МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

**ФИЛИАЛ БГТУ**

**«БОБРУЙСКИЙ ГОСУДАРСТВЕННЫЙ ЛЕСОТЕХНИЧЕСКИЙ КОЛЛЕДЖ»**

Специальность 2-40 01 01

«Программное обеспечение

информационных технологий»

**ОТЧЁТ**

по лабораторным работам

**Учебная дисциплина:** Разработка программ с использованием системы управления версиями

**Выполнил уч-ся** гр.ПО-5 С.Р. Тарасов

**Руководитель** \_\_\_\_\_\_\_\_\_\_\_\_ Г.Г. Булыго

Бобруйск

2025

Содержание

[Лабораторная работа №1 3](#_Toc191930114)

[Лабораторная работа №2 10](#_Toc191930115)

**Лабораторная работа №1**

**Номер учебной группы:** ПО-5

**Дата выполнения работы:** 18.02.2025

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

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

**Задание:** Изучить основные теоретические сведения. Выполнить задания 1-3.

**Оснащение работы:** Методические указания к выполнению, ОС Windows, Word, Sublime Text.

**Выполнение работы**

**Задание 1–2.** 1. Подготовить рабочее окружение в соответствии с типом операционной системы.

2. Установить Git, в соответствии с приведенной инструкцией по установке и настройке Git для пользователей Windows.

Подготовить и изучить исходные данные для выполнения работы.

Дана папка, в которой присутствуют файлы блога о диких животных.

На рисунке 1 представлена структура папки.

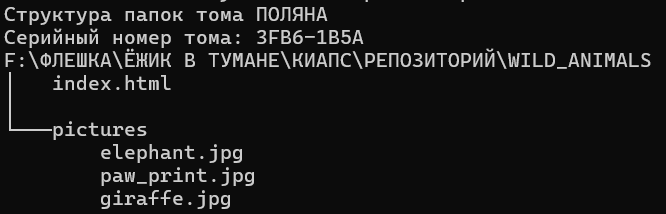


Рисунок 1 – Структура папки

**Задание 3.** В ходе работы необходимо создать на основе директории **wild\_animals** репозиторий Git и исследовать его некоторые изменения, после чего зафиксировать их средствами Git.

Созданный репозиторий внутри папки wild\_animals представлена на рисунке 2.



Рисунок 2 – Создание репозитория

Настройка пользователя Git на уровне репозитория wild\_animals представлена на рисунках 3 – 5. На рисунке 3 видно, что отсутствует файл индекса (.git/index), папка с объектами Git пустая (.git/objects).

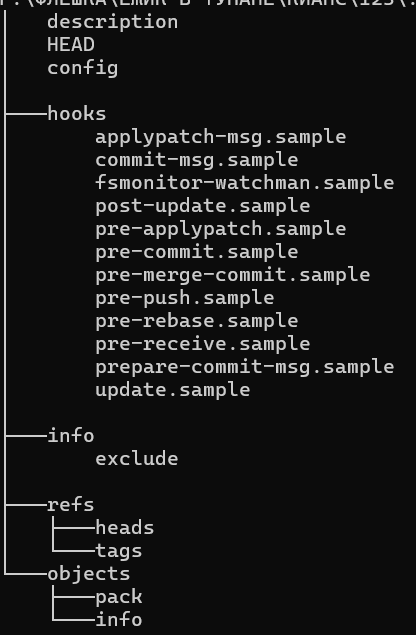


Рисунок 3 – Содержимое файла конфигурации Git

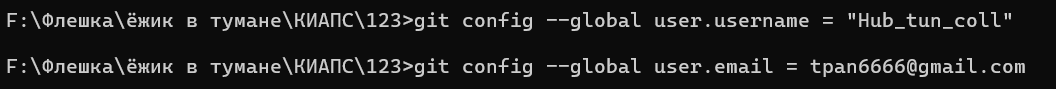


Рисунок 4 – Настройка имя и email пользователя

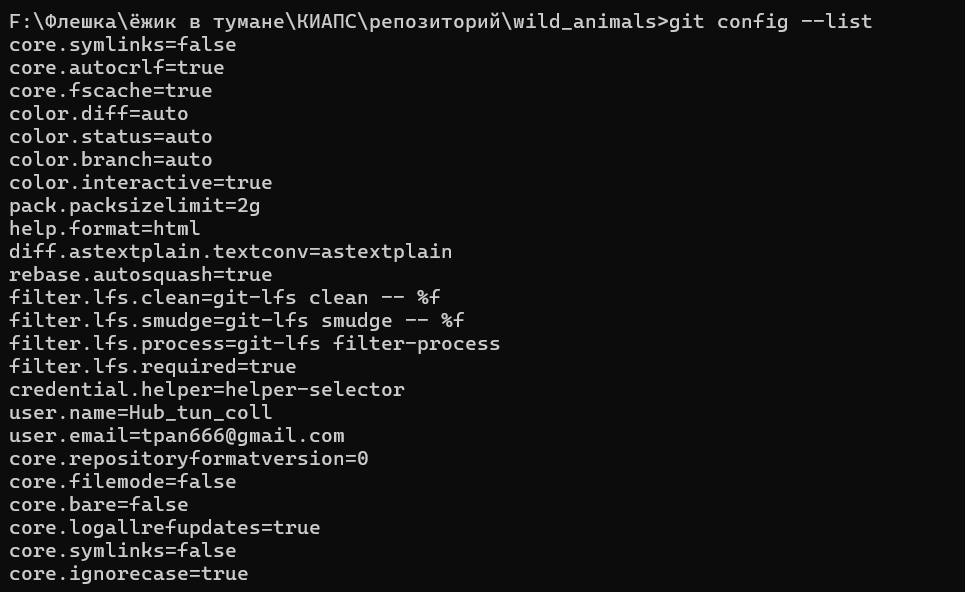


Рисунок 5 – Проверка настройки имени и email пользователя

На рисунке 6 можно убедиться, что указатель HEAD указывает на ветку master.



Рисунок 6 – Просмотр указателя HEAD

Сделал файлы папки wild\_animals отслеживаемыми. На рисунке представлен результат отслеживания репозитория.

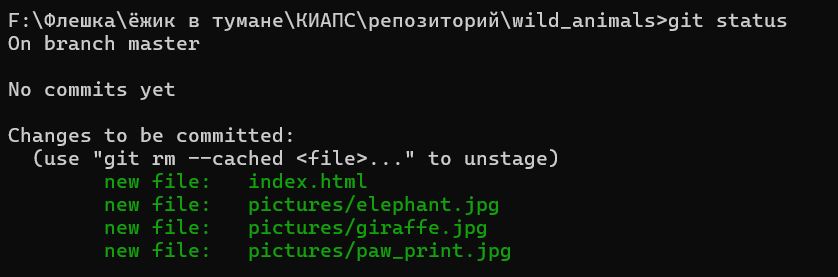


Рисунок 7 – Отслеживание репозитория

На рисунке 8 представлена папка с объектами (.git/objects).



Рисунок 8 – Папка с объектами после отслеживания файлов

На рисунке 9–10 представлено создание коммита и его хеш.

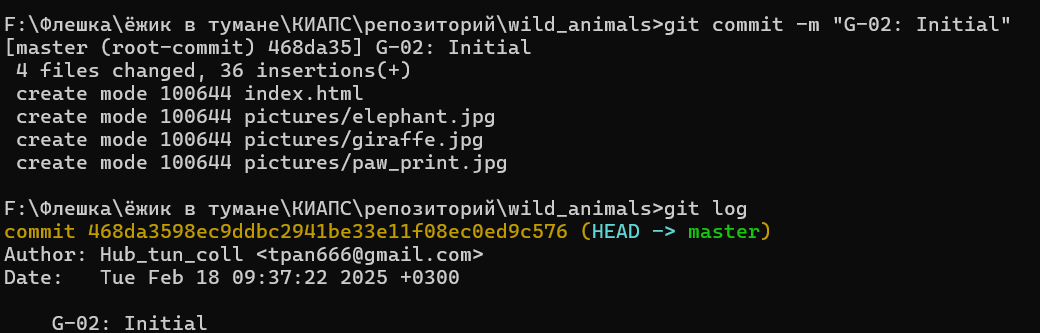


Рисунок 9 – Создание коммита и просмотр его хеш

Исправил опечатку в файле index.html. На рисунке 11 представлен результат выполнения.

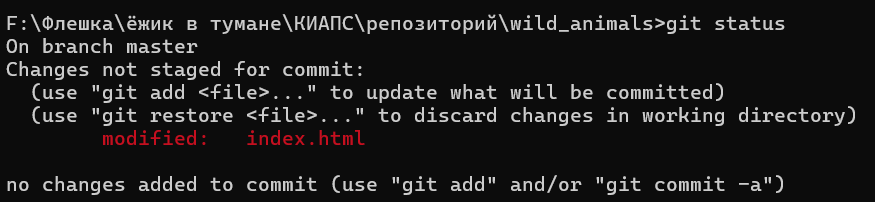


Рисунок 11 – Статус после изменения

Сделал отслеживаемым файлом исправленный index.html и создал новый коммит. На рисунке 12 представлен результат выполнения.

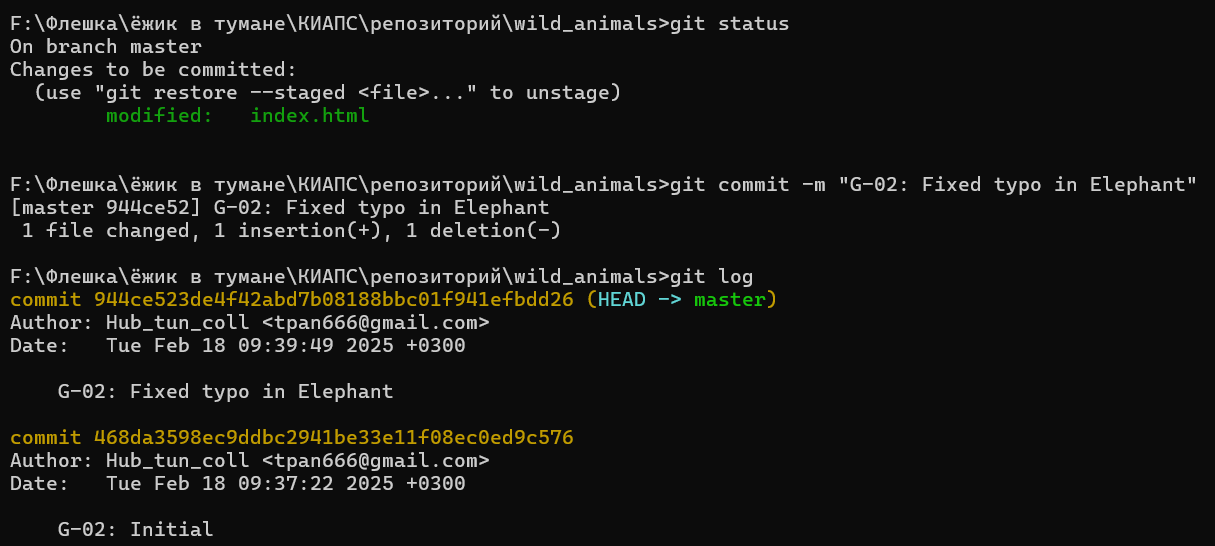


Рисунок 12 – Созданный коммит

Добавить в файл index.html секцию для еще одного животного. На рисунке 13 представлен результат выполнения.

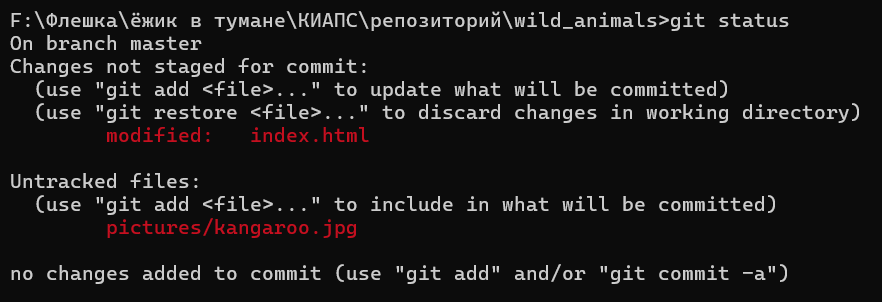


Рисунок 13 – Добавление новой секции

После сделал файлы отслеживаемыми и создал новый коммит. На рисунке 14 представлен результат выполнения.

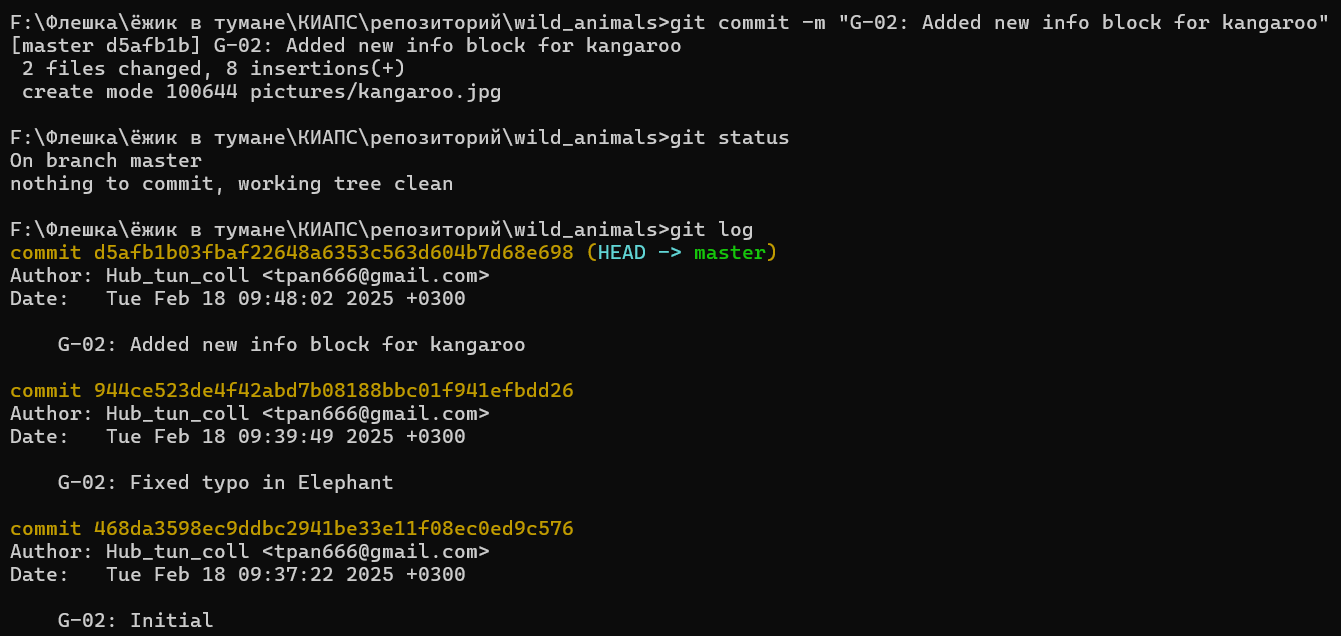


Рисунок 14 – Создание третьего коммита

**Ответы на контрольные вопросы**

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

Система контроля версий (VCS) — это инструмент для отслеживания изменений в исходном коде и других файлах проекта. Основные задачи VCS:

* Управление версиями файлов;
* Сохранение истории изменений;
* Совместная работа с несколькими людьми;
* Объединение изменений, сделанных разными пользователями;
* Возможность отката к предыдущим версиям файлов;
* Обеспечение безопасности и возможности восстановления данных.

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

* Хранилище — это место, где сохраняются все файлы проекта и их истории изменений. Может быть локальным или удалённым;
* Commit — операция, которая сохраняет изменения в хранилище с добавлением сообщения, описывающего эти изменения;
* История — последовательность всех изменений, сохранённых в хранилище с помощью commit-ов;
* Рабочая копия — это локальная версия проекта, с которой пользователь работает. Она может быть не синхронизирована с хранилищем.

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

* В централизованных VCS (CVCS) все изменения происходят в центральном репозитории, и пользователи подключаются к этому репозиторию для получения и отправки изменений. Примеры: Subversion (SVN), CVS.
* В децентрализованных VCS (DVCS) каждый разработчик имеет локальную копию всего репозитория, включая всю историю изменений. Пользователи могут работать автономно и синхронизировать изменения по мере необходимости. Примеры: Git, Mercurial.

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

* При единоличной работе с VCS, пользователь:
* Инициализирует репозиторий;
* Работает с рабочей копией;
* Вносит изменения в файлы;
* Сохраняет изменения с помощью commit;
* Осуществляет обновления и синхронизацию с хранилищем;
* Может откатить изменения, если нужно.

1. Опишите порядок работы с общим хранилищем VCS.

* При работе с общим хранилищем:
* Каждый разработчик клонирует репозиторий;
* Работает с локальной копией, внося изменения;
* Регулярно синхронизирует свою копию с центральным репозиторием, получая последние изменения от других разработчиков (pull);
* Отправляет свои изменения в репозиторий (push);
* Разрешает конфликты, если несколько пользователей изменяли одни и те же участки кода.

1. Краткая история Git.

Git был разработан в 2005 году Линусом Торвальдсом, создателем Linux, для управления исходным кодом ядра Linux. Он был создан как децентрализованная система контроля версий, которая могла бы эффективно работать с большими проектами и позволяла бы пользователям работать автономно.

1. Каковы основные задачи, решаемые инструментальным средством git?

* Управление версиями исходного кода;
* Ветвление и слияние изменений;
* Управление репозиториями;
* Сохранение и восстановление истории изменений;
* Совместная работа над проектами, синхронизация изменений.

1. Назовите и дайте краткую характеристику командам git.

* git init — инициализация нового репозитория;
* git clon — создание копии существующего репозитория;
* git add — добавление изменений в индекс (предкоммит);
* git commit — фиксация изменений в репозитории;
* git status — отображение текущего состояния рабочего каталога и индекса;
* git pull — получение изменений из удалённого репозитория и слияние с локальными;
* git push — отправка изменений в удалённый репозиторий;
* git branch — управление ветками;
* git merge — слияние веток;

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

Локальный репозиторий:

* git init — инициализация нового репозитория;
* git commit -m "Сообщение коммита" — сохранение изменений.

Удалённый репозиторий:

* git clone <url> — клонирование репозитория;
* git pull origin master — получение изменений с удалённого репозитория;
* git push origin master — отправка изменений в удалённый репозиторий.

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

Ветки (branches) — это способы создания параллельных линий разработки в Git. Ветки позволяют:

* Разрабатывать новые функции или исправления без воздействия на основную версию кода;
* Параллельно работать нескольким разработчикам;
* Изолировать экспериментальные изменения;
* Позволяют впоследствии сливать изменения обратно в основную ветку (например, master).

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

В Git можно игнорировать файлы, добавляя их в файл .gitignore. Используется, чтобы не добавлять временные файлы, логи или файлы, которые не должны попадать в репозиторий (например, файлы конфигурации, созданные средой разработки).

**Вывод:** На лабораторной работе научился использовать инструмент для контроля версий git.

**Лабораторная работа №2**

**Номер учебной группы:** ПО-5

**Дата выполнения работы:** 19.02.2025

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

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

**Задание:** Изучить основные теоретические сведения. Выполнить задания 1-3.

**Оснащение работы:** Методические указания к выполнению, ОС Windows, Word, Sublime Text.

**Выполнение работы**

**Задание 1**

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

|  |  |  |
| --- | --- | --- |
| **Наименование** | **Назначение, принципы работы** | **Достоинства/Недостатки** |
| **1** | **2** | **3** |
| Jenkis | Автоматизация сборки, тестирования и развёртывания. Работает по принципу мастер-агентов | Достоинства:  – Бесплатный с открытым исходным кодом  – Большое количество плагинов  – Гибкость настройки  Недостатки:  – Требует сложной конфигурации  – Высокая нагрузка на сервер  – Устаревший интерфейс |
| TeamCity | CI/CD-система от JetBrains. Поддерживает мощную систему билд-агентов и конфигураций. | Достоинства:  – Интуитивный интерфейс;  – Гибкая настройка билда;  – Хорошая интеграция с инструментами JetBrains.  Недостатки:  – - Платная (бесплатна только для малых проектов);  – Высокие требования к ресурсам. |
| Travis CI | Облачная CI/CD-система, тесно интегрирована с GitHub. Работает по принципу YAML-конфигураций. | Достоинства:  – Простая настройка;  – Поддержка множества языков;  – Хорошая интеграция с GitHub.  Недостатки:  – Ограничения в бесплатной версии;  – Медленная работа на |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
|  |  | больших проектах. |
| GitLab CI | Встроенная в GitLab система CI/CD. Работает с .gitlab-ci.yml. | Достоинства:  – Полная интеграция с GitLab;  – Хорошая безопасность;  – Поддержка Docker и Kubernetes.  Недостатки:  – Требовательность к ресурсам;  – Сложность настройки Runner'ов. |
| CircleCI | Облачная и локальная CI/CD-платформа. Работает с контейнерами и виртуальными машинами. | Достоинства:  – Хорошая масштабируемость;  – Интеграция с Docker;  – Высокая скорость выполнения.  Недостатки:  – Бесплатная версия с ограничениями;  – Может быть дорогой при большом числе билдов. |
| GoCD | Open-source CI/CD от ThoughtWorks, фокусируется на pipeline-стратегиях. | Достоинства:  - Хорошая визуализация pipeline'ов;  – Поддержка сложных workflow.  Недостатки:  – Сложность конфигурации;  – Малое количество плагинов. |

**Задание 2.**

На рисунке 1изображено отображение версии git.



Рисунок 1 – Версия git

Для получения справки по Git использовал команду «git --help».

Описание команд изображено на рисунке 2.

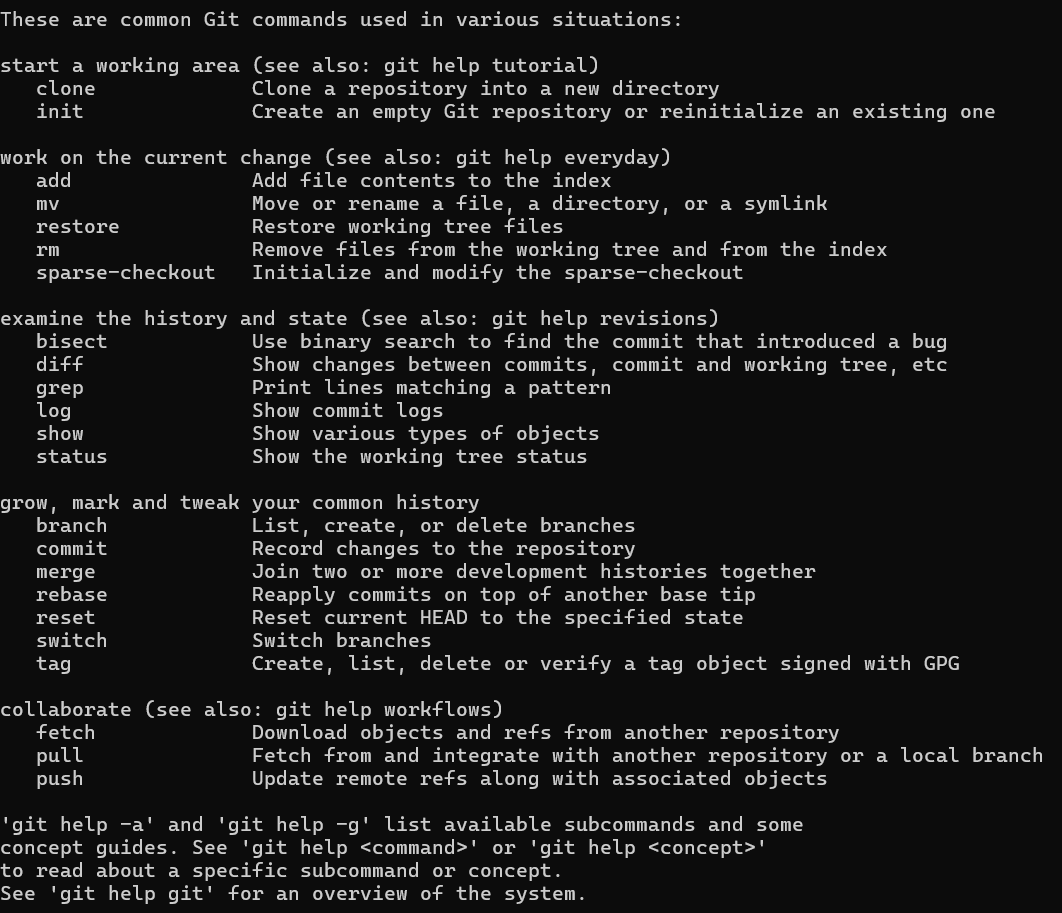


Рисунок 2 – Описание команд

Создал репозиторий внутри папки «репозиторий», для этого я с помощью команды «cd» перешёл в папку, затем применил команду «git init».

Созданный репозиторий внутри папки представлена на рисунке 3.

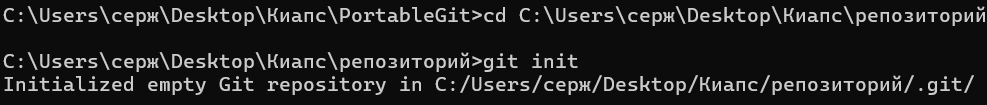


Рисунок 3 – Создание репозитория

Настройка пользователя Git на уровне репозитория представлена на рисунке 4.

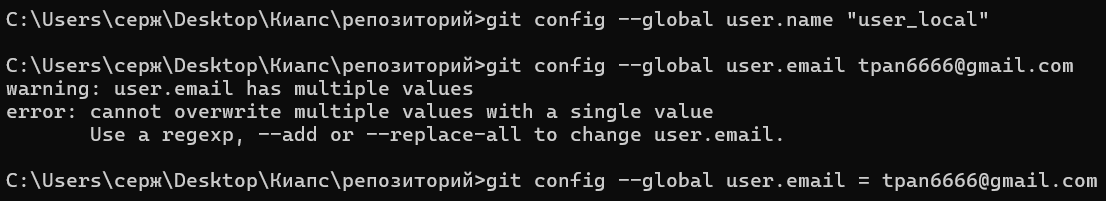


Рисунок 4 – Настройка имени и email

На рисунке 5 можно убедиться, что указатель что настройка была произведена верно.

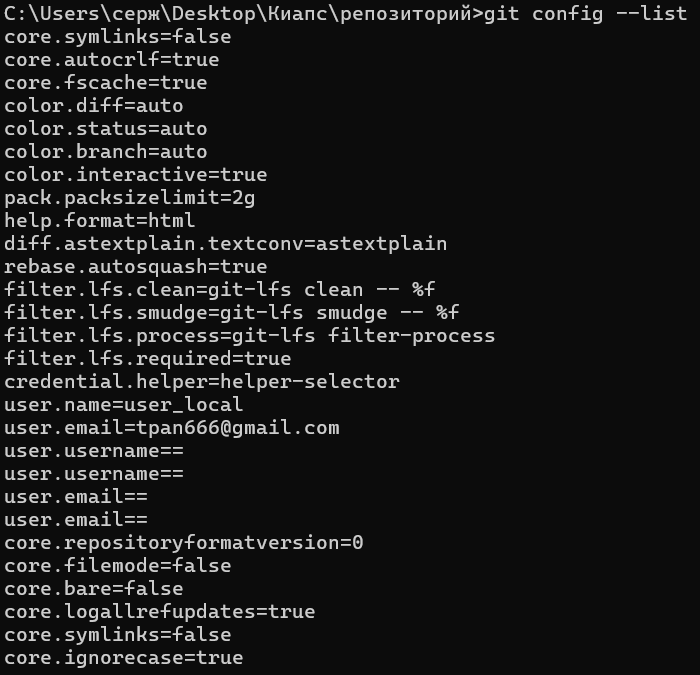


Рисунок 5 – Проверка корректности настройки

Получил сообщение о статусе с помощью команды «git status». На рисунке 6 представлен результат получения сообщения.

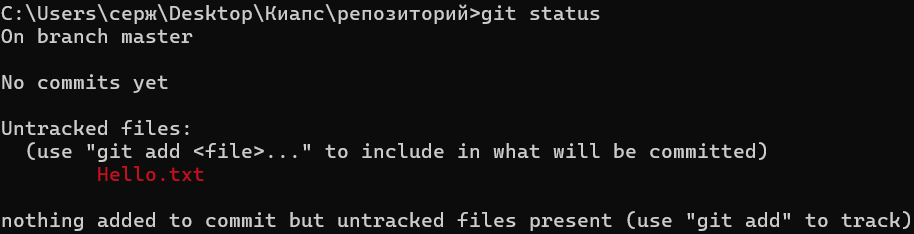


Рисунок 6 – Статус репозитория

Проиндексировал файл «Hello.txt». На рисунке 8 представлен вывод состояния после индексации файла «Hello.txt».

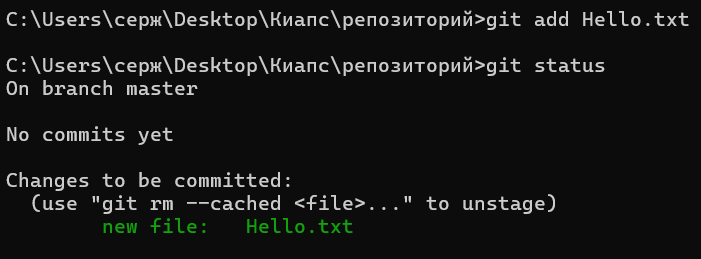


Рисунок 8 – Папка с объектами после отслеживания файлов

Создал ещё 3 файла текстового формата. На рисунке 9 представлен статус после создания ещё 3-х текстовых файлов.

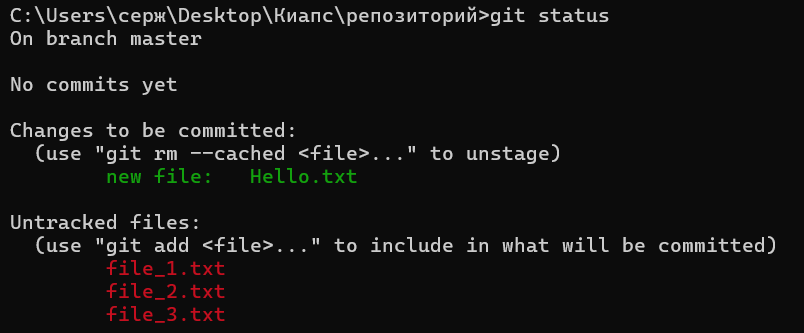


Рисунок 9 – Создание ещё 3-х файлов и их статус

Сделал коммит файла «Hello.txt». На рисунке 10 представлен результат коммита файла.

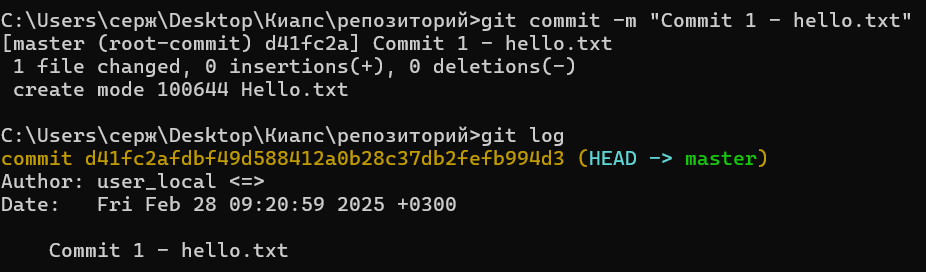


Рисунок 10 – Создание коммита

Коммит был произведён на ветке master. После создаю, перехожу и вывожу статус ветки. На рисунке 11 изображены выполнения данных действий.

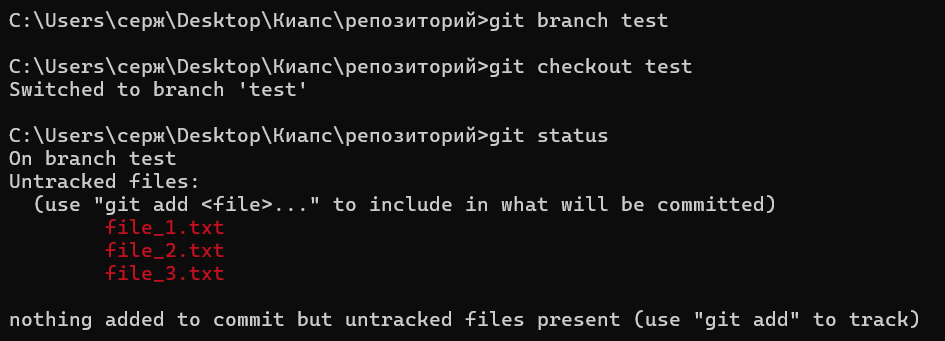


Рисунок 11 – Создание, переход и статус новой ветки

Сделал проверку файлов в папке проекта. На рисунке 12 представлена результат проверки.



Рисунок 12 – Проверка файлов проекта

Добавляю предыдущие файлы для коммита, произвожу коммит и вывожу лог. На рисунке 13 представлен результат выполнения.

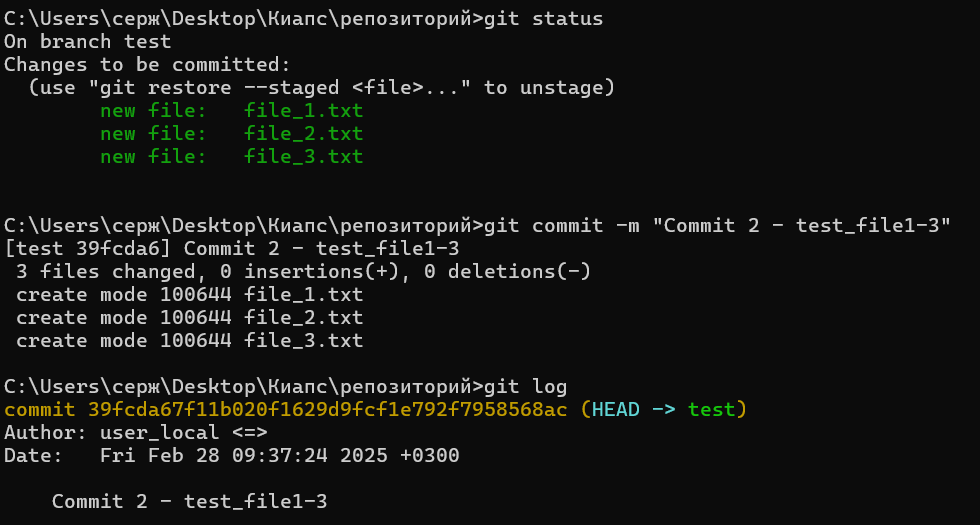


Рисунок 13 – Добавление, коммит и лог трёх тестовых файлов

После создаю новый файл «test.html», добавляю его и коммичу. На рисунке 14 представлен результат выполнения.

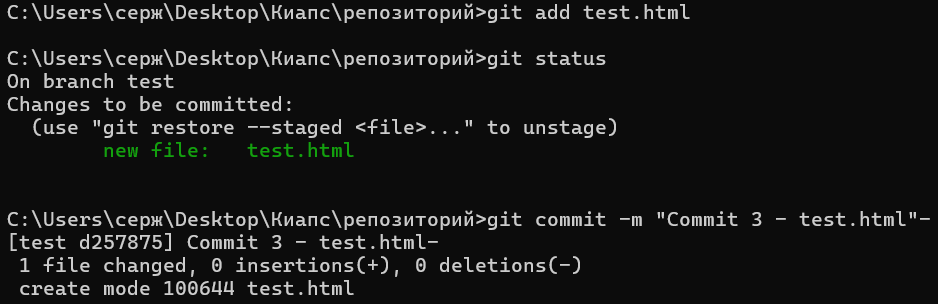


Рисунок 14 – Создание третьего коммита

Затем вывожу все ветки. На рисунке 15 представлен результат выполнения.

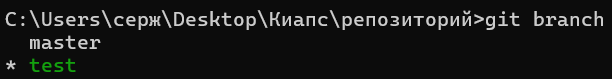
****

Рисунок 15 – Вывод всех веток

Перехожу на ветку «master» и провожу слияние веток. На рисунке 16 представлен результат выполнения.

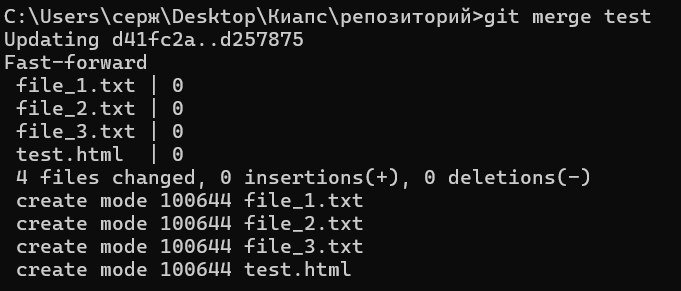


Рисунок 16 – Слияние веток

**Задание 3**

Перехожу на сайт «<https://github.com/>», создаю там аккаунт. На рисунке 17 представлен результат выполнения.

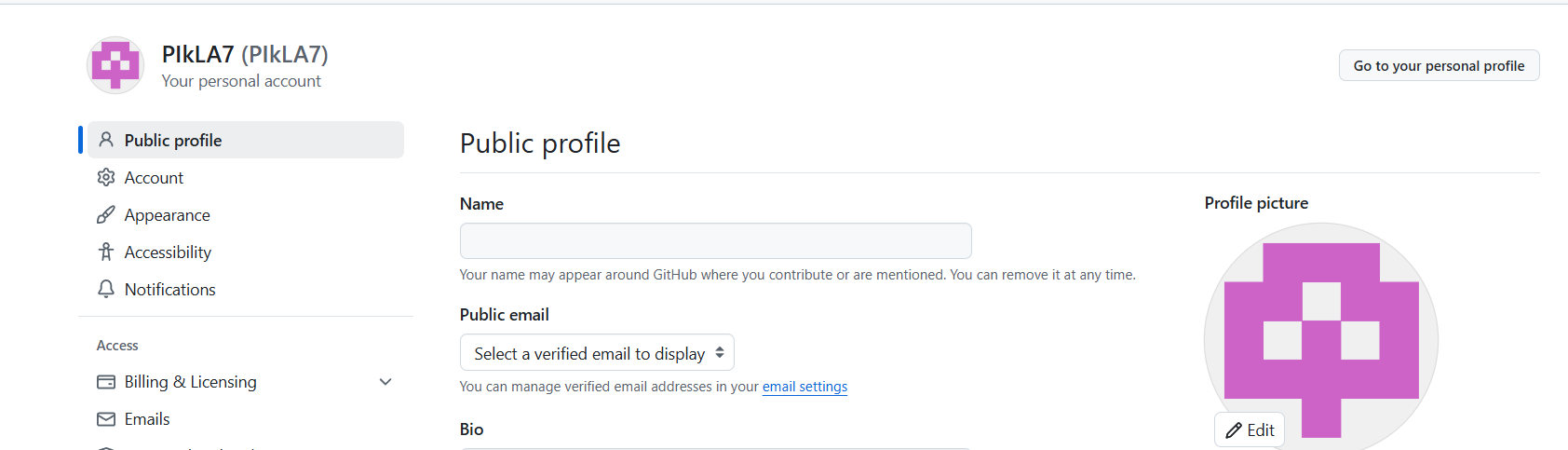


Рисунок 17 – Создание аккаунта

Далее создаю новый репозиторий с названием «OPI\_lab», добавляю в него Readme-файл и изменяю его согласно шаблону. На рисунке 18 представлен результат выполнения.

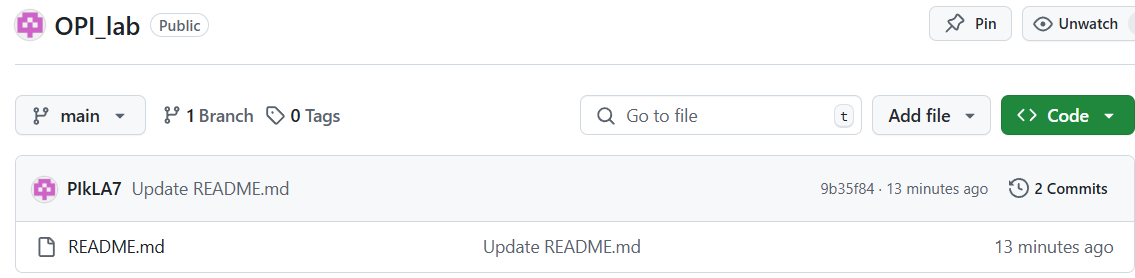


Рисунок 18 – Создание репозитория с Readme-файлом

Затем проверяю историю изменения файлов. На рисунке 19 представлен результат выполнения.

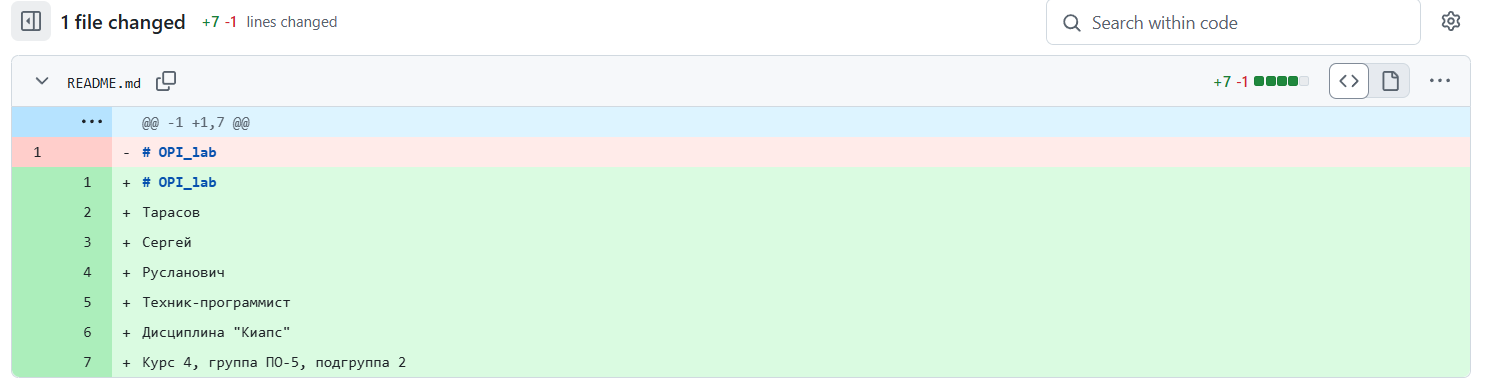


Рисунок 19 – История изменения файла

Провожу клонирование удалённого репозитория. На рисунке 20 представлен результат выполнения.

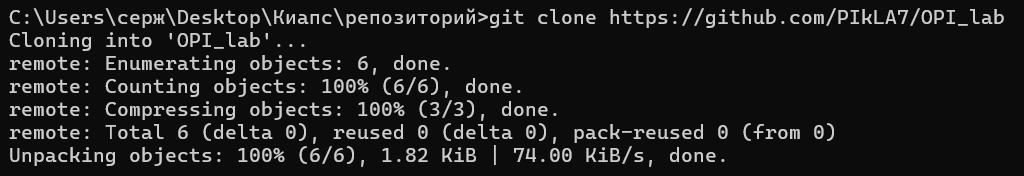


Рисунок 20 – Клонирование удалённого репозитория

Просматриваю дерево файлов внутреннего файла. На рисунке 21 представлен результат выполнения.

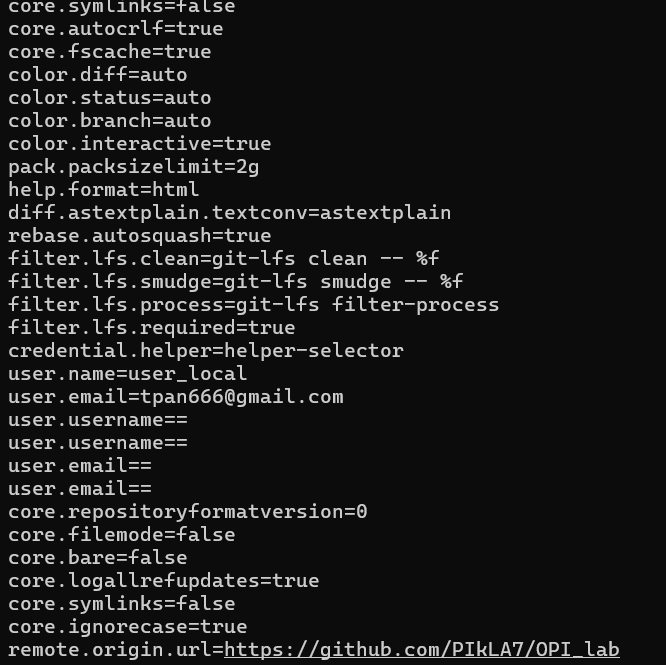


Рисунок 21 – Просмотр внутренних файлов

Вношу изменения в файл, фиксирую изменения. На рисунке 22 представлен результат выполнения.

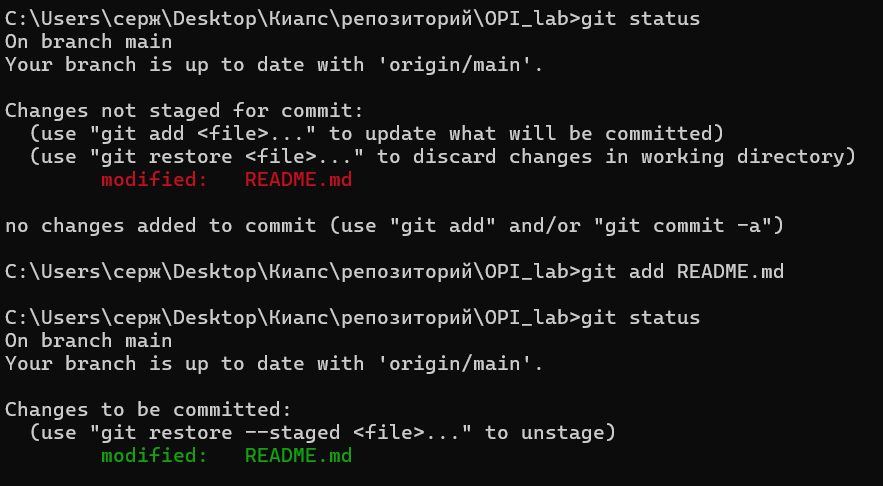


Рисунок 22 – Внесение изменений и фиксация изменений

Делаю коммит данного файла. На рисунке 23 представлен результат выполнения.

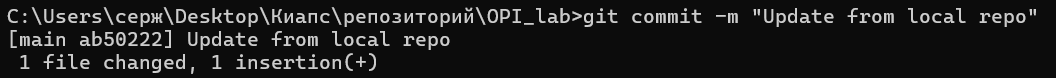


Рисунок 23 – Коммит файла

Просматриваю список настроенных удалённых репозиториев. На рисунке 24 представлен результат выполнения.



Рисунок 24 – Создание аккаунта

Затем произвожу отправку изменений в удалённый репозиторий. На рисунке 25 представлен результат выполнения.

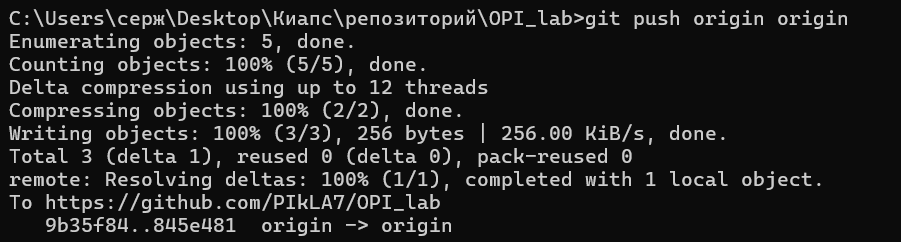


Рисунок 25 – Отправка изменений в удалённый репозиторий

После. На рисунке 17 представлен результат выполнения.

Рисунок 17 – Создание аккаунта

Вношу изменения . На рисунке 17 представлен результат выполнения.

Рисунок 17 – Создание аккаунта

**Ответы на контрольные вопросы**

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

Система контроля версий (VCS) — это инструмент для отслеживания изменений в исходном коде и других файлах проекта. Основные задачи VCS:

* Управление версиями файлов;
* Сохранение истории изменений;
* Совместная работа с несколькими людьми;
* Объединение изменений, сделанных разными пользователями;
* Возможность отката к предыдущим версиям файлов;
* Обеспечение безопасности и возможности восстановления данных.

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

* Хранилище — это место, где сохраняются все файлы проекта и их истории изменений. Может быть локальным или удалённым;
* Commit — операция, которая сохраняет изменения в хранилище с добавлением сообщения, описывающего эти изменения;
* История — последовательность всех изменений, сохранённых в хранилище с помощью commit-ов;
* Рабочая копия — это локальная версия проекта, с которой пользователь работает. Она может быть не синхронизирована с хранилищем.

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

* В централизованных VCS (CVCS) все изменения происходят в центральном репозитории, и пользователи подключаются к этому репозиторию для получения и отправки изменений. Примеры: Subversion (SVN), CVS.
* В децентрализованных VCS (DVCS) каждый разработчик имеет локальную копию всего репозитория, включая всю историю изменений. Пользователи могут работать автономно и синхронизировать изменения по мере необходимости. Примеры: Git, Mercurial.

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

* При единоличной работе с VCS, пользователь:
* Инициализирует репозиторий;
* Работает с рабочей копией;
* Вносит изменения в файлы;
* Сохраняет изменения с помощью commit;
* Осуществляет обновления и синхронизацию с хранилищем;
* Может откатить изменения, если нужно.

1. Опишите порядок работы с общим хранилищем VCS.

* При работе с общим хранилищем:
* Каждый разработчик клонирует репозиторий;
* Работает с локальной копией, внося изменения;
* Регулярно синхронизирует свою копию с центральным репозиторием, получая последние изменения от других разработчиков (pull);
* Отправляет свои изменения в репозиторий (push);
* Разрешает конфликты, если несколько пользователей изменяли одни и те же участки кода.

1. Краткая история Git.

Git был разработан в 2005 году Линусом Торвальдсом, создателем Linux, для управления исходным кодом ядра Linux. Он был создан как децентрализованная система контроля версий, которая могла бы эффективно работать с большими проектами и позволяла бы пользователям работать автономно.

1. Каковы основные задачи, решаемые инструментальным средством git?

* Управление версиями исходного кода;
* Ветвление и слияние изменений;
* Управление репозиториями;
* Сохранение и восстановление истории изменений;
* Совместная работа над проектами, синхронизация изменений.

1. Назовите и дайте краткую характеристику командам git.

* git init — инициализация нового репозитория;
* git clon — создание копии существующего репозитория;
* git add — добавление изменений в индекс (предкоммит);
* git commit — фиксация изменений в репозитории;
* git status — отображение текущего состояния рабочего каталога и индекса;
* git pull — получение изменений из удалённого репозитория и слияние с локальными;
* git push — отправка изменений в удалённый репозиторий;
* git branch — управление ветками;
* git merge — слияние веток;

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

Локальный репозиторий:

* git init — инициализация нового репозитория;
* git commit -m "Сообщение коммита" — сохранение изменений.

Удалённый репозиторий:

* git clone <url> — клонирование репозитория;
* git pull origin master — получение изменений с удалённого репозитория;
* git push origin master — отправка изменений в удалённый репозиторий.

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

Ветки (branches) — это способы создания параллельных линий разработки в Git. Ветки позволяют:

* Разрабатывать новые функции или исправления без воздействия на основную версию кода;
* Параллельно работать нескольким разработчикам;
* Изолировать экспериментальные изменения;
* Позволяют впоследствии сливать изменения обратно в основную ветку (например, master).

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

В Git можно игнорировать файлы, добавляя их в файл .gitignore. Используется, чтобы не добавлять временные файлы, логи или файлы, которые не должны попадать в репозиторий (например, файлы конфигурации, созданные средой разработки).

**Вывод:** На лабораторной работе научился использовать инструмент для контроля версий git.