Филиал учреждения образования

«Белорусский государственный технологический университет»

«Бобруйский государственный лесотехнический колледж»

Специальность 2-40 01 01

Программное обеспечение

информационных технологий

**ОТЧЁТ**

по лабораторным работам

**Учебный предмет, модуль:** КиАПС

**Выполнили обуч-ся гр.** ПО-4 А.О.Алексеева, Е.С. Астапович\_\_\_\_\_\_\_

**Преподаватель**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Г.Г.Булыго\_\_\_

Бобруйск

2023

**Содержание**

[**Лабораторная работа №1** 3](#_Toc159057950)

[**Лабораторная работа №2** 18](#_Toc159057951)

# **Лабораторная работа №1**

**Тема:** Разработка программ с использованием системы управления версиями.

**Цель:** Изучить идеологию и применение средств контроля версий. Выработать умения использовать средства контроля версий при разработке программ.

**Технологическое оснащение:** Методические указания к выполнению, ОС Windows, текстовый процессор MS Word 2010, СКВ Git.

**Результаты выполнения работы**

**Задание 1.** Установить Git.

1.Выбираем имя основной ветки.

На рисунке 1 представлен скриншот установки.

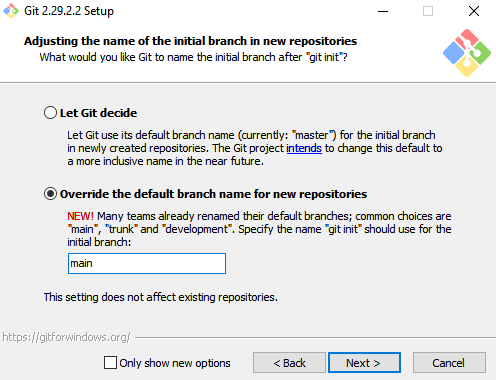


Рисунок 1 – Скриншот установки

2.Настраиваем запуск Git из консоли.

На рисунке 2 представлен скриншот установки.



Рисунок 2 – Скриншот установки

**Задание 2.** Изучить исходные данные, для выполнения работы.

На рисунке 3 представлены исходные данные для выполнения работы.

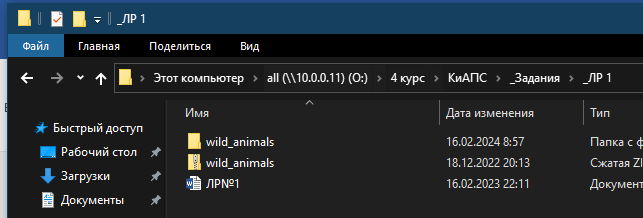


Рисунок 3 – Исходные данные для выполнения работы.

На рисунке 4 представлено содержимое папки wild\_animals.

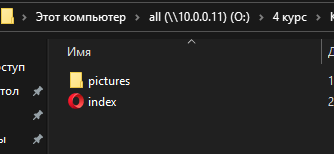


Рисунок 4 – Содержимое папки wild\_animals

**Задание 3.**

1.Создать репозиторий внутри папки wild\_animals. Убедиться, что внутри папки wild\_animals появилась папка. git.

На рисунке 5 представлен созданный репозиторий.

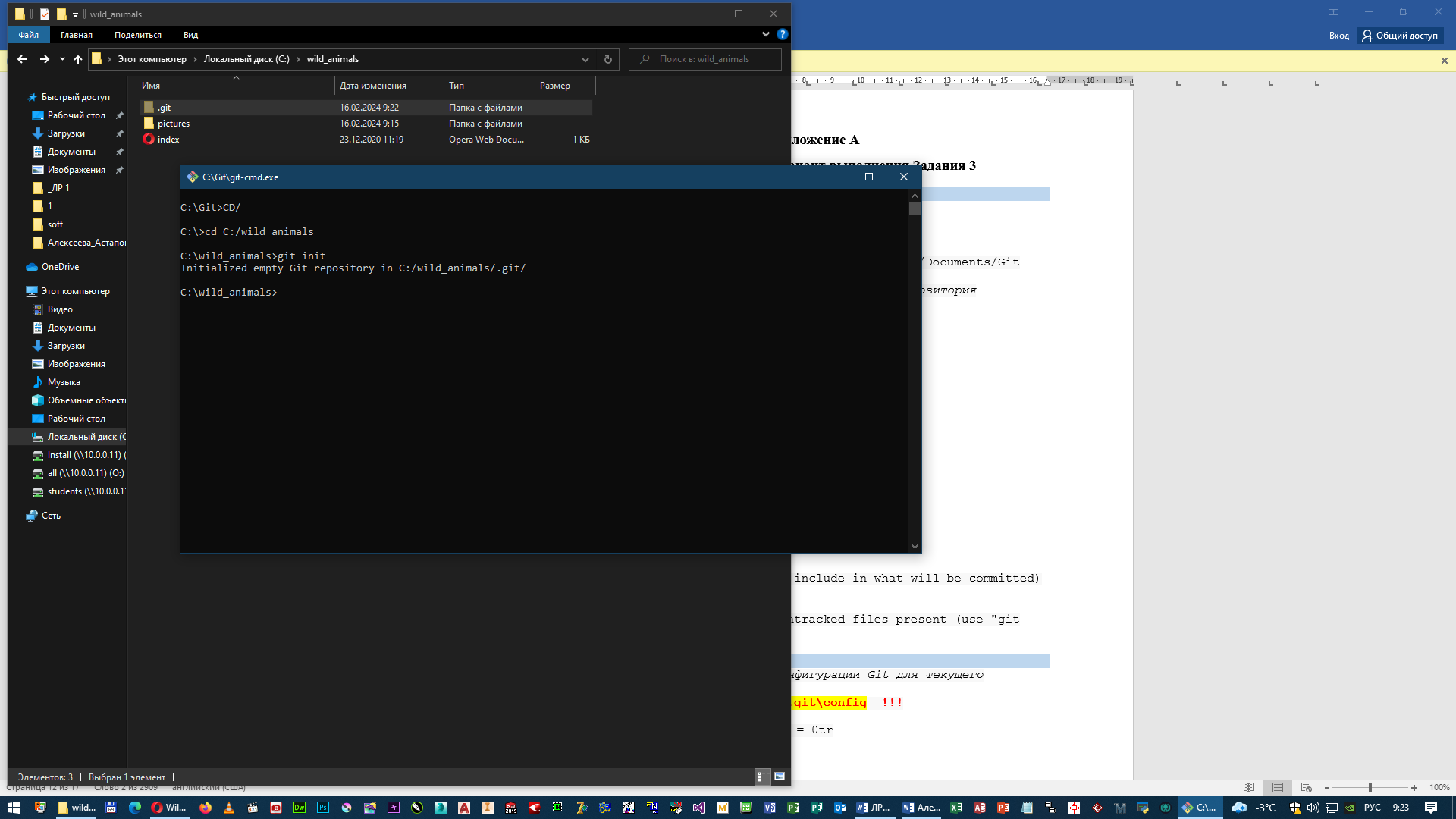


Рисунок 5 – Созданный репозиторий

2.Настроить пользователя Git на уровне репозитория wild\_animals.

На рисунке 6-7 представлено содержимое файла конфигурации Git.

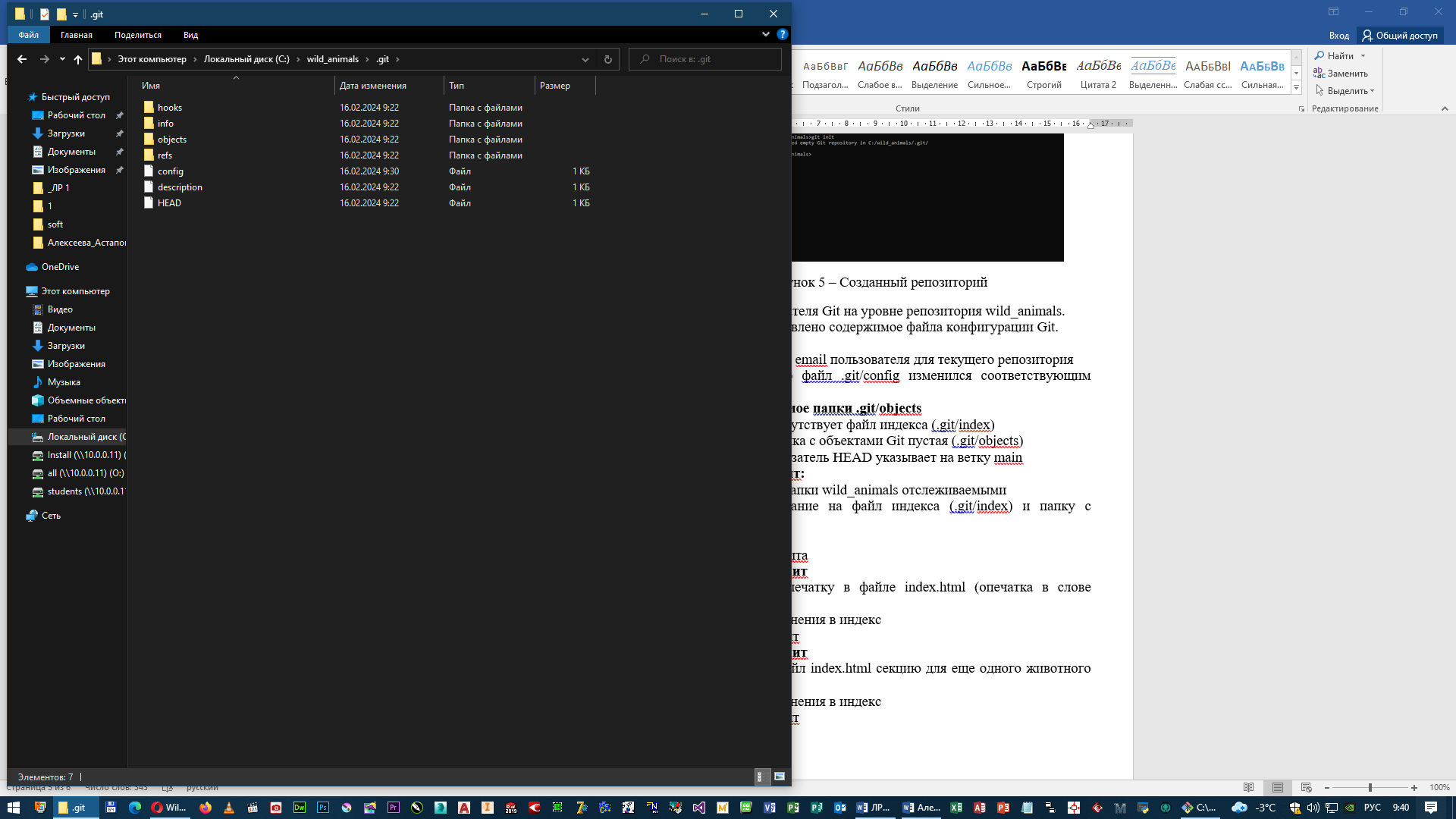


Рисунок 6 – Содержимое файла конфигурации Git

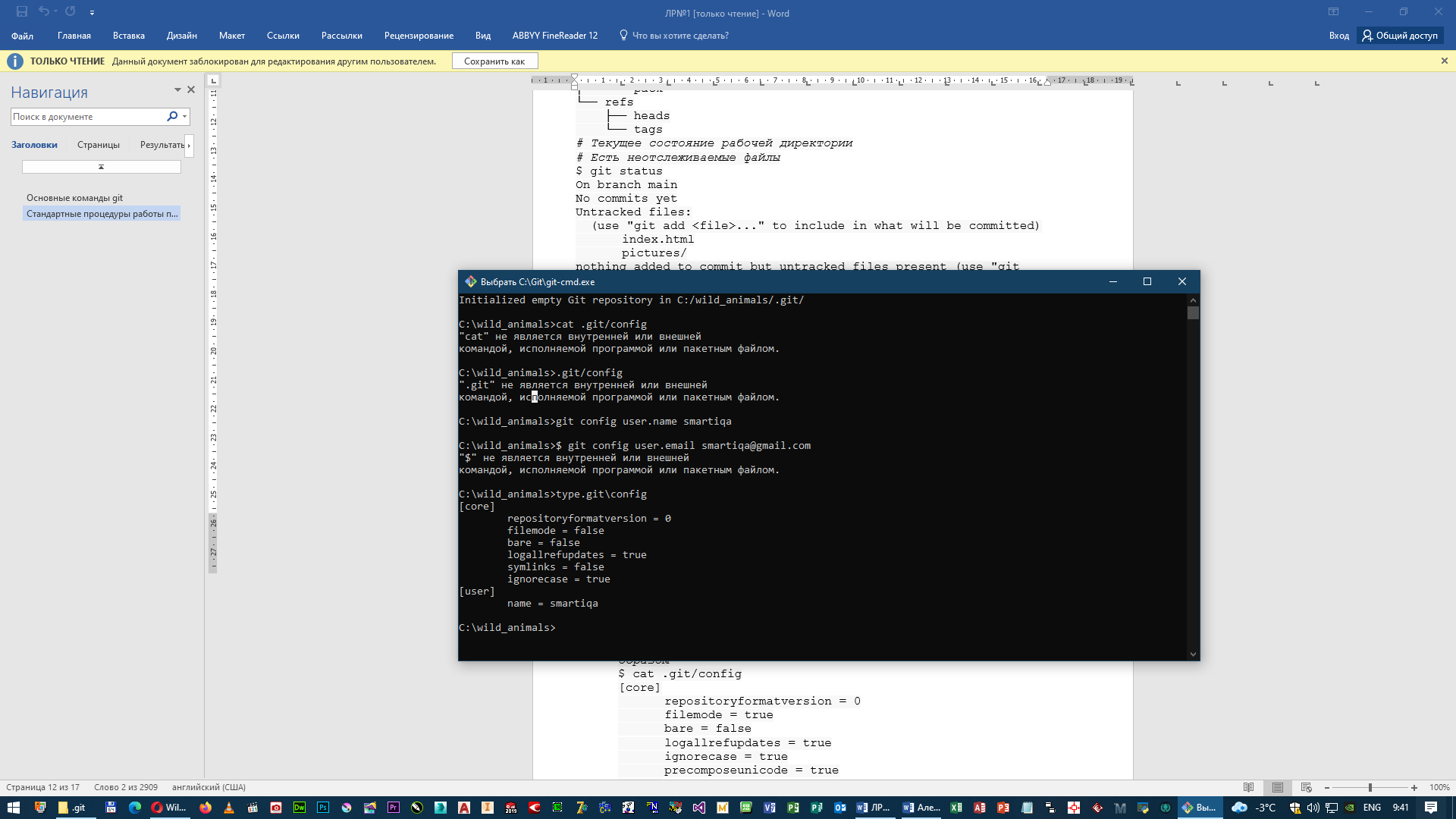


Рисунок 7 – Содержимое файла конфигурации Git (через консоль)

На рисунке 8 представлена настройка имени и email пользователя для текущего репозитория.

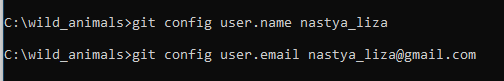


Рисунок 8 – Настройка имени и email пользователя для текущего репозитория

На рисунке 9 представлен результат выполненного задания.

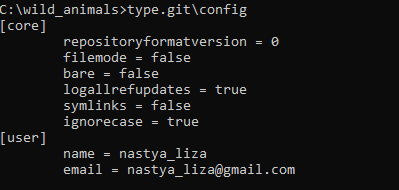


Рисунок 9 – Результат выполненного задания

**3. Изучить содержимое папки .git/objects**

1) Убедиться, что отсутствует файл индекса (.git/index)

2) Убедиться, что папка с объектами Git пустая (.git/objects)

3) Убедиться, что указатель HEAD указывает на ветку main

На рисунке 10 представлен результат выполненного задания.

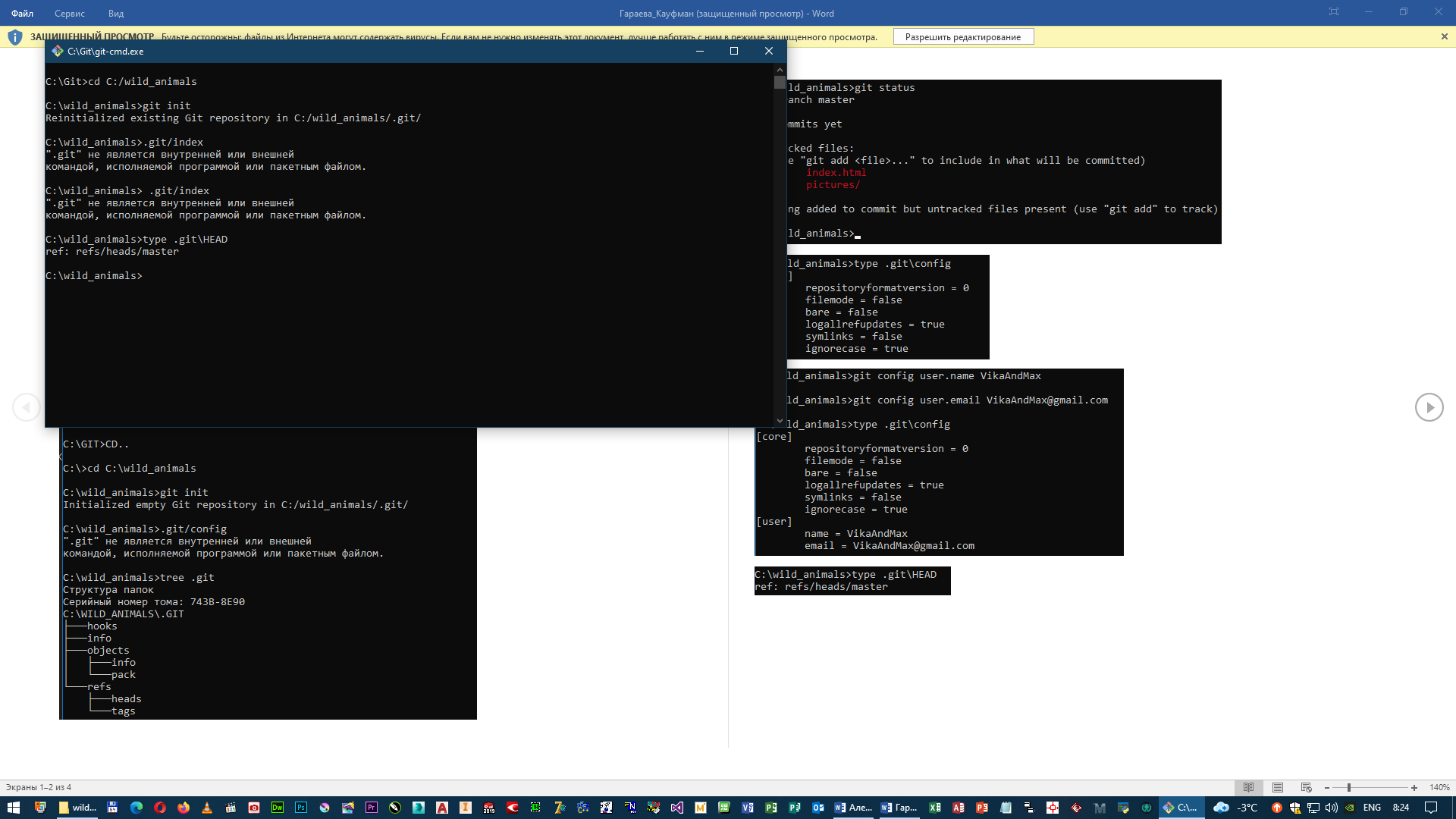


Рисунок 10 – Результат выполненного задания

На рисунке 11 представлена структура папки git.

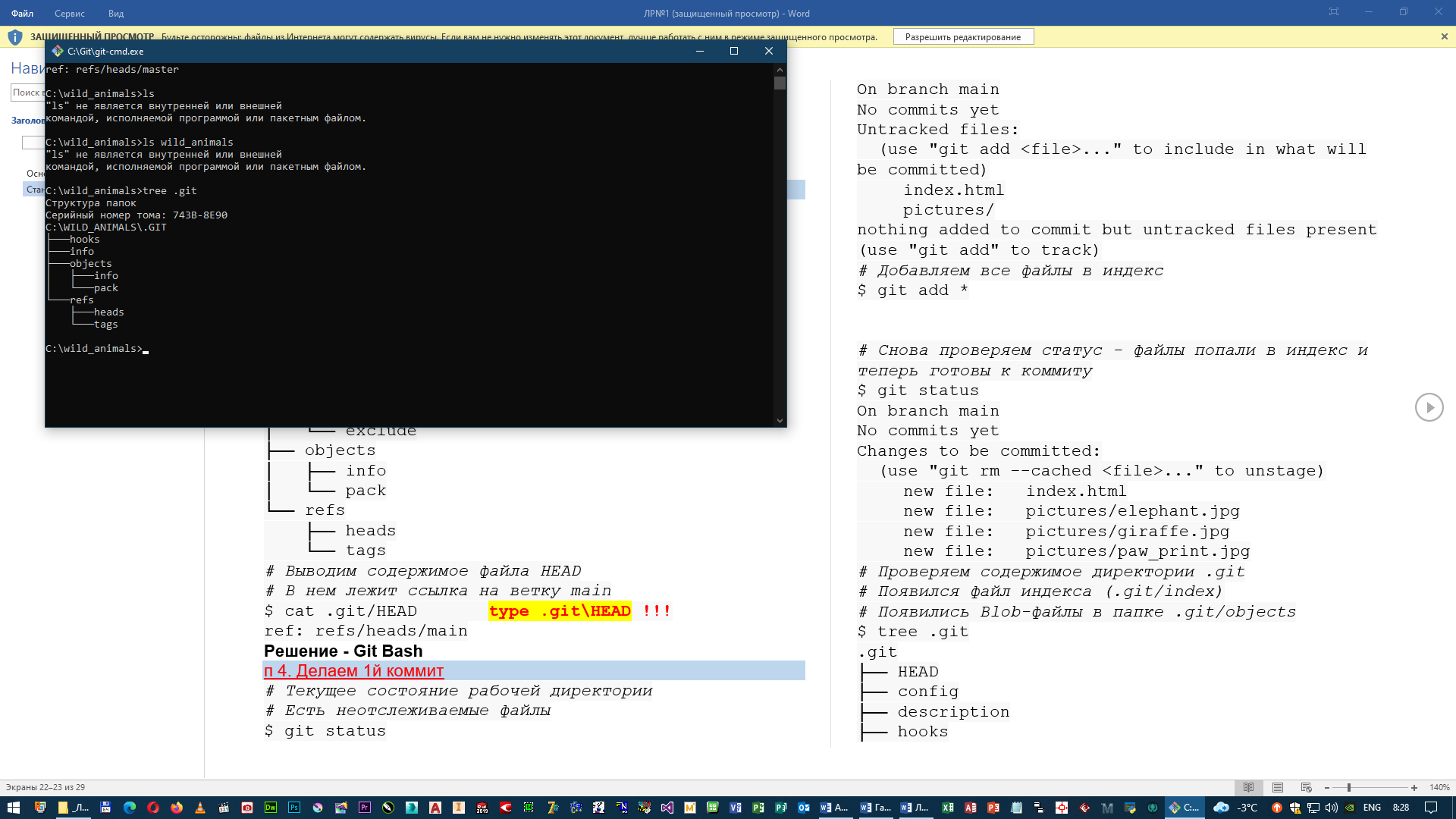


Рисунок 11 – Структура папки git

**4. Сделать 1й коммит:**

1) Сделать файлы папки wild\_animals отслеживаемыми

2) Обратить внимание на файл индекса (.git/index) и папку с объектами (.git/objects)

3) Сделать коммит

4) Найти хэш коммита

На рисунке 12 представлен скриншот выполненного задания.

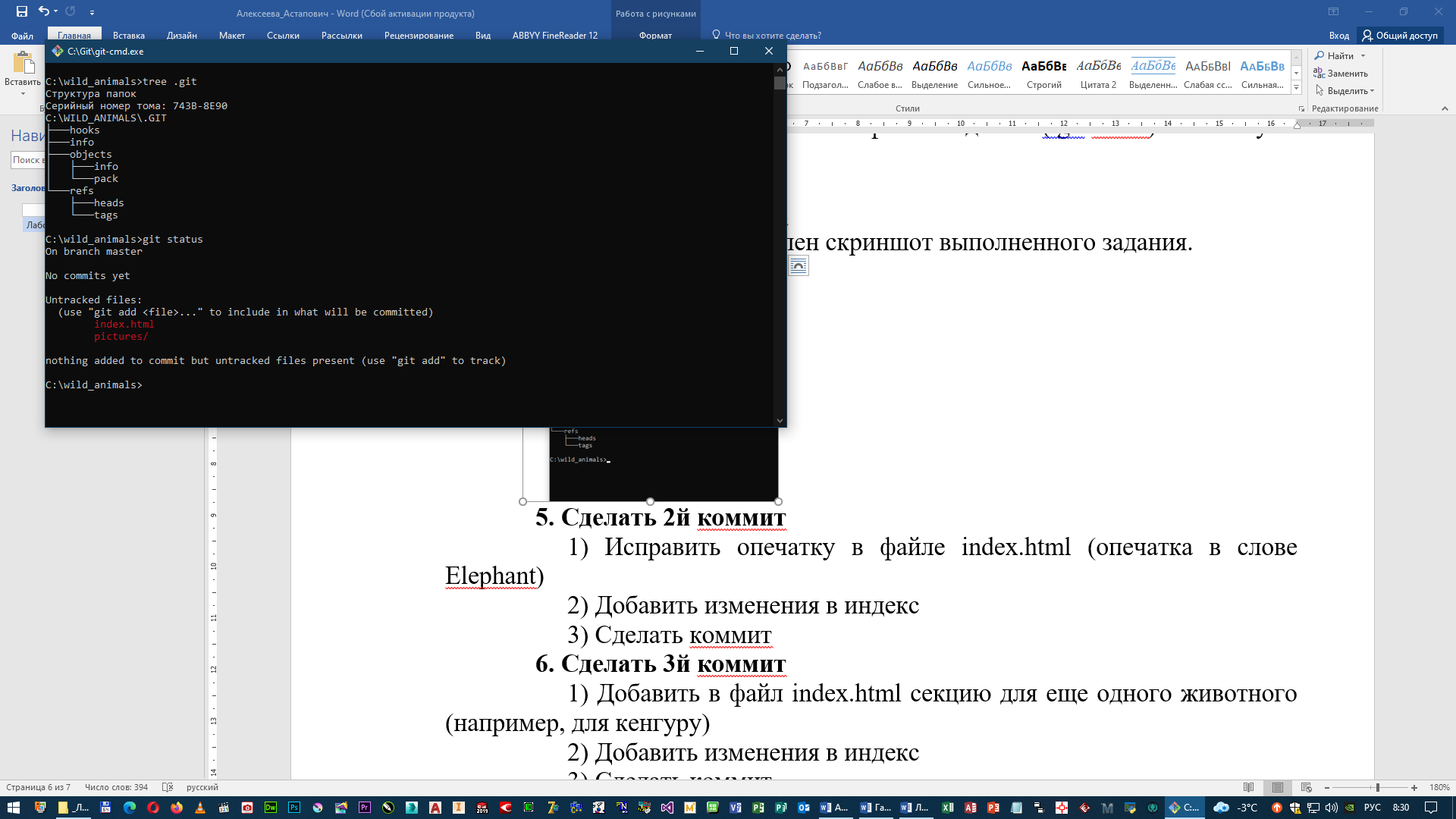


Рисунок 12 – Скриншот выполненного задания

На рисунке 13 представлен скриншот выполненного задания.

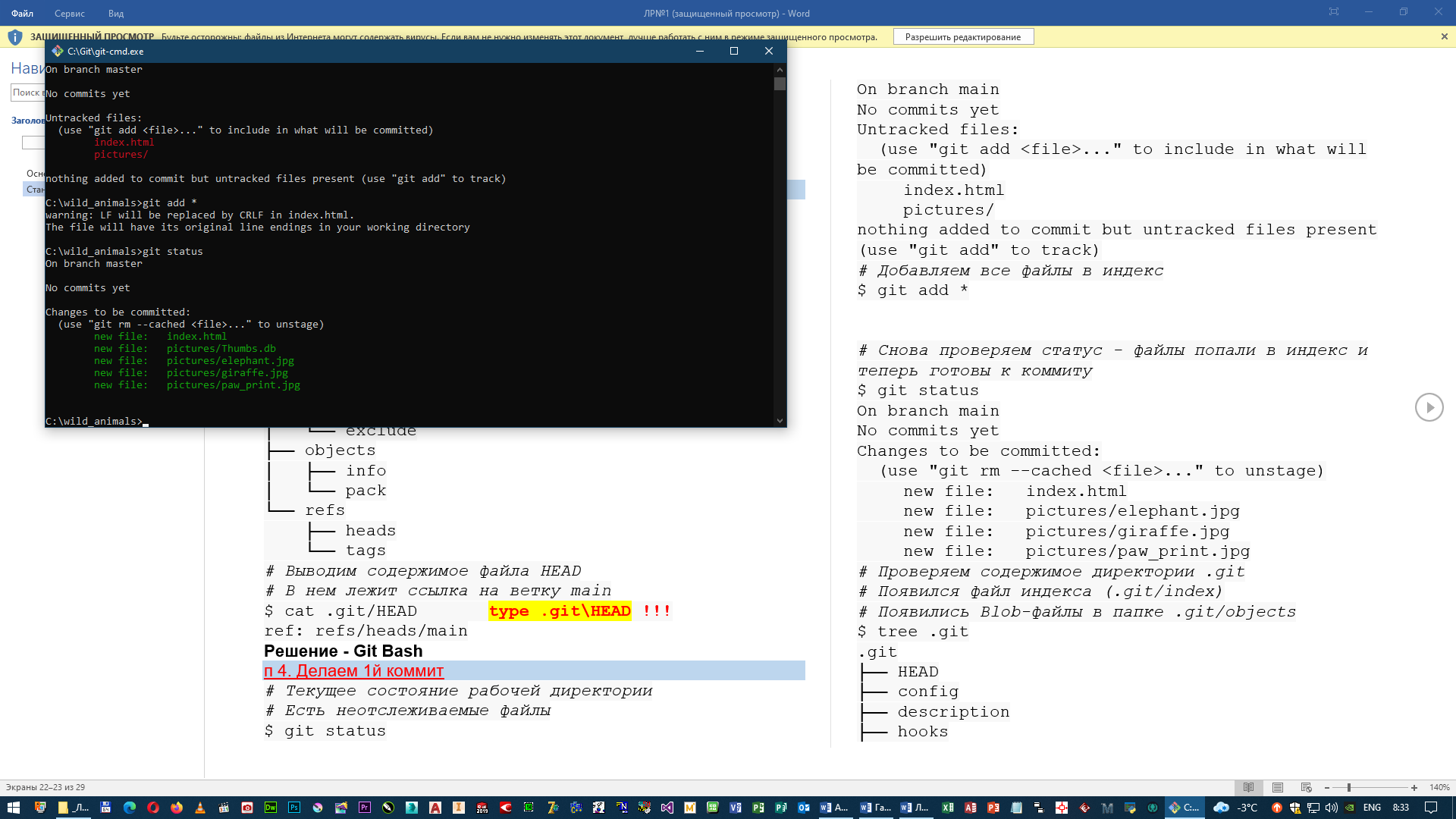


Рисунок 13 – Скриншот выполненного задания

На рисунке 13 представлен скриншот выполненного задания.

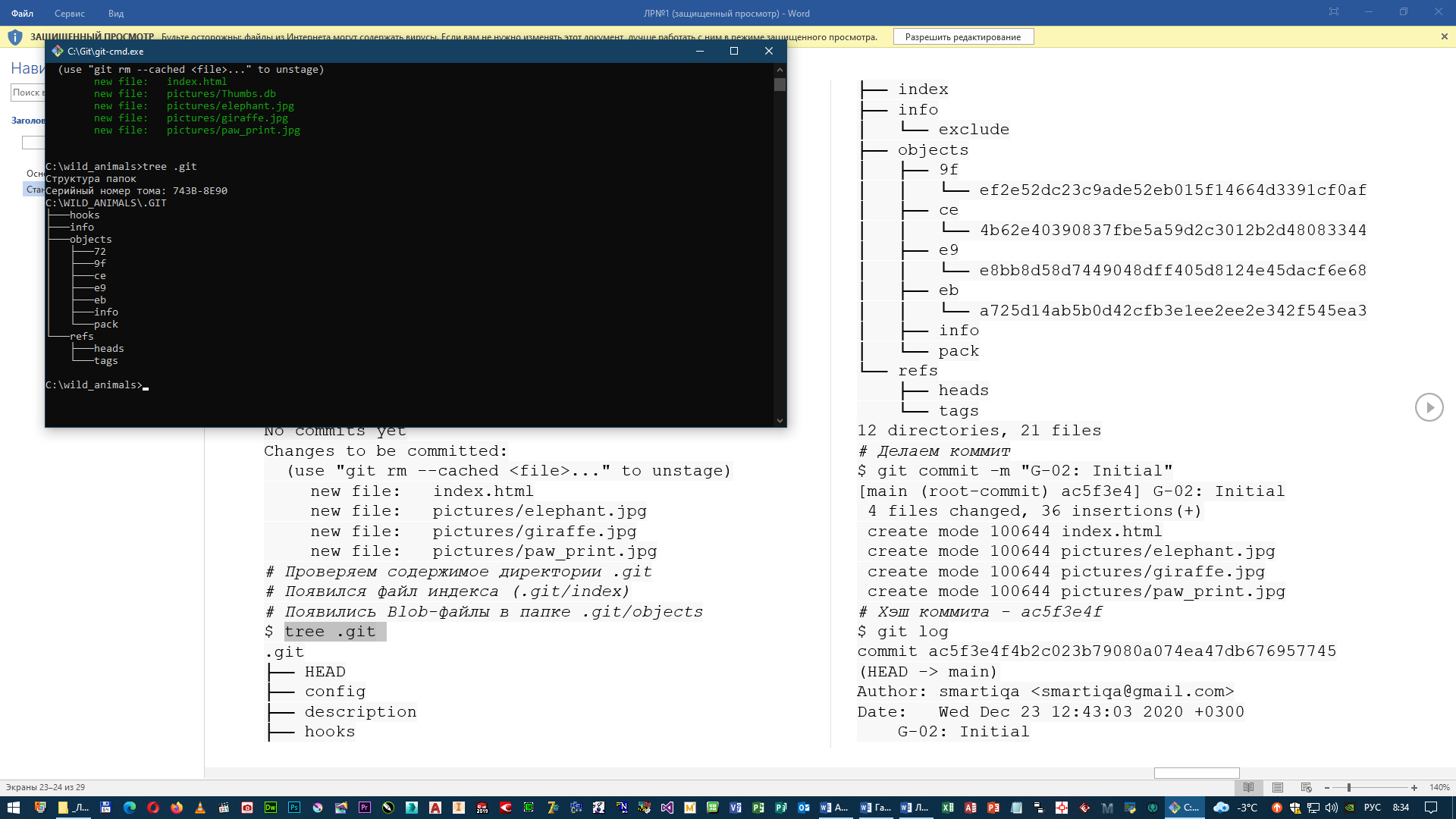


Рисунок 13 – Скриншот выполненного задания

На рисунке 14 представлен скриншот выполненного задания.

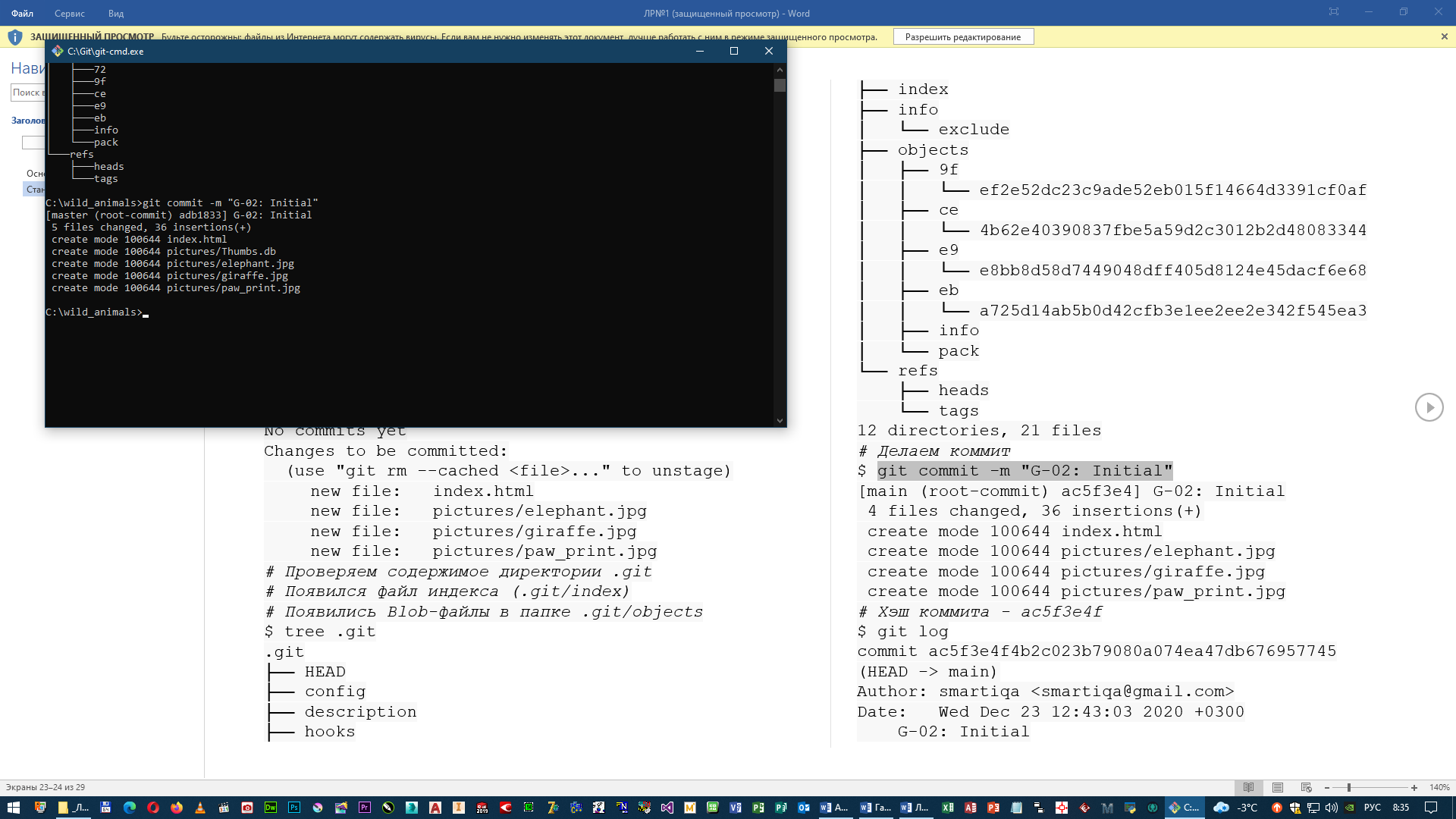


Рисунок 14 – Скриншот выполненного задания

На рисунке 15 представлен скриншот выполненного задания.

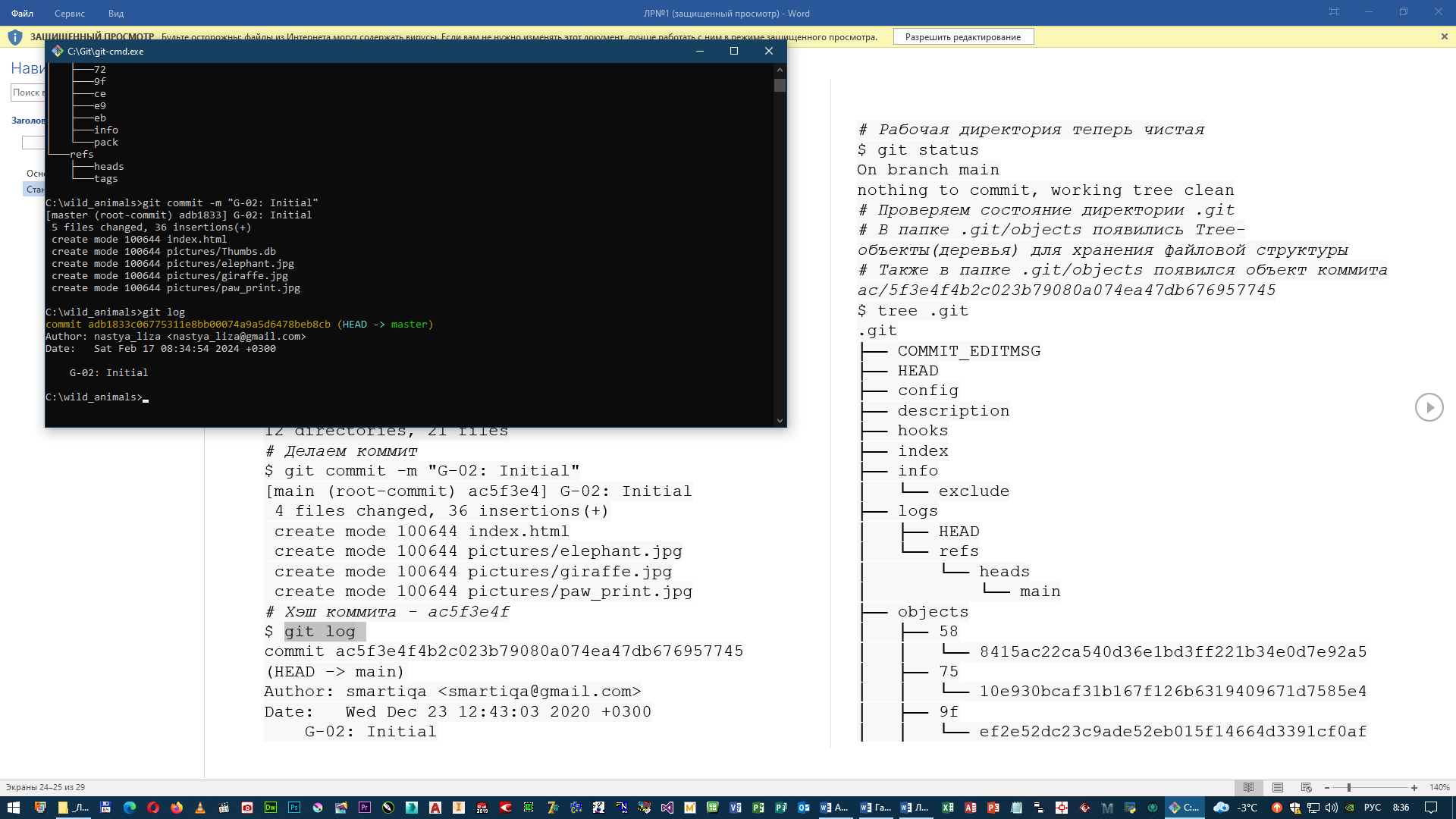


Рисунок 15 – Скриншот выполненного задания

**5. Сделать 2й коммит**

1) Исправить опечатку в файле index.html (опечатка в слове Elephant)

2) Добавить изменения в индекс

3) Сделать коммит

На рисунке 16-18 представлен скриншот выполненного задания.



Рисунок 16 – Скриншот выполненного задания

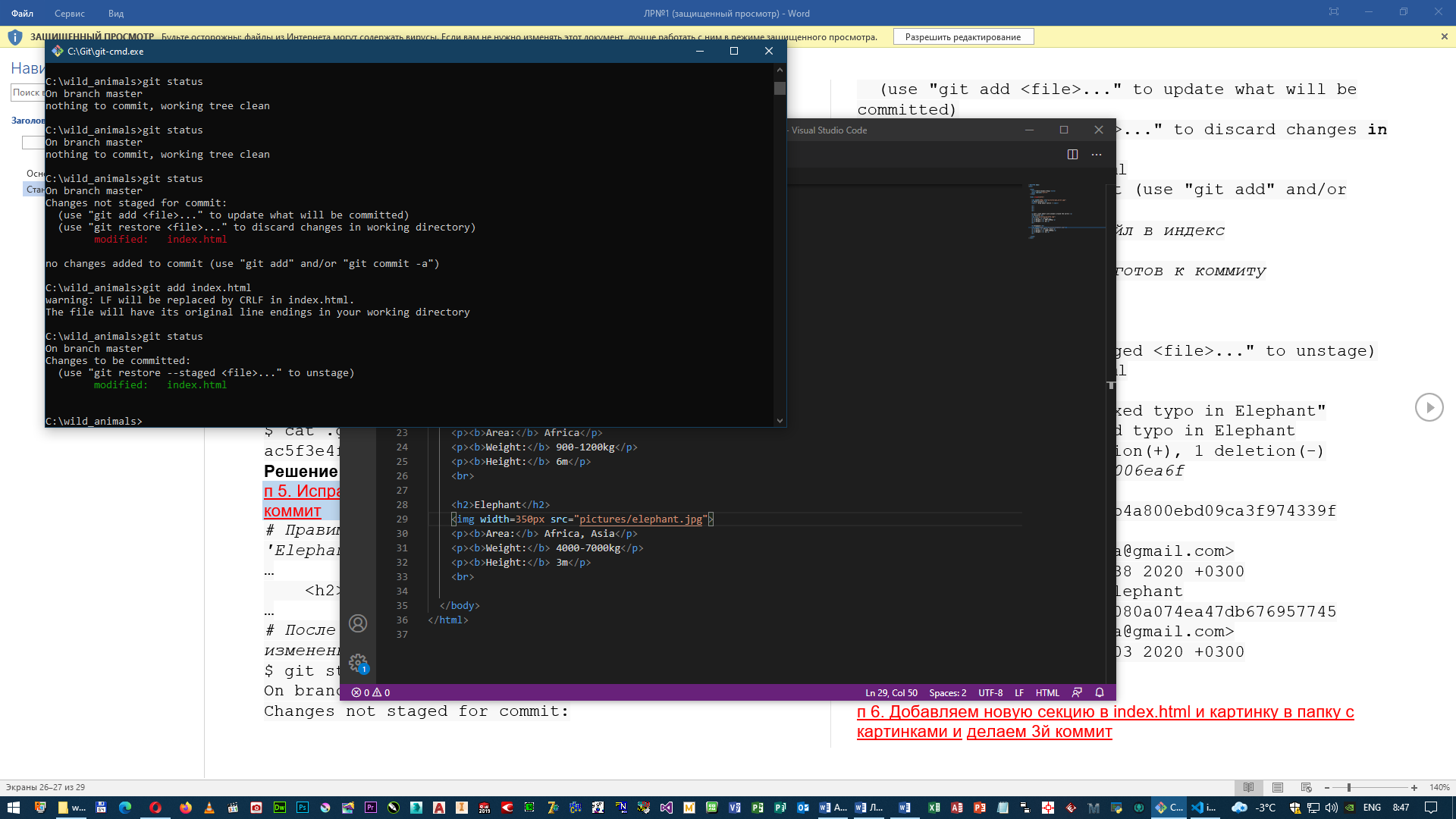


Рисунок 17 – Скриншот выполненного задания

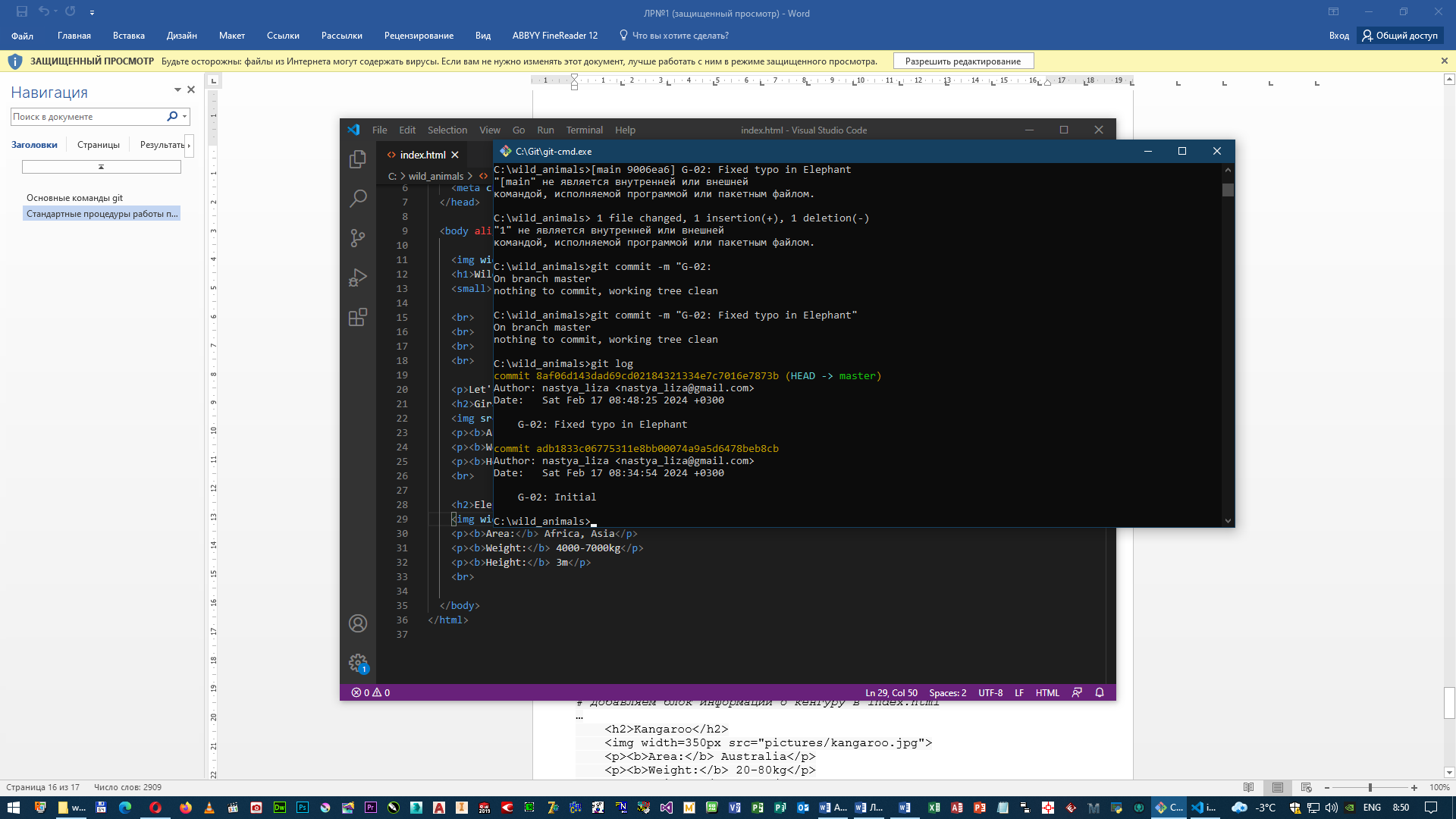


Рисунок 18 – Скриншот выполненного задания

**6. Сделать 3й коммит**

1) Добавить в файл index.html секцию для еще одного животного (например, для кенгуру)

2) Добавить изменения в индекс

3) Сделать коммит

На рисунке 19-23 представлен скриншот выполненного задания.

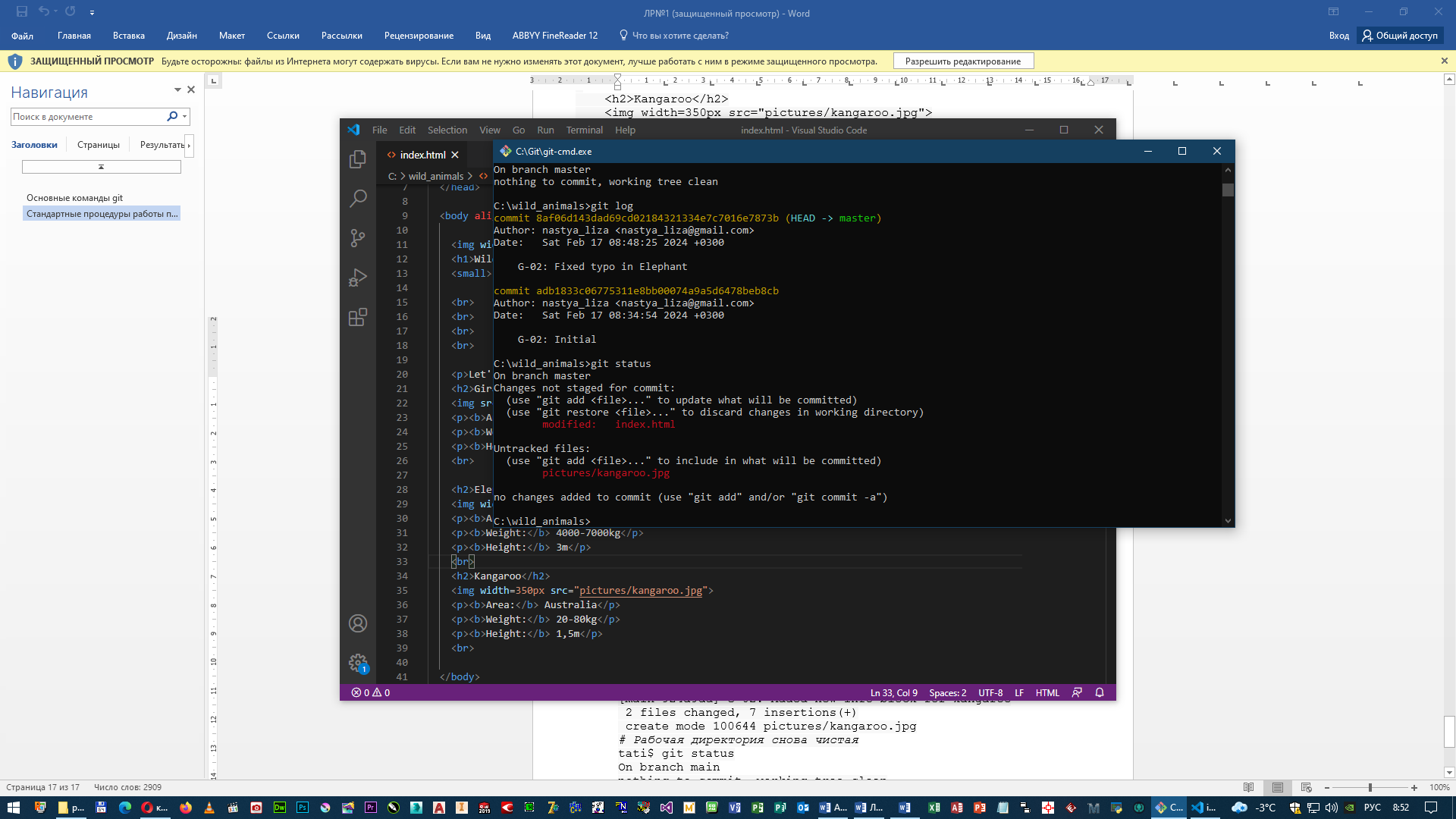


Рисунок 19 – Скриншот выполненного задания

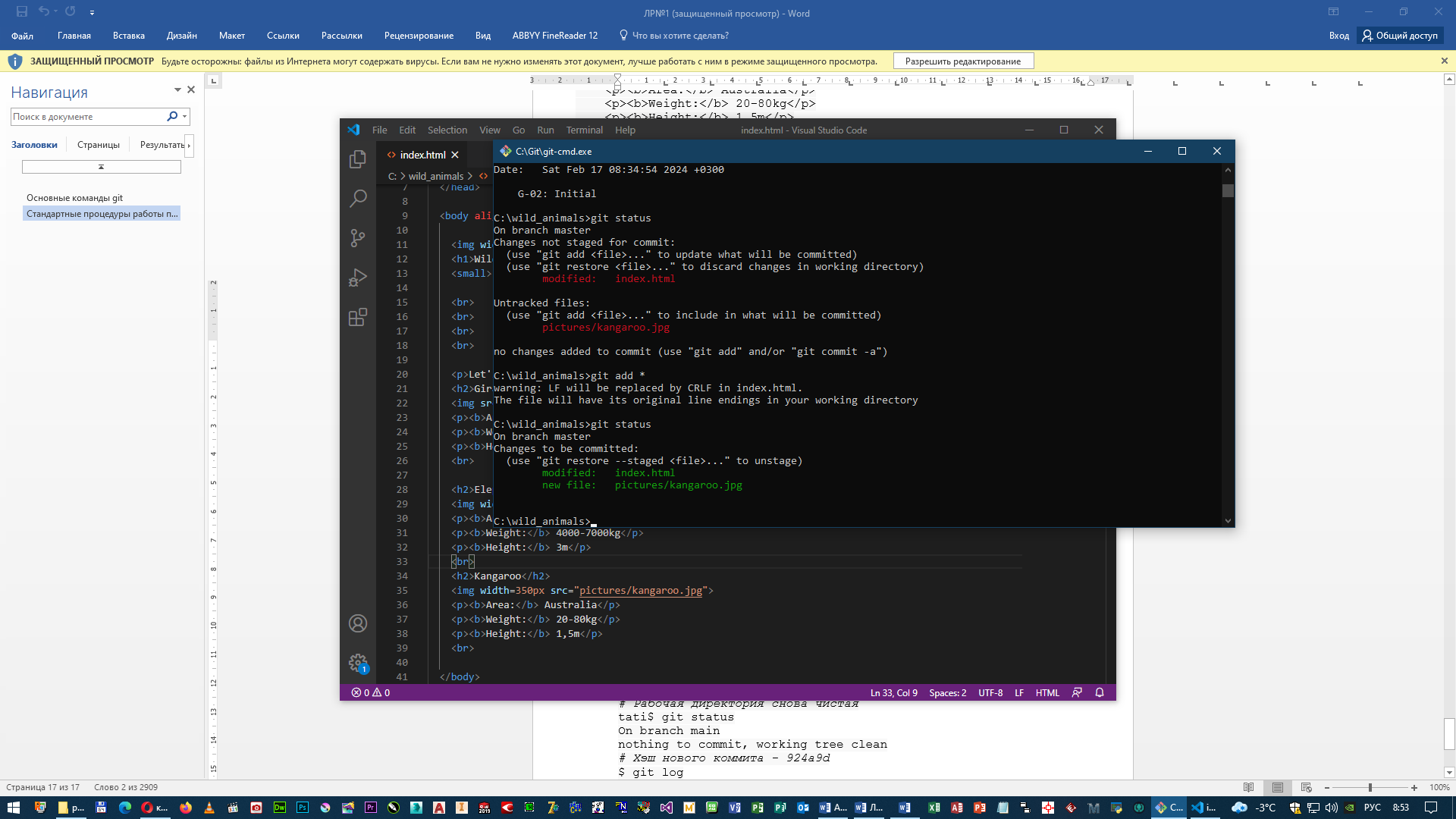


Рисунок 20 – Скриншот выполненного задания

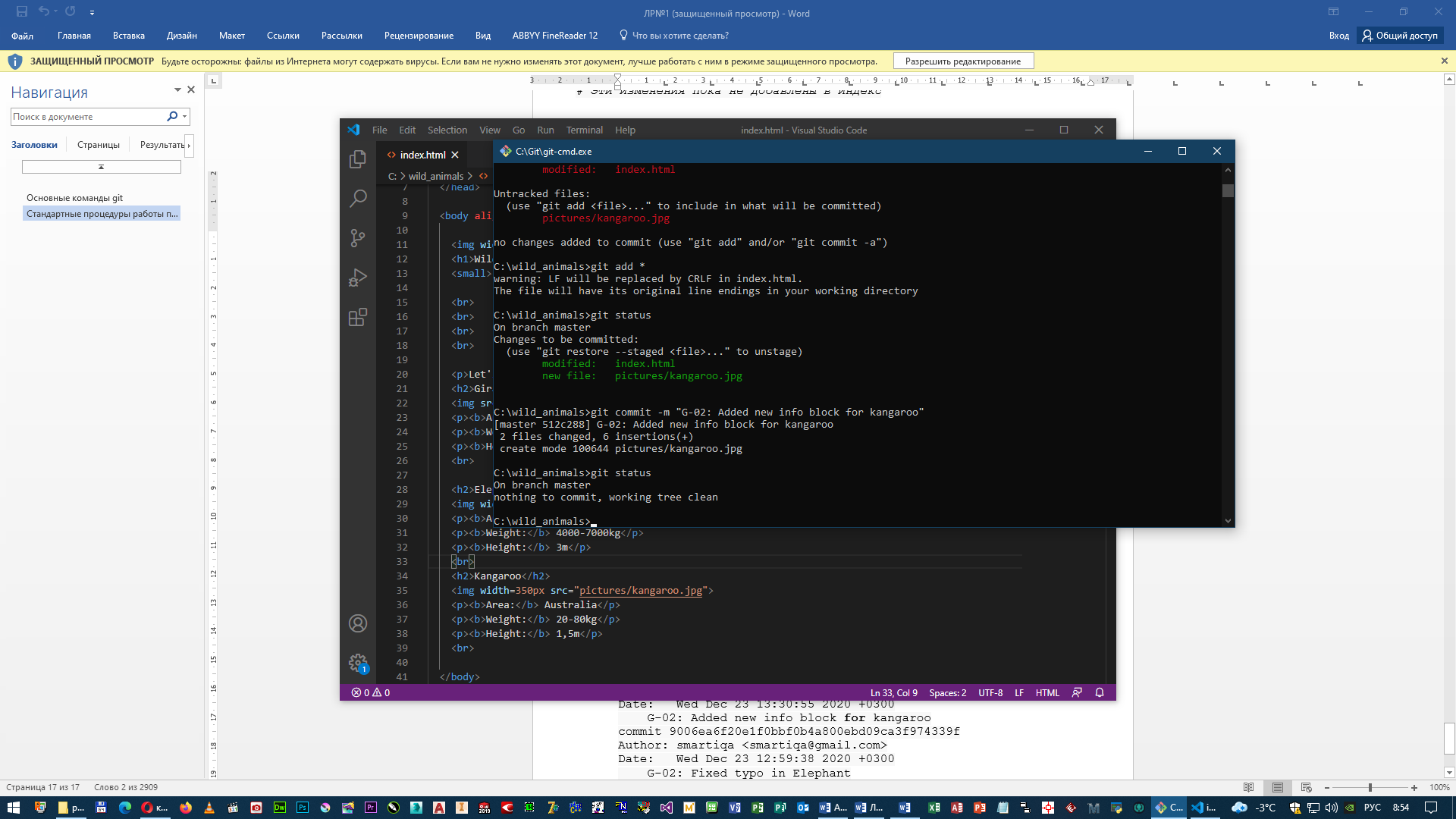


Рисунок 21 – Скриншот выполненного задания

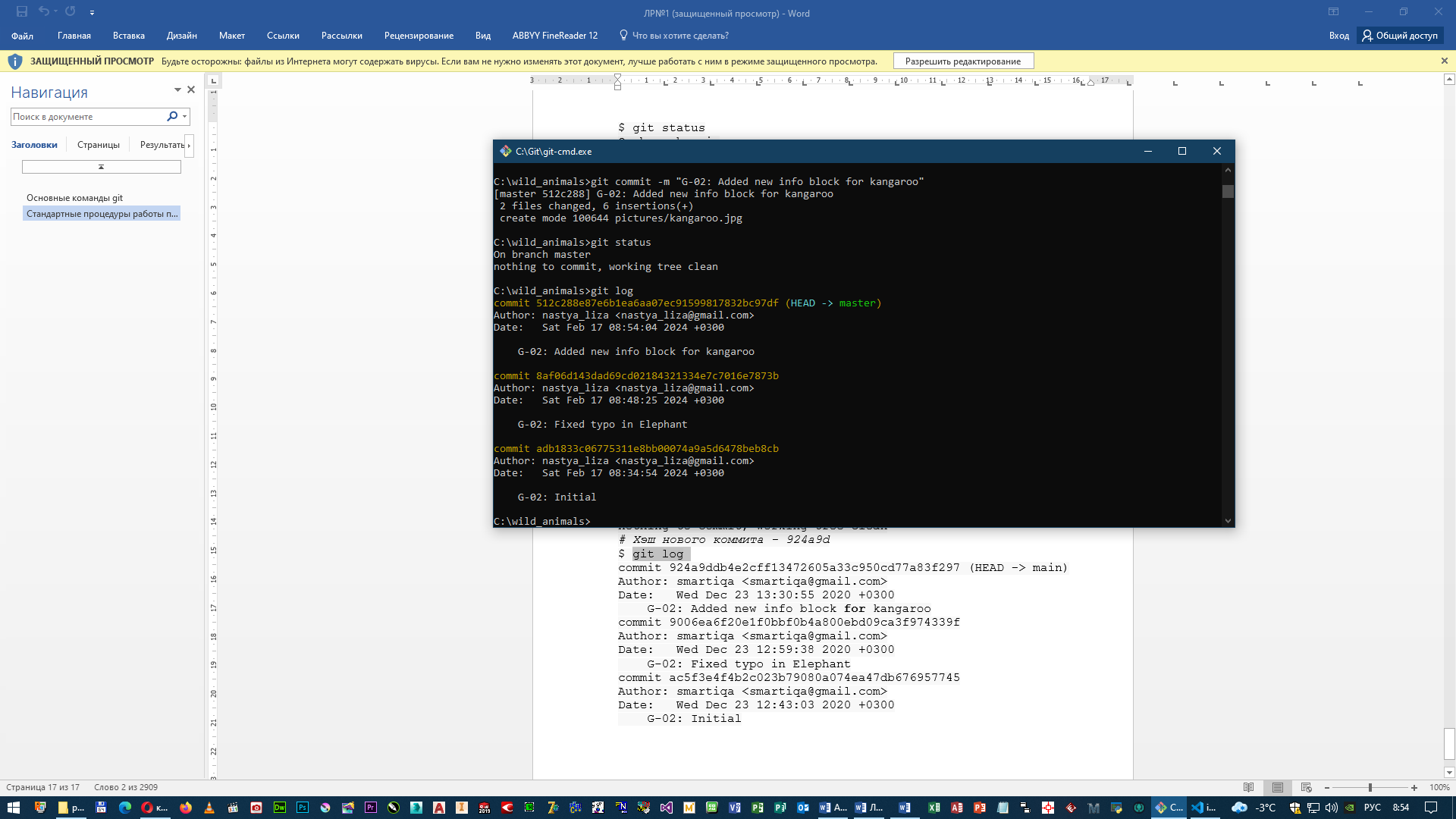


Рисунок 22 – Скриншот выполненного задания



Рисунок 23 – Скриншот выполненного задания

**Ответы контрольные вопросы**

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

Системы контроля версий - это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени. *Системы контроля версий (Version Control System, VCS)* применяются при работе нескольких человек над одним проектом.

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

VCS (Version Control System) - это система контроля версий, которая позволяет отслеживать изменения в файловой системе и управлять историей изменений. В VCS каждое изменение в файле сохраняется в виде commit (коммита), который представляет собой снимок состояния файлов в определенный момент времени. Каждый коммит имеет уникальный идентификатор и содержит информацию о том, какие файлы были изменены и какие изменения были внесены.

Хранилище (repository) - это место, где хранятся все коммиты и история изменений. Хранилище может быть централизованным или распределенным. В централизованной системе хранилище находится на сервере, к которому подключаются все пользователи. В распределенной системе каждый пользователь имеет свою собственную копию хранилища, что позволяет работать независимо и сливать изменения с другими пользователями.

История (history) - это список всех коммитов, которые были сделаны в хранилище. История позволяет просматривать все изменения, внесенные в файлы, и восстанавливать предыдущие версии файлов, если это необходимо. История также содержит информацию о том, кто и когда сделал каждый коммит.

Рабочая копия (working copy) - это локальная копия файлов из хранилища, которая находится на компьютере пользователя. Рабочая копия позволяет пользователю работать с файлами, вносить изменения и создавать новые коммиты. Пользователь может обновлять рабочую копию, чтобы получить последние изменения из хранилища, а также откатывать изменения, если что-то пошло не так.

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

Централизованные VCS - это системы, в которых хранилище файлов и история изменений находятся на центральном сервере. Пользователи подключаются к серверу и работают с файлами в рамках рабочей копии на своих компьютерах. Примером централизованной VCS является Subversion (SVN). В SVN все коммиты и изменения происходят в центральном хранилище, и пользователи получают обновления и отправляют свои изменения на сервер.

Децентрализованные VCS - это системы, в которых каждый пользователь имеет свою собственную копию хранилища и полную историю изменений. Каждый пользователь может работать независимо и делать коммиты в свою локальную копию хранилища. Примерами децентрализованных VCS являются Git и Mercurial. В Git каждый пользователь имеет полную копию репозитория, что позволяет работать офлайн и сливать изменения с другими пользователями при необходимости.

1. *Опишите действия с VCS при единоличной работе с хранилищем.*

При работе с VCS в одиночку вы можете выполнять следующие действия:

* + инициализация репозитория;
  + добавление файлов;
  + фиксация изменений;
  + откат изменений;
  + просмотр истории;
  + создание и переключение веток;
  + слияние веток;
  + удаление и переименование файлов;
  + отправка изменений на удаленный сервер.

1. *Опишите порядок работы с общим хранилищем VCS.*

При работе с общим хранилищем VCS (Version Control System) можно следовать следующему порядку действий:

* инициализация репозитория: создайте или подключитесь к существующему репозиторию;
* добавление файлов: добавьте файлы, которые хотите отслеживать;
* фиксация изменений: выполните коммит, чтобы зафиксировать изменения;
* обновление репозитория: получите последние изменения от других участников;
* создание и переключение веток: создайте и переключитесь на новую ветку для работы над задачами;
* слияние веток: слейте изменения из разных веток в основную ветку;
* отправка изменений на удаленный сервер: отправьте свои изменения на удаленный сервер;
* просмотр истории: просмотрите историю изменений в репозитории;
* удаление и переименование файлов: удалите или переименуйте файлы в репозитории.

1. *Краткая история Git.*

Git был создан Линусом Торвальдсом в 2005 году. Он разработал Git для управления разработкой ядра Linux и решил создать свою собственную систему контроля версий, потому что он был недоволен существующими решениями. Название "Git" происходит от английского сленга и означает "неприятного человека".

Git был разработан как низкоуровневый движок системы контроля версий, на основе которого можно создавать различные интерфейсы. Он был вдохновлен системами контроля версий BitKeeper и Monotone Впоследствии Git стал полноценной системой контроля версий, которую можно использовать напрямую.

Git быстро стал популярным и широко принятым инструментом разработки. Он обладает функциональностью, производительностью, безопасностью и гибкостью, которые нужны большинству команд и разработчиков Git стал де-факто стандартом в индустрии разработки программного обеспечения и широко используется во многих проектах.

1. *Каковы основные задачи, решаемые инструментальным средством git?*

Основные задачи Git:

* управление версиями кода и отслеживание изменений;
* работа с ветками для параллельной разработки различных задач;
* совместная работа и синхронизация изменений между разработчиками;
* откат изменений и восстановление предыдущих версий проекта;
* управление удаленными репозиториями для совместной работы и синхронизации кода.

1. *Назовите и дайте краткую характеристику командам git.*

Команды Git и их краткая характеристика:

* git init: инициализирует новый репозиторий Git;
* git clone <URL>: клонирует удаленный репозиторий на локальную машину;
* git add <file>: добавляет файлы в индекс для фиксации изменений;
* git commit -m "<message>": фиксирует изменения в репозитории с сообщением;
* git status: показывает текущее состояние репозитория;
* git diff: показывает различия между текущим состоянием и последним коммитом;
* git branch: показывает список веток в репозитории;
* git checkout <branch>: переключается на указанную ветку;
* git merge <branch>: сливает изменения из указанной ветки в текущую ветку;
* git remote: показывает список удаленных репозиториев;
* git fetch: получает последние изменения из удаленного репозитория;
* git push: отправляет локальные изменения на удаленный сервер;
* git pull: получает и объединяет изменения с удаленного репозитория;
* git config: устанавливает или просматривает настройки Git.

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

Локальный репозиторий:

* инициализация нового локального репозитория: git init;
* добавление файлов в индекс: git add <file>;
* фиксация изменений в репозитории: git commit -m "<message>";
* просмотр состояния репозитория: git status;
* просмотр различий между текущим состоянием и последним коммитом: git diff;
* создание новой ветки: git branch <branch>;
* переключение на другую ветку: git checkout <branch>;
* слияние изменений из одной ветки в другую: git merge <branch>.

Удаленный репозиторий:

* клонирование удаленного репозитория на локальную машину: git clone <URL>;
* просмотр списка удаленных репозиториев: git remote;
* получение последних изменений из удаленного репозитория: git fetch;
* отправка локальных изменений на удаленный сервер: git push;
* получение и объединение изменений с удаленного репозитория: git pull.

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

Ветви (branches) в Git позволяют разработчикам работать параллельно над разными задачами, изолировать изменения, управлять версиями кода, работать с различными средами и защищать ветки. Они обеспечивают гибкость и контроль над разработкой проекта.

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

При работе с Git вы можете игнорировать определенные файлы и директории при выполнении команды commit. Это полезно, когда вы не хотите, чтобы определенные файлы попадали в историю версий или распространялись с вашим репозиторием. Для этого вы можете использовать файл .gitignore, который содержит шаблоны для игнорирования файлов. Игнорирование файлов при работе с Git позволяет управлять содержимым репозитория и избегать добавления нежелательных файлов в историю версий. Файл .gitignore является мощным инструментом для определения правил игнорирования файлов и директорий.

**Вывод:** Во время выполнения лабораторной работы, мы изучили идеологию и применение средств контроля версий. Выработать умения использовать средства контроля версий при разработке программ.

# **Лабораторная работа №2**

**Тема:** Настройка и конфигурирование системы непрерывной интеграции.

**Цель:** Изучить идеологию и применение систем непрерывной интеграции и автоматической сборки проекта.

**Задание:** Изучить основные теоретические сведения. Выполнить задания 1-2.

**Технологическое оснащение:** Методические указания к выполнению, ОС Windows, текстовый процессор MS Word 2010, СКВ Git.

**Результаты выполнения работы**

**Задание 1.** Провести сравнительный анализ ПО для CI. Результат представить в виде таблицы.

В таблице 1 представлен сравнительный анализ ПО для CI.

Таблица 1 – Сравнительный анализ ПО для CI

|  |  |  |
| --- | --- | --- |
| Наименование | Назначение, принципы работы | Достоинства/Недостатки |
| Jenkins | Jenkins предоставляет возможность тестирования кода в режиме реального времени, а также дает возможность получать отчеты об отдельных изменениях в обширной кодовой базе. Этот инструмент, главным образом, позволяет разработчикам быстро помечать и исправлять ошибки и баги в коде, а затем автоматически тестировать сборку кода. | Достоинства:   * позволяет разработчикам быстро помечать и исправлять ошибки и баги в коде; * благодаря интуитивному пользовательскому интерфейсу Jenkins очень легко настраивать и конфигурировать; * наличие больше 1000 плагинов позволяет автоматизировать практически что угодно. |
| TeamCity | TeamCity это сервер непрерывной интеграции корпоративного уровня. Он поддерживает большое количество мощного функционала, а также имеет очень надежную бесплатную версию для маленьких проектов (до 100 конфигураций сборки) | Достоинства:   * обширная поддержка множества плагинов с открытым исходным кодом; * поддержка .NET; * высокая надежность, не зависящей от запуска сборок. |
| Travis CI | Travis это очень популярный инструмент непрерывной интеграции (больше 7 тысяч звезд на GitHub). Он бесплатен для проектов с открытым исходным кодом. Этот инструмент можно назвать не кроссплатформенным, а платформо-независимым, поскольку что это веб-ресурс. | Достоинства:   * поддержка множества языков программирования и конфигураций сборки; * легко настраивать интерфейс, а инсталлировать и вовсе не нужно. |
| Gitlab CI | С помощью Gitlab вы можете изменять метаданные merge-реквеста без команд slash в комментарии. По сути это веб-приложение с API, сохраняющим его состояние в базе данных. | Достоинства:   * предоставляет дружественный к пользователю, интуитивный интерфейс. |
| Circle CI | Circle CI это гибкий сервер непрерывной интеграции, запускаемый в любой среде: в кросс-платформенном мобильном приложении, Python API сервере или кластере Docker. Инструмент нацелен на обозначение багов и улучшение качества приложения в целом (в долгосрочной перспективе). | Достоинства:   * хорошо интегрируется с системами контроля версий; * поддерживает много языков, а также Docker. |
| GoCD | GoCD это сервер непрерывной интеграции, собравший больше 4,5 тысяч звезд на GitHub. Используется для моделирования и визуализации сложных рабочих процессов. GoCD также является инструментом непрерывной доставки, таким образом, это прекрасное решение для CI/CD процессов. | Достоинства:   * высокая степень безопасности; * благодаря такому функционалу как value stream map вы можете визуализировать весь рабочий процесс в одном месте; * регулярно создаются и добавляются тысячи плагинов. |
| Bamboo | Это build-сервер непрерывной интеграции, осуществляющий сборку, тестирование и релизы. Он создан для поддержки его «сестринского» инструмента Jira от Atlassian и систем контроля версий. Bamboo поддерживает много языков программирования и технологий, например, AWS Buckets. | Достоинства:   * простой в использовании интерфейс; * содержит встроенный git branching, а также интегрируется с HipChat. |
| Codeship | Codeship это популярный инструмент непрерывной интеграции, автоматизирующий как процесс разработки, так и процесс деплоймента. Он запускает автоматизированный рабочий процесс, упрощая отправку в хранилище. | Достоинства:   * поддерживает шифрованное внешнее кэширование образов Docker; * отличный функционал отладки, включая возможность отладки прямо из среды непрерывной интеграции; * приятная панель инструментов. |
| Buildbot | Buildbot это фреймворк для сборки инструмента тестирования и непрерывной интеграции, автоматизирующий цикл компиляции/тестирования. Этот фреймворк используется для валидации изменений во многих проектах, поскольку предлагает очень распределенное параллельное выполнение задач. | Достоинства:   * централизованные CI/CD; * поддержка разнообразных платформ тестирования. |
| StriderCD | Strider это CI-инструмент с открытым исходным кодом. Этот инструмент предлагает поддержку различных плагинов, модифицирующих схему базы данных и регистрирующих маршруты HTTP. | Достоинства:   * позволяет вам публиковать и даже подписываться на события сокетов; * поддерживает большое количество плагинов для изменения дефолтной конфигурации; * имеет очень хорошую поддержку Docker; * возможность создавать собственный пользовательский интерфейс. |

**Задание 2.**

На рисунке 1 представлено предоставление справки о Git командой – git help.

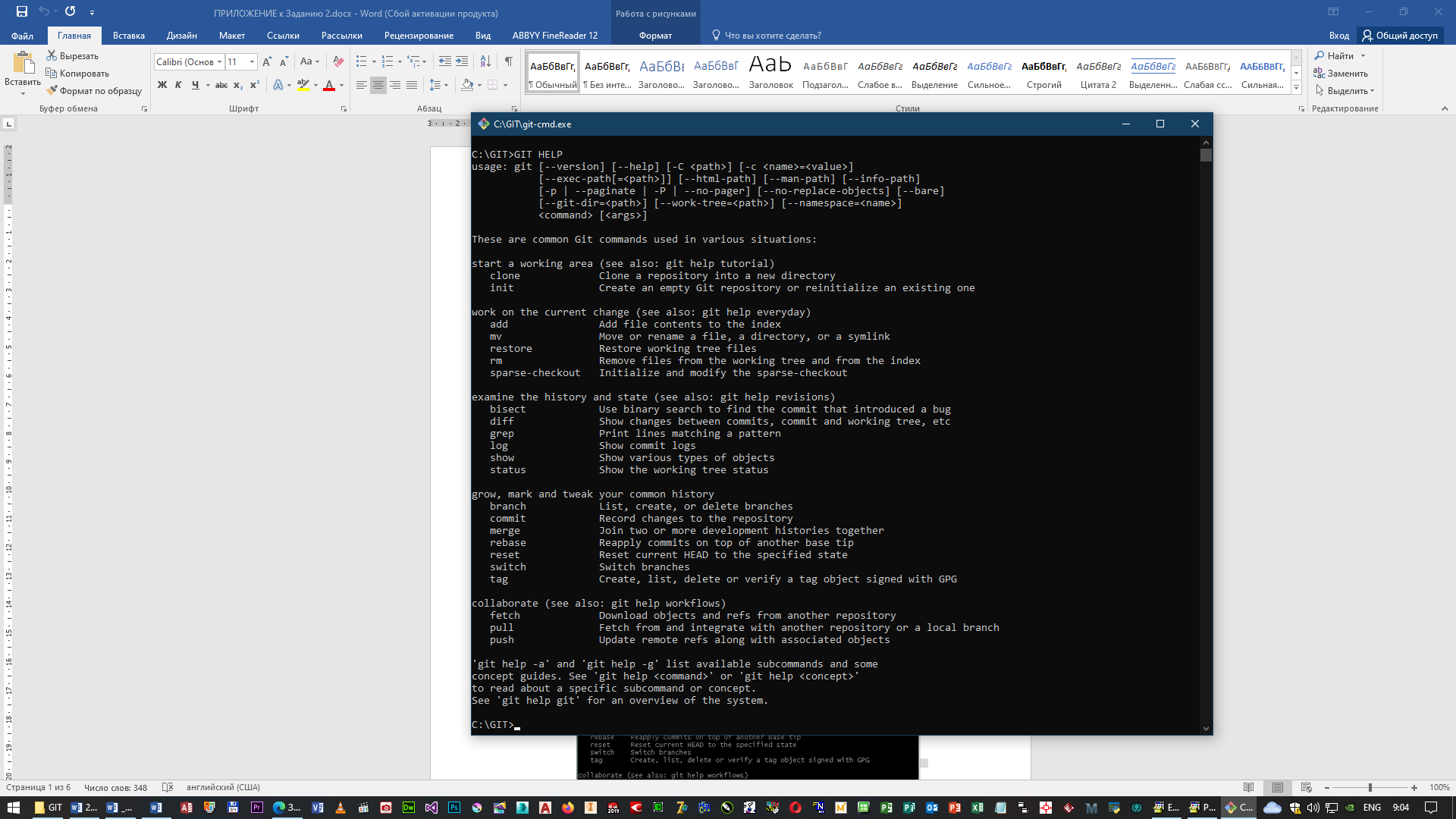


Рисунок 1 – Представленная справка

Выбираем папку, в которой будем создавать свои репозитории с помощью команды – cd C:/GIT/projects. Затем выполняем команду git init для создания подкаталога с именем .git. Репозиторий создан. На рисунке 2 представлен результат создания репозитория.

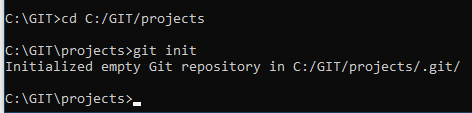


Рисунок 2 – Результат создания репозитория

На рисунке 3 представлена настройка файла конфигураций с помощью команд «$git config --global user.name "XXX"» и «$ git config --global user.email [XXX@XXX.com](mailto:XXX@XXX.com)» и просмотр настроек конфигураций с помощью команды «git config –list».

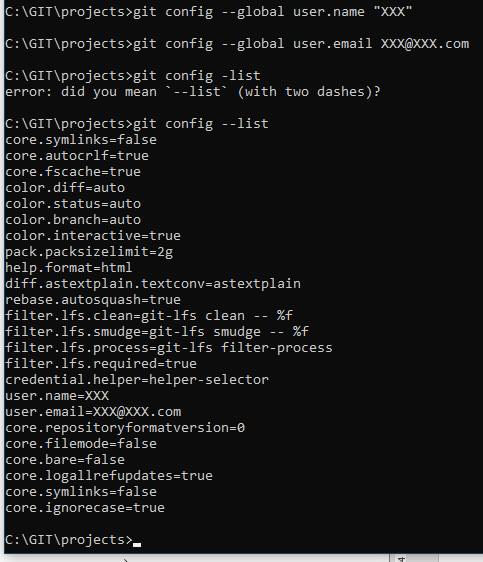


Рисунок 3 – Настройка и просмотр файла конфигураций

На рисунке 4 представлено изменение в папке после создания репозитория.

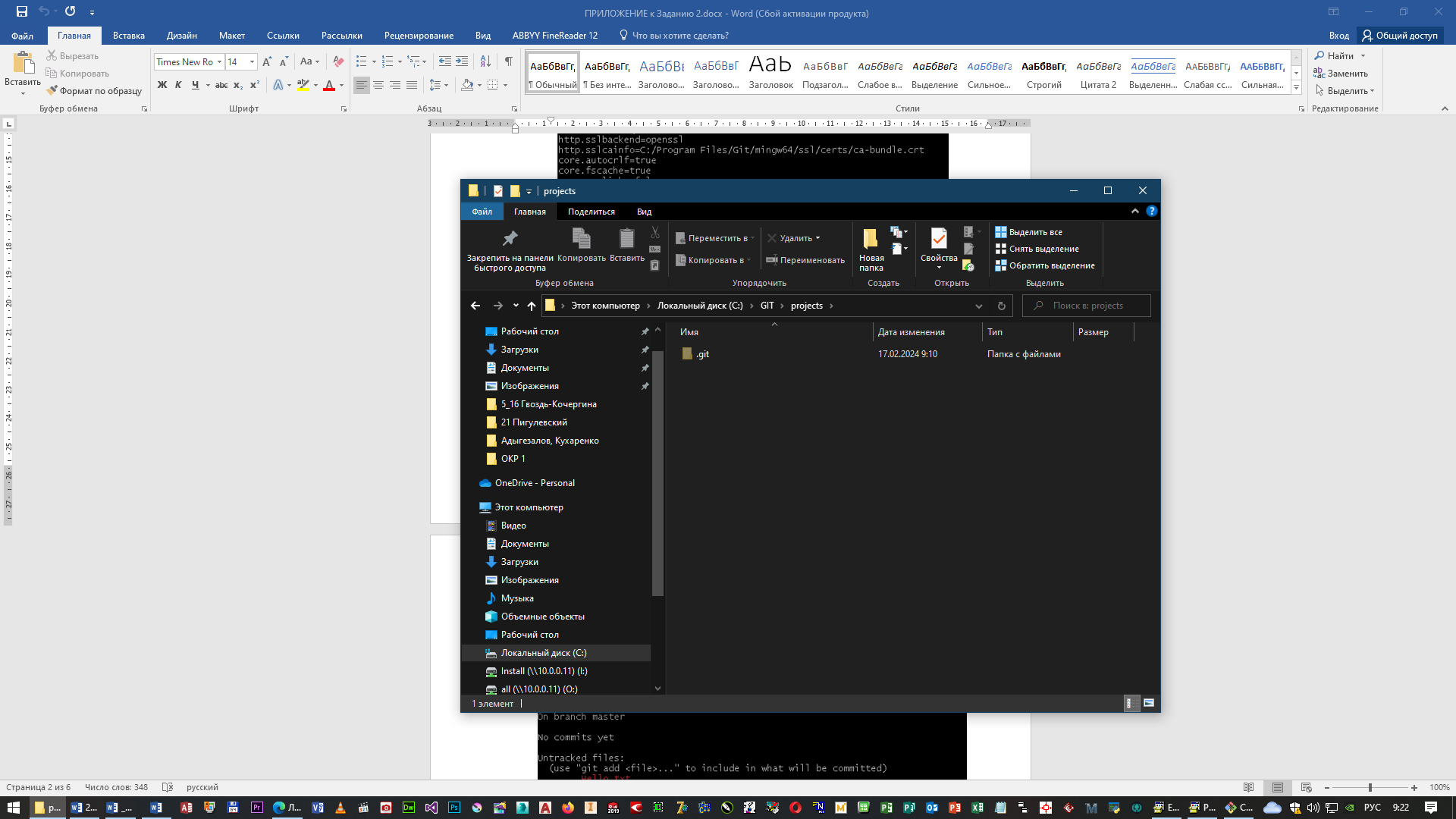


Рисунок 4 – Изменение в папке после создания репозитория

На рисунке 5 представлен просмотр статуса репозитория с помощью команды git status, в результате которого выводятся все файлы и каталоги в репозитории, где написаны изменения. Файл был добавлен с помощью команды git add, в результате чего он был добавлен в главную ветку main.

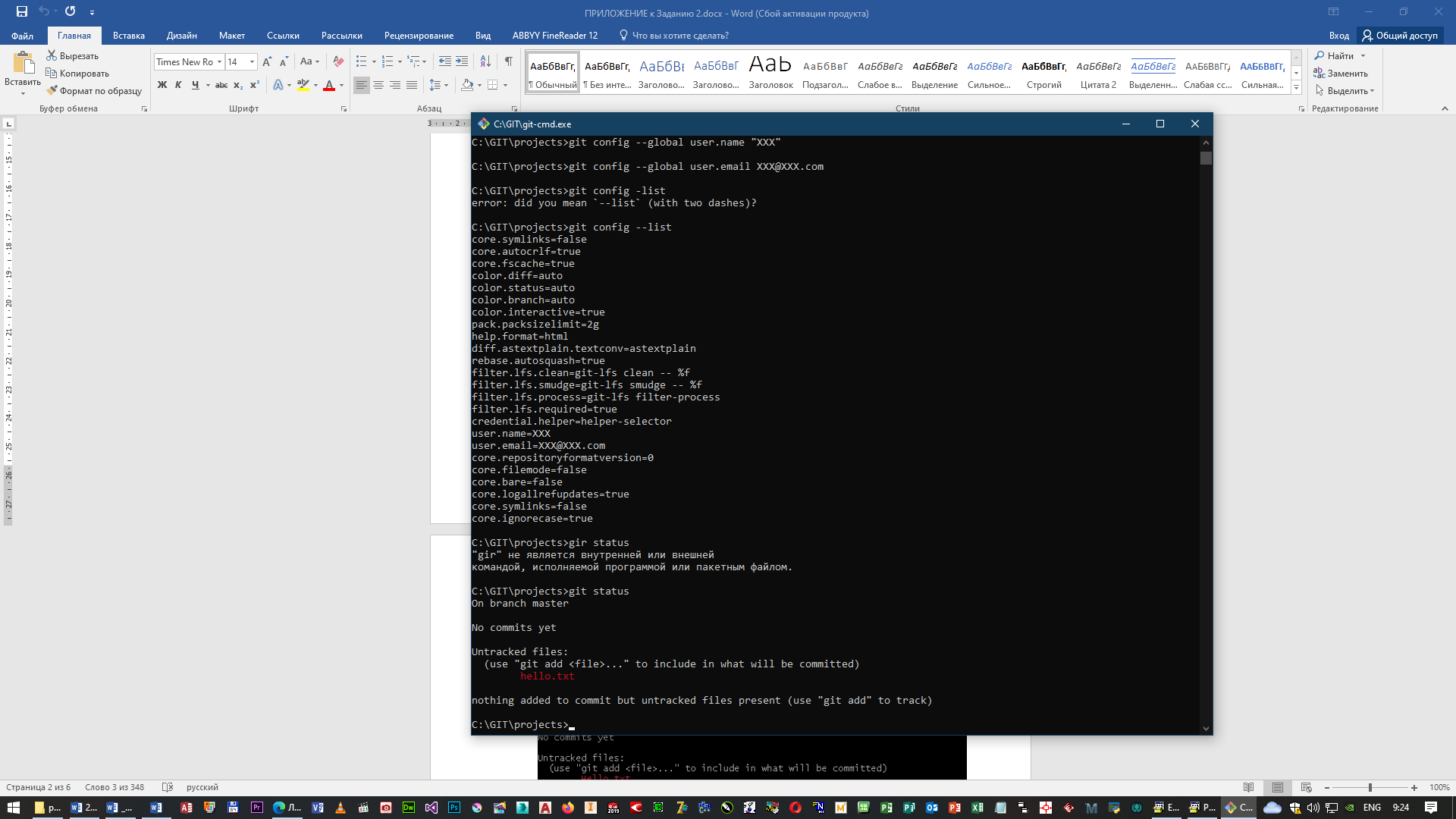


Рисунок 5 – Просмотр статуса репозитория

На рисунке 6 представлен просмотр журнала.

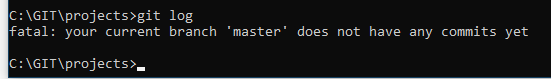


Рисунок 6 – Просмотр журнала

На рисунке 7 представлен статус репозитория после добавления нового файла и изменения старого.

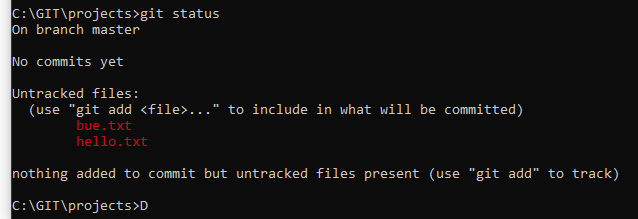


Рисунок 7 – Статус репозитория после добавления нового файла и изменения старого файла

На рисунке 8 представлено добавление коммита.

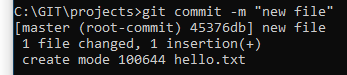


Рисунок 8 – Добавление коммита

На рисунке 9 представлено создание новой ветки и переход в нее с помощью команд branch и checkout.

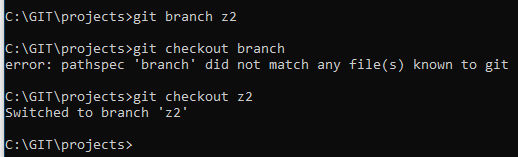


Рисунок 9 – Создание новой ветки и переход в нее

На рисунке 10 представлен выполнение команды git log, где представлено, что head указывает на созданную ветку «z2».

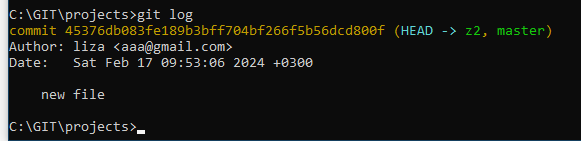


Рисунок 10 – Выполнение команды git log

На рисунке 11 представлен просмотр содержимого всех файлов в папке.

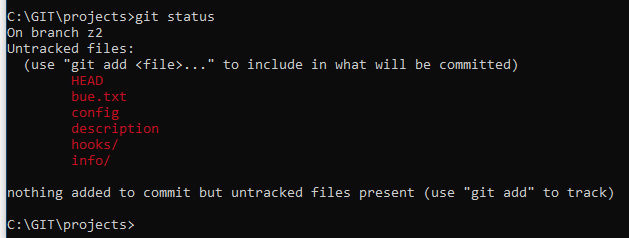


Рисунок 11 – Просмотр содержимого всех файлов в папке

На рисунке 12 представлено создание нового файла и добавление коммита.

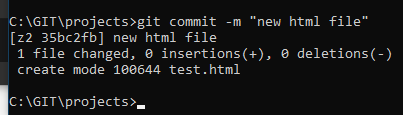


Рисунок 12 – Создание нового файла и добавление коммита

На рисунке 13 представлено создание новой ветки master и просмотр всех веток.

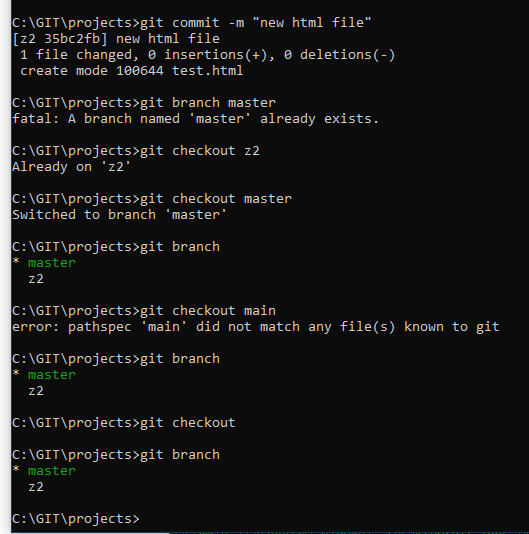


Рисунок 13 – Создание новой ветки и просмотр всех веток

На рисунке 14 представлено слияние веток.

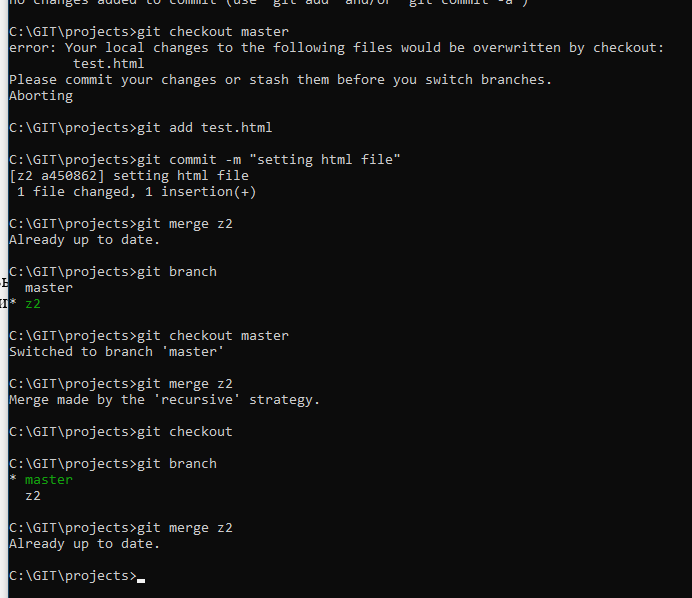


Рисунок 14 – Слияние веток

На рисунке 15 представлен созданный аккаунт в GitHub.

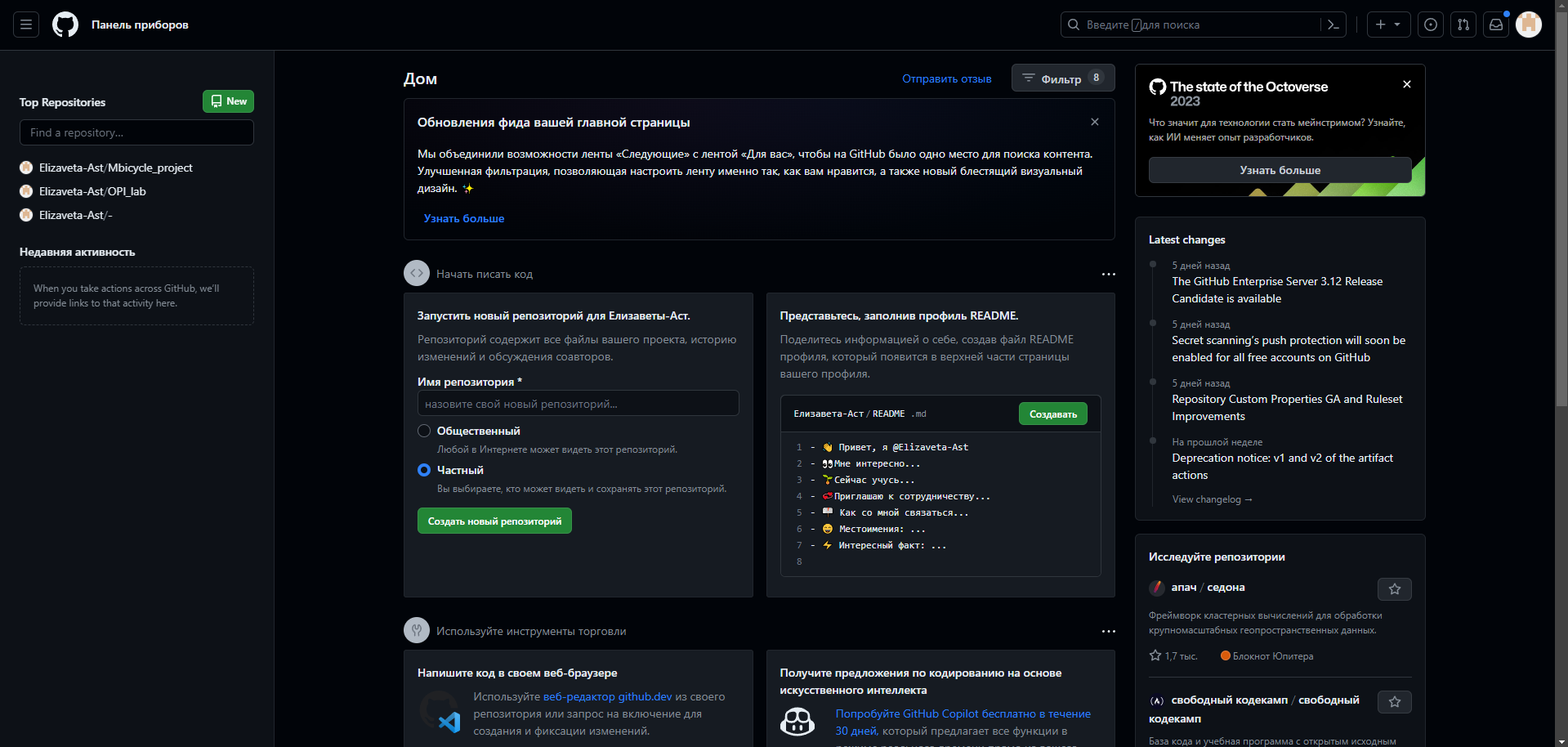


Рисунок 15 – Созданный аккаунт GitHub

На рисунке 16 представлен созданный репозиторий OPI\_lab с новым файлом readme.txt.

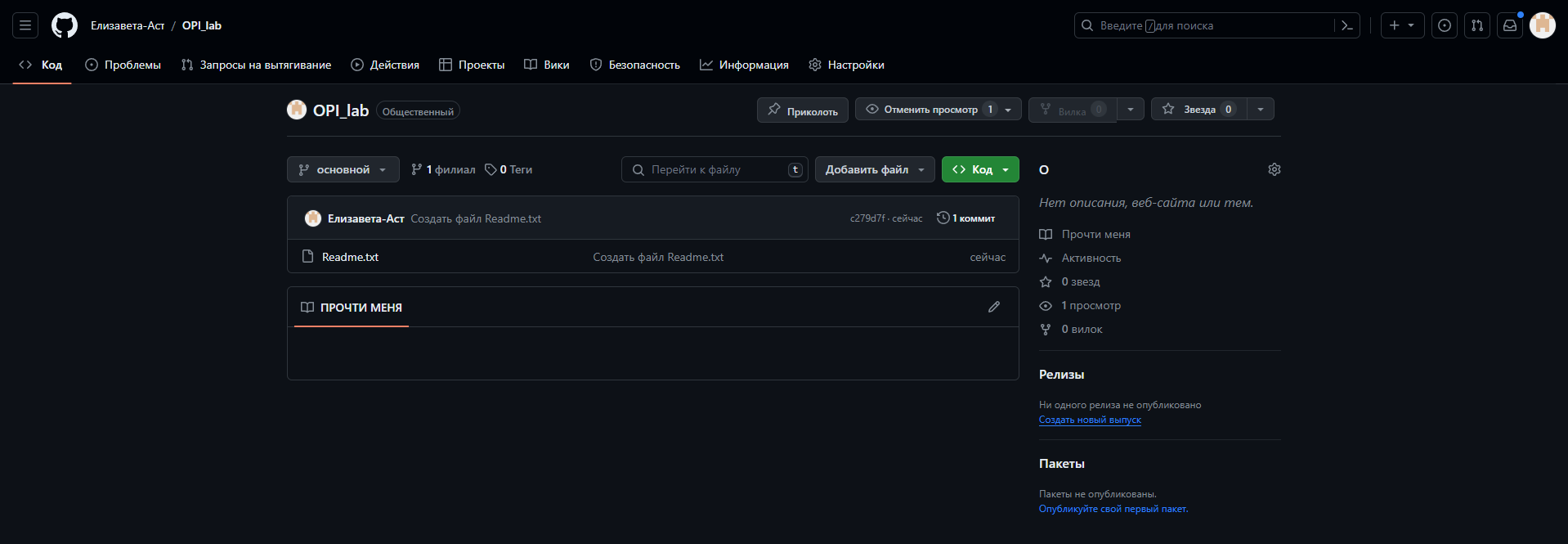


Рисунок 16 – Созданный репозиторий OPI\_lab с новым файлом readme.txt

На рисунке 17 представлено изменение файла.

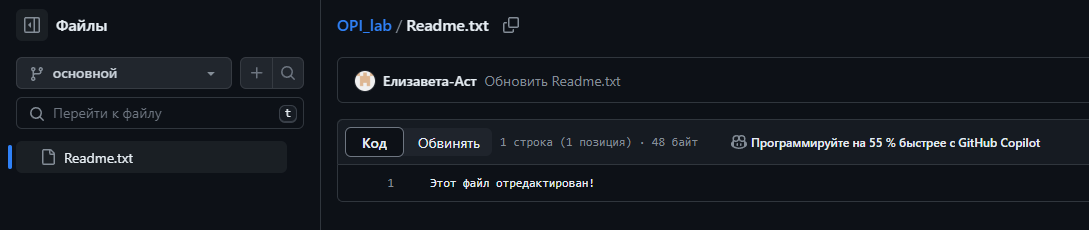


Рисунок 16 – Изменение файла

На рисунке 17 представлено информация о работе с репозиторием, а то есть файлом.

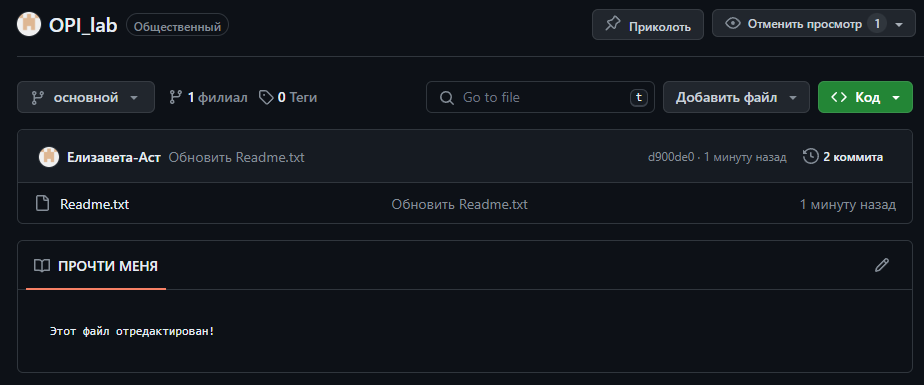


Рисунок 17 – Информация о работе с репозиторием, а то есть файлом

На рисунке 18 представлено клонирование репозитория.

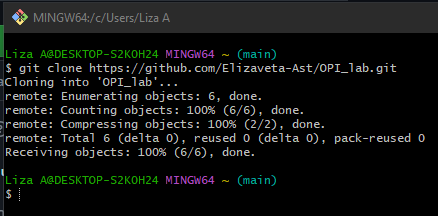


Рисунок 18 – Клонирование репозитория

На рисунке 19 представлен просмотр конфигурационного файла локального репозитория.



Рисунок 19 – Конфигурационного файла локального репозитория

На рисунке 20 представлено внесение изменение и их фиксирование в файле.

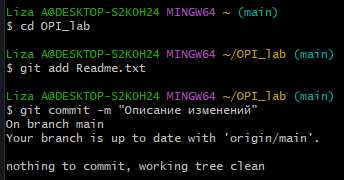


Рисунок 20 – Внесение изменение и их фиксирование в файле

На рисунке 21 представлен список настроек удаленного репозитория.

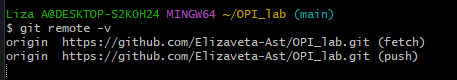


Рисунок 21 – Список настроек удаленного репозитория

На рисунке 22 представлена отправка в удаленный репозиторий.



Рисунок 22 – Отправка в удаленный репозиторий

Рисунок 23

**Вывод:** Во время выполнения мы изучили идеологию и применение систем непрерывной интеграции и автоматической сборки проекта.