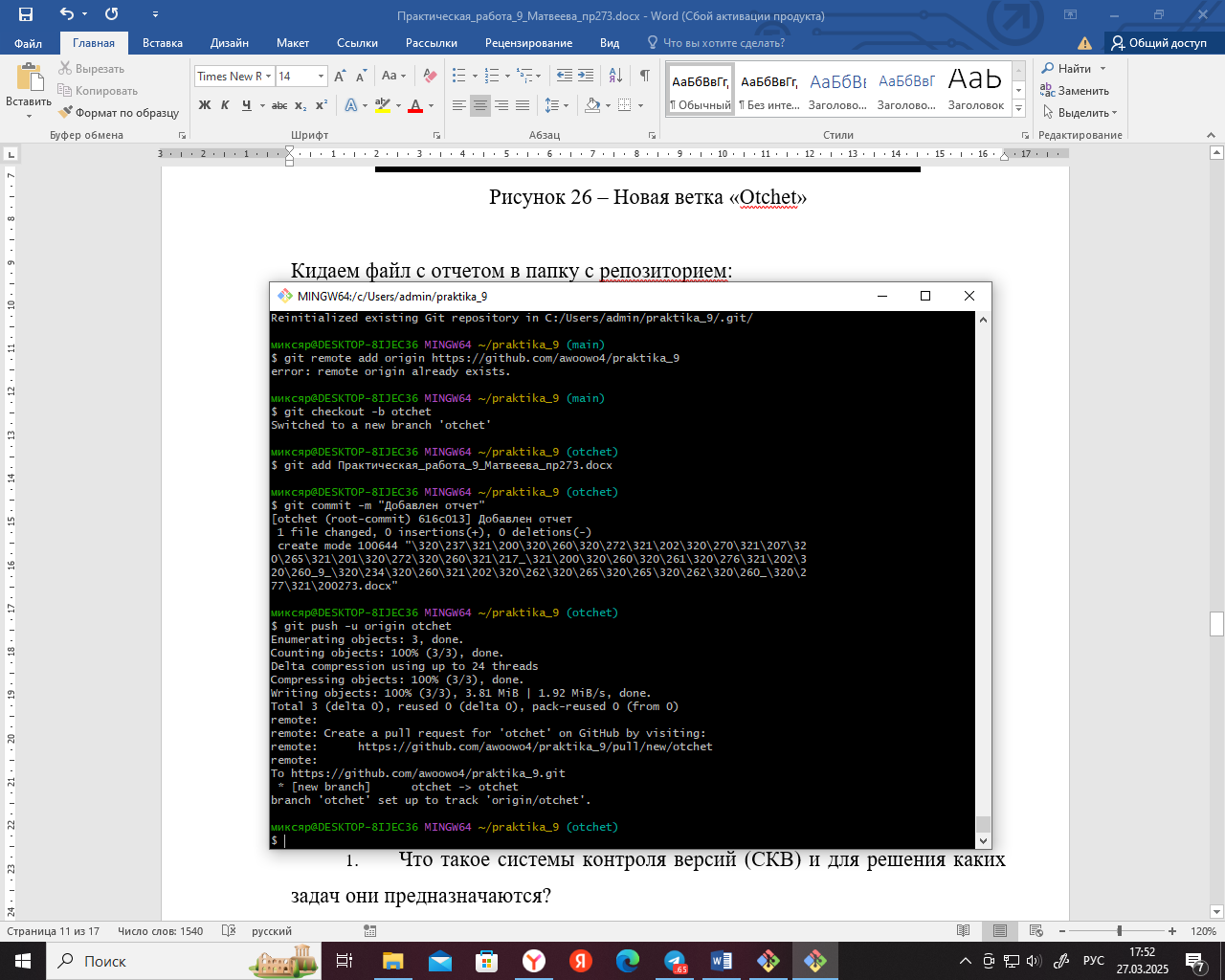


Рисунок 30 – Отправляем изменения

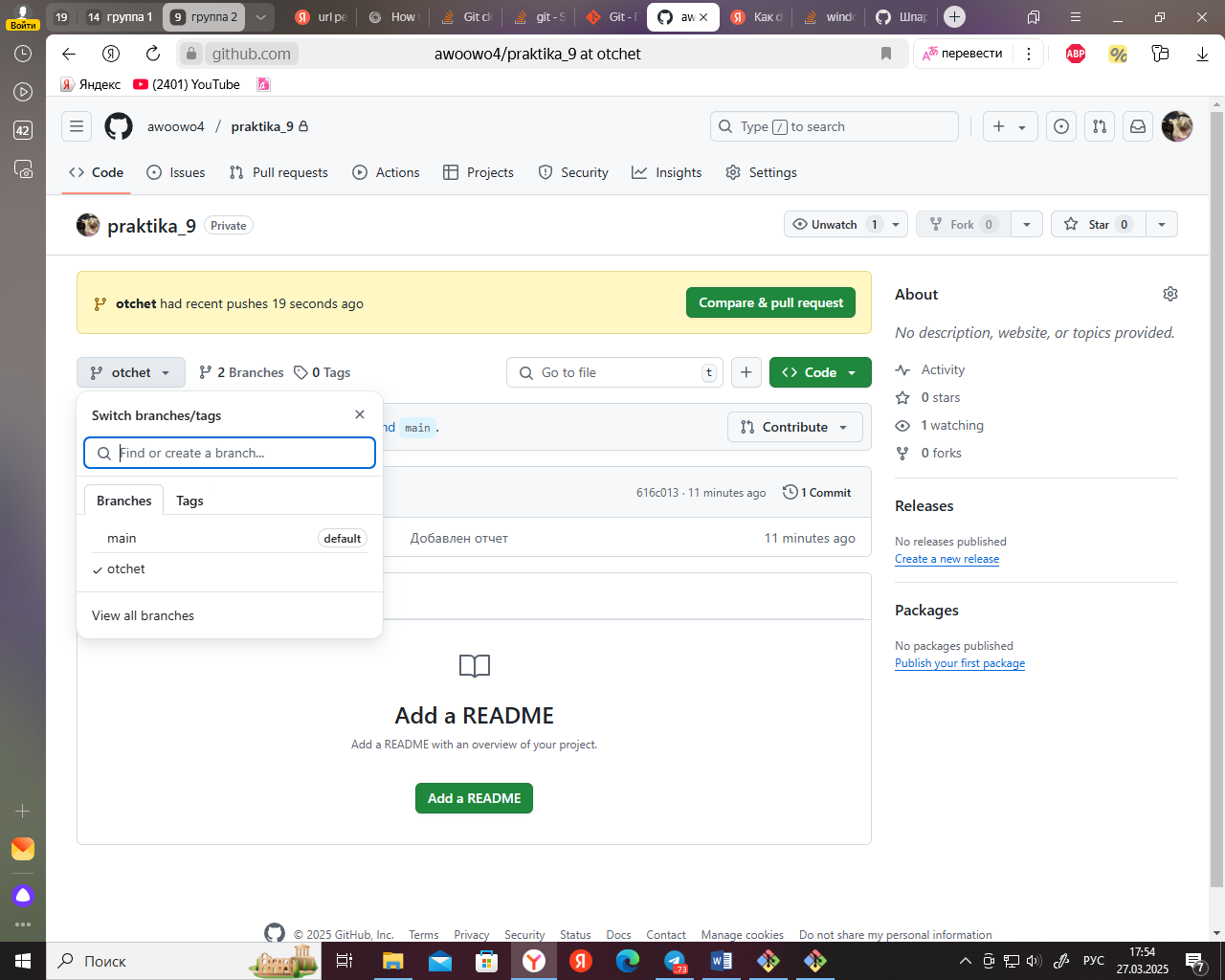


Рисунок 31 – Принимаем

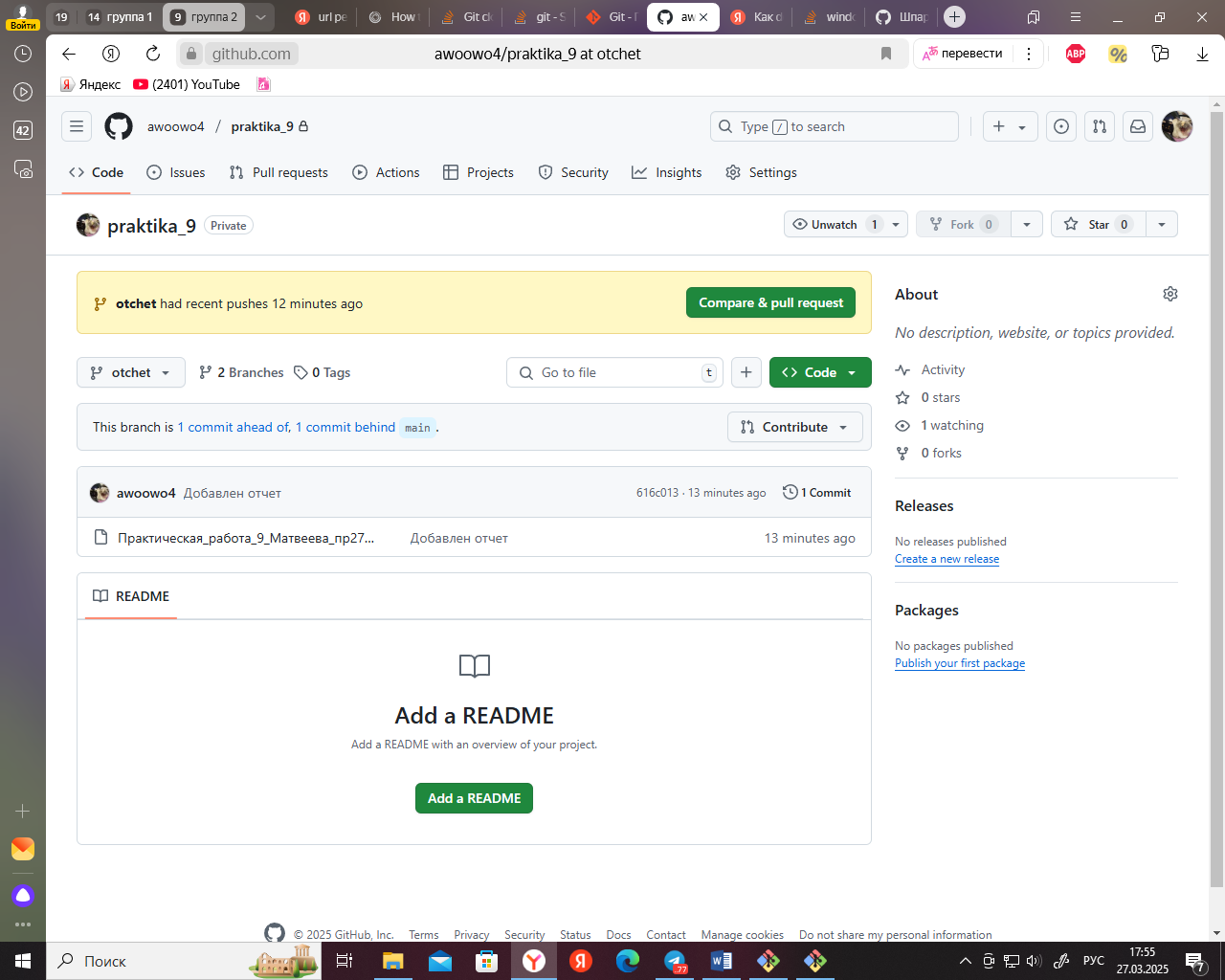


Рисунок 32 – Все есть

Ссылка на репозиторий: <https://github.com/awoowo4/praktika_9>

**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. Клонируешь проект.
3. Работаешь, вносишь изменения.
4. Делаешь commit.
5. Повторяешь, если надо.
6. Опишите порядок работы с общим хранилищем в централизованной СКВ.
7. Клонируешь проект с основного сервера.
8. Работаешь, вносишь изменения.
9. Делаешь commit.
10. Отправка изменений в общее хранилище (push)
11. Повторяешь, если надо.
12. 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?

Version Control System (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. Клонируешь проект с основного сервера.
8. Работаешь, вносишь изменения.
9. Делаешь commit.
10. Отправка изменений в общее хранилище (push)
11. Повторяешь, если надо.
12. Каковы основные задачи, решаемые инструментальным средством git?
13. Управление историей изменений.
14. Поддержание различных ветвей разработки.
15. Удобство совместной работы команды.
16. Возможность восстановления предыдущих версий.
17. Назовите и дайте краткую характеристику командам git.

– git init – инициализация нового репозитория;

– git clone – клонирование существующего репозитория;

– git push – Команда «git push» 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;

– git add – добавление изменений в индекс;

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

– git push – отправка изменений на удаленный сервер;

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

– git branch – создание новых веток;

– git merge – слияние веток.

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

При работе с локальными репозиториями:

– Создаем новый проект: git init;

– Добавляем файлы: git add…;

– Фиксируем изменения: git commit -m "Initial commit".

При работе с удаленными репозиториями:

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

– Отправляем изменения: git push origin main;

– Получаем обновления: git pull origin main.

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.