Изображение выглядит как зарисовка, рисунок, символ, графическая вставка

Контент, сгенерированный ИИ, может содержать ошибки.

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

**РТУ МИРЭА**

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 1** **по дисциплине**

**«**Технологии разработки программных приложений**»**

**Тема: «Системы контроля версий»**

Выполнил студент группы ИНБО-30-23 Телендий В.А.

Принял преподаватель Исабекова О.А.

Практическая работа выполнена «\_\_»\_\_\_\_\_\_\_2025 г.

*(подпись студента)*

«Зачтено» «\_\_»\_\_\_\_\_\_\_2025 г.  *(подпись руководителя)*

**СОДЕРЖАНИЕ**

[ЧАСТЬ 1. ОСНОВНЫЕ КОМАНДЫ GIT 3](#_Toc192112131)

[ЧАСТЬ 2. СИСТЕМЫ УПРАВЛЕНИЯ РЕПОЗИТОРИЯМИ 9](#_Toc192112132)

[ЧАСТЬ 3. РАБОТА С ВЕТВЛЕНИЕМ И ОФОРМЛЕНИЕМ КОДА 15](#_Toc192112133)

[ОТВЕТЫ НА ВОПРОСЫ 20](#_Toc192112134)

[ВЫВОД 22](#_Toc192112135)

# ЧАСТЬ 1. ОСНОВНЫЕ КОМАНДЫ GIT

1. Установите и настройте клиент git на своей рабочей станции.



**Рисунок 1 – Результат установки Git**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 2 – Результат настройки Git**

1. Создайте локальный репозиторий и добавьте в него несколько файлов.

Изначально создаем пустую директорию под названием WorkDir, где и создаем локальный репозиторий (рис. 3), после чего последовательно добавляем несколько файлов (рис. 4-6).

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 3 – Создание локального репозитория**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 4 – Добавление файла 1**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 5 – Добавление файла 2**

1. Внесите изменения в один из файлов.

Внесём изменения в file1.cpp после выполнения команды git status отображается, что file1.cpp был изменён (рис. 9), значит git распознал изменения, но они не зафиксированы в репозитории.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 6 – Отображение изменения file1.cpp**

1. Проиндексируйте изменения и проверьте состояние.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 7 – Индексация изменения**

1. Сделайте коммит того, что было проиндексировано в репозиторий. Добавьте к коммиту комментарий.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 8 – Создание коммита**

1. Измените еще один файл. Добавьте это изменение в индекс git. Измените файл еще раз. Проверьте состояние и произведите коммит проиндексированного изменения. Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды git status. Сделайте коммит второго изменения.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 9 – Создание нескольких коммитов**

1. Просмотрите историю коммитов с помощью команды git log. Ознакомьтесь с параметрами команды и используйте некоторые из них для различного формата отображения истории коммитов.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 10 – История коммитов**



**Рисунок 11 - Форматированное отображение истории коммитов**

1. Верните рабочий каталог к одному из предыдущих состояний.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 12 – Возвращение к предыдущему состоянию**

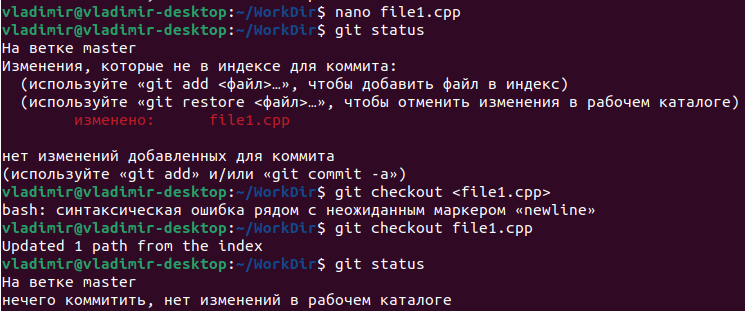
1. Изучите, как создавать теги для коммитов для использования в будущем.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 13 – Создание тега**

1. Отмените некоторые изменения в рабочем каталоге (до и после индексирования).



**Рисунок 14 – Отмена изменений до индексации**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 15 – Отмена изменений после индексации**

1. Отмените один из коммитов в локальном репозитории.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 16 – Отмена коммита**

# ЧАСТЬ 2. СИСТЕМЫ УПРАВЛЕНИЯ РЕПОЗИТОРИЯМИ

***Задача персонального варианта № 10:***

1. Клонируйте непустой удаленный репозиторий на локальную машину Создайте новую ветку и выведите список всех веток
2. Произведите 3 коммита в новой ветке в разные файлы
3. Выгрузите изменения в удаленный репозиторий
4. Произведите какие-нибудь изменения в файле который существует в ветке, но не коммитьте их
5. Внесите эти изменения в последний коммит (amend)
6. Выведите в консоли различия между веткой master и новой веткой
7. Слейте новую ветку с master при помощи merge
8. Создайте аккаунт на GitHub.

Изображение выглядит как снимок экрана, текст, Мультимедийное программное обеспечение, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 17 – Профиль GitHub**

1. Создайте репозиторий на GitHub.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 18 – Созданный репозиторий на GitHub**

1. Создайте несколько файлов на локальной машине при помощи консоли.

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

**Рисунок 19 – Создание локального репозитория**

1. Создайте SSH-ключ для авторизации.

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 20 – Создание SSH-ключа**

1. Свяжите репозиторий локальной машины с репозиторием на GitHub при помощи консоли.



**Рисунок 21 – Установление связи между локальным и удалённым репозиториями**

1. Создайте новую ветку в репозитории с помощью команды, произведите в ней какие-нибудь изменения, а после слейте с веткой master.

Сначала создаём ветку main , на ней в пустой файл file1.cpp добавляем код, затем переходим в ветку master, производим слияние и видим, что содержимое файлов на ветке master изменилось, соответственно слияние прошло успешно.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 22 - Фиксация изменений и слияние с веткой master**

1. Выполните цепочку действий в репозитории, согласно варианту

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 23 – Клонирование репозитория**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 24 – Создание ветки и вывод списка всех веток**

Изображение выглядит как текст, снимок экрана, Шрифт, меню

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 25 – Создание трёх коммитов в новой ветке в разные файлы**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 26 – Выгрузка изменений в удалённый репозиторий**

Изображение выглядит как текст, снимок экрана, Шрифт

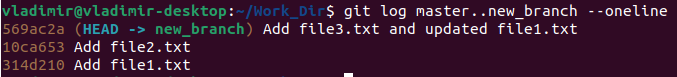
Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 27 – Создание изменений в файле в ветке без создания коммита**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 28 – Внесение изменений в последний коммит**



**Рисунок 29 – Вывод в консоль различий между веткой master и новой веткой**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 30 – Слияние новой ветки с master при помощи merge**

# ЧАСТЬ 3. РАБОТА С ВЕТВЛЕНИЕМ И ОФОРМЛЕНИЕМ КОДА

**Вариант № 1**

1. Сделайте форк репозиториев в соответствии с вашим вариантом.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 31 – Форк репозитория**

1. Склонируйте его на локальную машину

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 32 – Клонирование на локальную машину**

1. Создайте две ветки branch1 и branch2 от последнего коммита в master'е

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 33 – Создание веток**

1. Проведите по 3 коммита в каждую из веток, которые меняют один и тот же кусочек файла

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

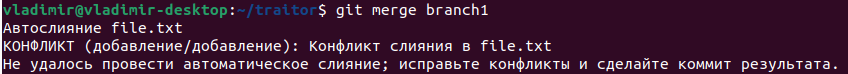
**Рисунок 34 – 3 коммита в ветке branch1**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 35 – 3 коммита в ветке branch2**

1. Выполните слияние ветки branch1 в ветку branch2, разрешив конфликты при этом



**Рисунок 36 – Слияние веток branch1 и branch2**

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 37 - Решение конфликта при слиянии веток branch1 и branch2**

1. Выгрузите все изменения во всех ветках в удаленный репозиторий

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 38 - Выгрузка изменений со всех веток на GitHub**

1. Проведите еще 3 коммита в ветку branch1

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 39 - 3 коммита в ветку branch1**

1. Склонируйте репозиторий еще раз в другую директорию

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 40 - Клонирование репозитория в другую директорию**

1. В новом клоне репозитория сделайте 3 коммита в ветку branch1

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 41 - 3 коммита в ветку branch1 в новом репозитории**

1. Выгрузите все изменения из нового репозитория в удаленный репозиторий

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 42 - Выгрузка всех изменений из нового репозитория на GitHub**

1. Вернитесь в старый клон с репозиторием, выгрузите изменения с опцией –force

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 43 –** push --force

1. Получите все изменения в новом репозитории

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

**Рисунок 44 - Измененный репозиторий на GitHub**

# ОТВЕТЫ НА ВОПРОСЫ

1. Что такое репозиторий Git?

Репозиторий Git – это хранилище для кода и истории изменений. Он позволяет отслеживать версии файлов, управлять изменениями и сотрудничать с другими разработчиками.

1. Что такое коммит?

Коммит — это зафиксированное изменение в репозитории. Каждый коммит представляет собой снимок состояния файлов на момент его создания и сопровождается сообщением, описывающим изменения.

1. Что такое ветка в репозитории Git?

Ветка в репозитории Git — это отдельная линия разработки. Она позволяет работать над различными задачами или функционалом параллельно, не мешая стабильной версии кода в основной ветке, обычно называемой main или master.

1. Что такое тег в репозитории Git?

Тег в репозитории Git — это фиксированная точка в истории коммитов, часто используемая для отметки релизов или важных версий. Теги могут быть аннотированными (с дополнительной информацией) или простыми (без метаданных).

1. Что такое слияние двух веток?

Слияние двух веток — это процесс объединения изменений из одной ветки в другую. Обычно слияние происходит между веткой разработки и основной веткой, чтобы интегрировать новую функциональность или исправления ошибок.

1. Что такое конфликт в Git? Как его решить и почему они бывают?

Конфликт в Git возникает, когда изменения в двух ветках касаются одних и тех же строк кода, и Git не может автоматически объединить эти изменения. Конфликты нужно решать вручную, выбирая нужные изменения, после чего нужно будет зафиксировать разрешение конфликта. Они могут возникать при одновременной работе нескольких разработчиков над одними и теми же файлами.

1. Как отменить слияние веток, если произошел конфликт?

Чтобы отменить слияние веток, если произошел конфликт, можно использовать команду: git merge --abort. Эта команда вернёт к состоянию до начала слияния.

1. Для чего нужен .gitignore?

Файл .gitignore нужен для указания Git, какие файлы или каталоги следует игнорировать и не добавлять в репозиторий. Это полезно для исключения временных файлов, настроек среды разработки или других несущественных данных из отслеживания.

# ВЫВОД

В ходе работы с Git были изучены различные доступные команды.

Git – это мощный инструмент контроля версий, который помогает разработчикам управлять кодовой базой и поддерживать её. Git позволяет отслеживать изменения, возвращаться к предыдущим версиям, сотрудничать с другими и многое другое. Он предоставляет эффективный и надежный способ управления кодом, а также понимание того, как его эффективно использовать, что может значительно улучшить рабочий процесс и производительность.