



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиПО)

КУРСОВАЯ РАБОТА

по дисциплине: Инструментальные средства разработки программного обеспечения с открытым исходным кодом
по профилю: Проектирование и разработка сред и приложений дополненной и виртуальной реальностей
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Локальная система контроля версий программного обеспечения

Студент: Маалуф Александр

Группа: ИКБО -34-22

Работа представлена к защите 09.12.23 (дата) Ма / Маалуф /
(подпись и ф.и.о. студента)

Руководитель: Беляев Павел Вячеславович, к.т.н., доцент кафедры

Работа допущена к защите 23.12.23 (дата) Беляев ПВ / Беляев ПВ /
(подпись и ф.и.о.рук-ля)

Оценка по итогам защиты: Отлично

Беляев ПВ к.т.н., доцент /
Кузнецов А.В. к.т.н. уч. /

(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2023 г.



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

**Институт информационных технологий (ИТ)
Кафедра игровой индустрии (ИИ)**

**ЗАДАНИЕ
на выполнение курсовой работы**

по дисциплине: Инструментальные средства разработки программного обеспечения с открытым исходным кодом

Студент: Маалуф Александр

Группа: ИКБО-34-22

Срок представления к защите: 08.12.2023

Руководитель: Беляев Павел Вячеславович, доцент

Тема: Локальная система контроля версий программного обеспечения

Исходные данные:используемые технологии: C++/Python(на выбор),библиотеки или модули для упрощения операций с файлами.(Файловая система), Интерфейс командной строки (CLI) , Алгоритм сравнения файлов (diff),нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18, ГОСТ 7.32-2017.

Перечень вопросов, подлежащих разработке, и обязательного графического материала:
1.Схема архитектуры системы. Создайте визуальное представление архитектуры системы, включая компоненты, модули и то, как они взаимодействуют 2. Диаграмма рабочего процесса: разработать диаграмму рабочего процесса, которая иллюстрирует, как пользователи будут взаимодействовать с системой, от инициализации до внесения изменений. 3. Макеты пользовательского интерфейса. Если система будет иметь графический пользовательский интерфейс, создайте макеты или каркасы для визуализации дизайна пользовательского интерфейса и взаимодействия с пользователем. 4. Схема потока данных. Опишите поток данных в системе, показывая, как файлы отслеживаются, хранятся и извлекаются.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Директор ИИТ: А. С. Зуев /А. С. Зуев/, «15» сентября 2023 г.

Задание на КР выдал: П.В.Беляев /П.В.Беляев/, «15» сентября 2023 г.

Задание на КР получил: Маалуф Александр /М.Александр/, «15» сентября 2023 г.

Директору
Института информационных технологий (ИТ)

Зуеву Андрею Сергеевичу

От студента Маалуф Александр

ФИО

ИКБО-34-22

группа

2 курс

курс

Контакт: alexandermaaluf.pro@gmail.com,
+7 (916) 964-60-46

Заявление

Прошу утвердить мне тему курсовой работы по дисциплине «Инструментальные средства разработки программного обеспечения с открытым исходным кодом» образовательной программы бакалавриата 09.03.04 (Программная инженерия) «Локальная система контроля версий программного обеспечения»

Приложение: лист задания КР/КП на двустороннем листе (проект)

Подпись студента	<u>Маалуф</u>	/Маалуф Александр
	подпись	ФИО
Дата	15.09.2023	
Подпись руководителя	<u>Беляев</u>	Доцент
	подпись	Беляев П.В.
		Должность, ФИО
Дата	15.09.2023	

Сопровождено

УДК 004.4

Маалуф Разработка программы для локальная система контроля версий программного обеспечения. / Курсовая работа по дисциплине «Инструментальные средства разработки программного обеспечения с открытым исходным кодом» направления профессиональной подготовки бакалавриата 09.03.04 «Программная инженерия» (3ый семестр) / руководитель доцент Беляев П.В. / кафедра ИППО Института ИТ МИРЭА – с. 41, илл. 28, ист. 20.

Цель работы – разработка локальной системы контроля версий.

В рамках работы осуществлены анализ предметной области, проведен синтез Программного Обеспечения и приведена инструкция к эксплуатации.

Maaluf Development of a program for a local software version control system / Course work on the discipline "Open Source Software Development Tools" of the direction of professional training of Bachelor's degree 09.03.04 "Software Engineering" (3rd semester) / Head Associate Professor Belyaev P.V. / Department of IPPO Institute of IT MIREA - p. 41, fig. 28, rf. 20.

The purpose of the work is to develop a program for a local software version control system

The work includes domain analysis, technical specifications and system architecture

М. МИРЭА. Ин-т ИТ. Каф. ИиППО. 2023 г. Маалуф

АННОТАЦИЯ

В рамках дисциплины «Средства разработки открытого программного обеспечения» в соответствии с заданием курсовой работы необходимо разработать локальную систему контроля версий.

Объём пояснительной записки к курсовой работе составляет 41 страницы. В курсовой работе содержится 28 иллюстрации, 20 источников.

Введение и анализ предметной области главы знакомят с обсуждаемой темой. Цель работы – разработка локальной системы контроля версий. В поставленные задачи входит анализ предметной области, «Эволюция инструментов контроля версий», разница между двумя типами централизованных систем контроля версий и почему мы используем такой инструмент.

В главе «Анализ» предметной области также определены объект и предмет исследования, особое внимание уделено роли ЛГУВПО в современном развитии. Затем мы переходим к анализу предметной области, раскрывая преимущества и уникальные особенности LSUVPO, а затем завершаем тенденциями использования LSUVPO.

В главе «Синтез программного обеспечения» описаны функциональные требования, такие как «Основные функции», «Дополнительные функции». И определение алгоритма под названием Diff, используемого в локальной версии контроля версий. Подкрепленные скриншотами.

В Главе «Архитектура системы » знакомит с архитектурой системы, уделяя особое внимание простоте, эффективности и интерфейсу командной строки для локального управления версиями. Ключевые компоненты включают управление репозиториями, контроль репозитория и истории проектов, а также ядро контроля версий, обеспечивающее надежное отслеживание изменений. Взаимодействие с пользователем облегчается через пользовательский интерфейс на основе консоли, что способствует

повышению эффективности работы. Философия системы отдает приоритет простой структуре, эффективным операциям контроля версий и адаптируемости к разнообразным потребностям проекта без ненужной сложности. Последующие разделы более подробно рассматривают взаимодействие компонентов, начиная с рассмотрения в разделе 3.2 роли и взаимодействия каждого ключевого компонента внутри системы.

В заключение говорится о том, что Локальные системы управления версиями программного обеспечения (ЛСУВПО) играют ключевую роль в разработке, обеспечивая эффективное управление версиями, гибкость и безопасность кодовой базы. Прогнозируется углубление интеграции с облачными сервисами и рост популярности распределенных систем для улучшения доступа и работы в оффлайн-режиме.

СОДЕРЖАНИЕ

Введение.....	7
Глава 1. Анализ предметной области.....	8
Глава 2. Техническое задание.....	14
Глава 3. Архитектура системы	21
Заключение.....	24
Приложения.....	26
Список литературы	39

Введение

В динамичном мире разработки программного обеспечения, где точность и эффективность стоят на первом плане, важно обратить внимание на средства, которые мы имеем в распоряжении. Часто пренебрегаемым, но неотъемлемым героем в этом множестве инструментов является Локальная Система Управления Версиями Программного Обеспечения (ЛСУВПО). В этой презентации мы углубимся в тонкости ЛСУВПО, раскрывая не только ее значение, но и уникальные преимущества, которые она предоставляет разработчикам. Сегодня, когда динамизм в разработке программного обеспечения неотъемлем, и точность играет ключевую роль, автоматизированное отслеживание изменений в репозиториях становится настоящей необходимостью. Это не только обеспечивает безопасность, но и повышает операционную эффективность. Поэтому, рассматривая наше решение, мы предлагаем взглянуть на ЛСУВПО как на эффективный инструмент в этом динамичном контексте.

В этом проекте моя цель — более глубоко понять Локальную Систему Управления Версиями Программного Обеспечения (ЛСУВПО) и продемонстрировать работоспособный прототип, освещая ключевые операции.

Глава 1. Анализ предметной области

В динамике современной разработки программного обеспечения, акцентирующей внимание на точности и эффективности, обоснование выбора Локальной Системы Управления Версиями Программного Обеспечения (ЛСУВПО) становится неотъемлемым элементом. Наш анализ предметной области включает в себя следующие ключевые аспекты:

1. Эволюция инструментов управления версиями:

Давайте проследим за эволюцией инструментов управления версиями, акцентируя внимание на различных этапах развития систем контроля версий. Рассмотрим, как Локальная Система Управления Версиями Программного Обеспечения (ЛСУВПО), в контексте их развития, становится стратегическим выбором для поддержания структурированной и безопасной истории разработки.

Примеры централизованных систем контроля версий:

А- Централизованный

В- Децентрализованный

А. Примеры централизованных систем контроля версий

Наиболее распространенными централизованными системами контроля версий являются система параллельных версий (CVS), Perforce и Subversion (SVN). Существует также Microsoft Team Foundation Server (TFS), который теперь известен как Azure DevOps Server.

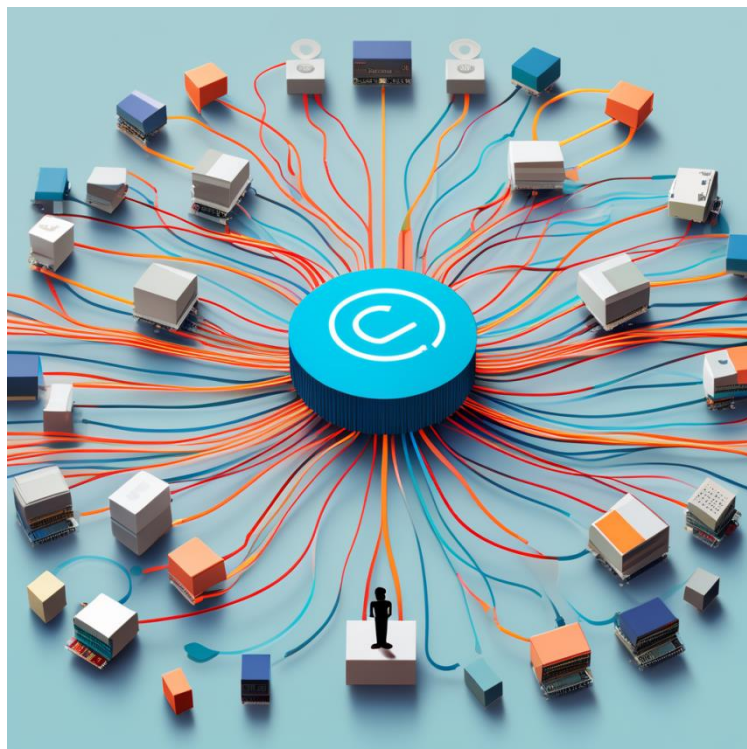


Рисунок 1 -централизованная система

Каковы преимущества централизованной системы контроля версий?

Эффективная работа с двоичными файлами:

Обеспечивает оптимальное хранение графических и текстовых файлов.

Позволяет команде экономить место, используя централизованный сервер.

Полная прозрачность:

Централизованное расположение дает каждому члену команды полную информацию о текущей работе над проектом.

Обеспечивает прозрачность состояния проекта для коллективной работы.

Удобное обучение и использование:

Просто понять и использовать, что ускоряет внесение изменений и облегчает совместную работу.

Минимизирует время обучения и упрощает рабочий процесс для разработчиков любого уровня опыта.

Каковы недостатки централизованной системы контроля версий?

Одна Точка Отказа:

Централизованные серверы могут привести к потере данных при сбое.

Отсутствие доступа в оффлайне во время сбоев может привести к остановке разработки и потере кода.

Команды рискуют потерей своего исходного кода в случае сбоя сервера.

Замедленная Скорость Задерживает Разработку:

Пользователи централизованных систем контроля версий часто сталкиваются с трудностями при быстром создании веток.

Задержки в ветвлении делают процесс разработки более затянутым, особенно при медленных сетевых подключениях.

Скорость работы напрямую влияет на сроки поставки функций и достижение ценности для бизнеса.

Мало Стабильных Моментов для Внесения Изменений:

При коллаборации крупных команд сложно найти стабильный момент для внесения изменений.

Неустойчивые изменения не могут быть отправлены в основной репозиторий, что задерживает проект и вызывает конфликты слияния.

Эти ограничения централизованных систем могут привести к задержкам в разработке и нестабильности процесса внесения изменений, что часто стимулирует команды переходить на другие системы контроля версий, такие как Git.

В. Примеры децентрализованных систем контроля версий:

Децентрализованные системы контроля версий, такие как Git и Mercurial, предоставляют разработчикам более гибкий подход к управлению версиями. В отличие от централизованных систем, они позволяют каждому участнику иметь полную копию репозитория, что способствует более эффективной работе в условиях отсутствия постоянного подключения к сети.

Преимущества децентрализованной системы контроля версий (например, Git и Mercurial):

Гибкость и надежность:

Работа в оффлайне позволяет разработчикам продолжать работу и вносить изменения, даже если нет подключения к центральному серверу.

Устойчивость к отказам сервера, так как каждый участник имеет полную историю версий.

Эффективное ветвление и слияние:

Быстрое и эффективное ветвление и слияние вносит гибкость в процесс разработки.

Легкость создания, комбинирования и управления ветками повышает эффективность и ускоряет процессы.

Локальные операции:

Многие операции выполняются локально, что ускоряет процессы и позволяет разработчикам работать независимо от сетевого соединения.

Высокая степень гибкости:

Множество инструментов и функций обеспечивают разработчикам эффективное управление версиями и совместную работу.

Теперь рассмотрим недостатки распределенных систем контроля версий:

Нестандартные для новичков:

Для новых пользователей распределенные системы могут казаться сложными из-за большего числа концепций и команд.

Сложные сценарии слияния:

В некоторых случаях сложные сценарии слияния могут возникнуть из-за параллельных изменений в разных ветках, требуя более тщательного внимания при решении конфликтов.

Неудобство для крупных проектов:

В крупных проектах, особенно при наличии большого объема данных, распределенные системы могут столкнуться с проблемой производительности.

Таким образом, хотя децентрализованные системы контроля версий, такие как Git, предоставляют множество преимуществ, необходимо также учитывать их особенности и возможные сложности в использовании, особенно для менее опытных пользователей и в специфичных сценариях.

2. Роль ЛСУВПО в современной разработке:

В современной разработке программного обеспечения Локальные Системы Управления Версиями Программного Обеспечения (ЛСУВПО) играют ключевую роль, обеспечивая эффективное управление версиями и обеспечивая точность в процессах разработки. Вот несколько аспектов, которые подчеркивают их значимость:

Гибкость и Удобство Использования:

ЛСУВПО предоставляют разработчикам возможность эффективно управлять версиями своего кода, сохраняя изменения и создавая ветви для экспериментов.

Интерфейсы ЛСУВПО часто интуитивны и легко осваиваются, что ускоряет процесс обучения и повышает продуктивность.

Локальная Работа:

Основное преимущество ЛСУВПО заключается в возможности работать локально. Разработчики могут вносить изменения в код, даже находясь в оффлайн-режиме, что обеспечивает непрерывность работы независимо от сетевых условий.

Эффективное Ветвление и Слияние:

ЛСУВПО обеспечивают легкость в создании веток и их последующем слиянии. Это особенно важно при параллельной разработке различных функциональностей или при решении нескольких задач одновременно.

Устойчивость к Сбоям:

В случае сбоев центрального сервера, ЛСУВПО позволяют сохранять историю версий локально. Это предотвращает потерю данных и обеспечивает надежность в управлении версиями.

Интеграция с Другими Инструментами:

ЛСУВПО хорошо интегрируются с другими современными инструментами разработки, такими как интегрированные среды разработки (IDE), системы непрерывной интеграции и тестирования.

Безопасность и Контроль Доступа:

Многие ЛСУВПО предоставляют механизмы контроля доступа и безопасности, обеспечивая, что только авторизованные разработчики могут вносить изменения в определенные ветки или репозитории.

Таким образом, ЛСУВПО становятся неотъемлемым инструментом в современной разработке, обеспечивая командам разработчиков эффективное управление версиями, гибкость и высокую точность в управлении кодовой базой.

3. Преимущества и уникальные черты ЛСУВПО:

Локальные Системы Управления Версиями Программного Обеспечения (ЛСУВПО) предоставляют целый ряд преимуществ и уникальных черт, которые значительно облегчают процессы разработки и обеспечивают безопасность кодовой базы. Рассмотрим ключевые особенности:

История Изменений:

ЛСУВПО поддерживают детальную историю изменений, позволяя разработчикам отслеживать каждое изменение в коде. Это не только полезно для понимания эволюции проекта, но и помогает быстро выявлять и исправлять ошибки.

Откат к Предыдущим Версиям:

Разработчики могут легко откатываться к предыдущим версиям кода в случае обнаружения проблемы. Это обеспечивает безопасность и быстрое восстановление функциональности.

Эффективное Слияние Кодовой Базы:

ЛСУВПО обеспечивают эффективные механизмы слияния изменений, даже если несколько разработчиков работают над одним проектом. Это снижает вероятность конфликтов и облегчает совместную работу.

Локальное Хранение Истории:

Каждый разработчик хранит историю версий локально, что делает процесс работы с историей более быстрым и надежным. В случае сбоя сервера, разработчики могут продолжить работу без потери данных.

Улучшенная Безопасность:

ЛСУВПО предоставляют механизмы контроля доступа и шифрования, обеспечивая безопасность кодовой базы. Разработчики могут управлять правами доступа и предотвращать несанкционированные изменения.

Работа в Ветках:

Возможность создания локальных веток позволяет разработчикам экспериментировать и вносить изменения, не затрагивая основной код. Это

способствует более структурированной и безопасной разработке.

Удобство Резервного Копирования:

Локальное хранение версий облегчает создание резервных копий проекта. Разработчики могут легко архивировать свою работу, предотвращая потерю данных.

Улучшенная Работа в Команде:

ЛСУВПО создают более удобные условия для совместной работы. Разработчики могут легко обмениваться изменениями, не завися от постоянного соединения с центральным сервером.

Таким образом, уникальные черты ЛСУВПО существенно облегчают управление версиями, повышают производительность разработчиков и обеспечивают безопасность в процессе разработки программного обеспечения.

4. Тенденции в использовании ЛСУВПО:

Современная динамика разработки программного обеспечения подвергается влиянию различных тенденций в использовании Локальных Систем Управления Версиями Программного Обеспечения (ЛСУВПО). Рассмотрим ключевые направления этих тенденций:

1. Интеграция с Облачными Сервисами
2. Расширенная Поддержка CI/CD
3. Мультипаравлельная Работа
4. Улучшенная Визуализация и Аналитика:
5. Поддержка DevOps-практик
6. Расширенные Возможности Ветвления и Слияния
7. Увеличение Популярности Распределенных Систем
8. Автоматизированный Мониторинг и Управление Историей

Глава 2. Техническое задание

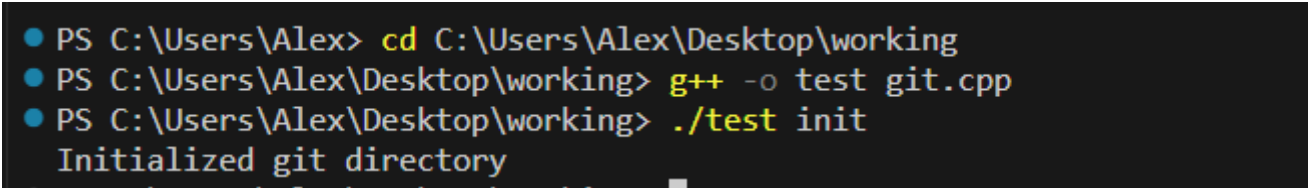
2.1 Функциональные требования:

2.1.1 Основные функции:

В этом разделе опишите основные функции, которые ваша Локальная Система Управления Версиями Программного Обеспечения (ЛСУВПО) предоставляет.

Создание репозитория: Одной из ключевых функций Локальной Системы Управления Версиями Программного Обеспечения (ЛСУВПО) является возможность пользователей создавать новые репозитории для отслеживания своих проектов.

Инициализация нового репозитория:



```
● PS C:\Users\Alex> cd C:\Users\Alex\Desktop\working
● PS C:\Users\Alex\Desktop\working> g++ -o test git.cpp
● PS C:\Users\Alex\Desktop\working> ./test init
  Initialized git directory
```

Рисунок 2 -Инициализация нового репозитория

Фиксация изменений: Детализируйте процесс записи изменений в репозиторий, включая добавление, модификацию и удаление файлов.

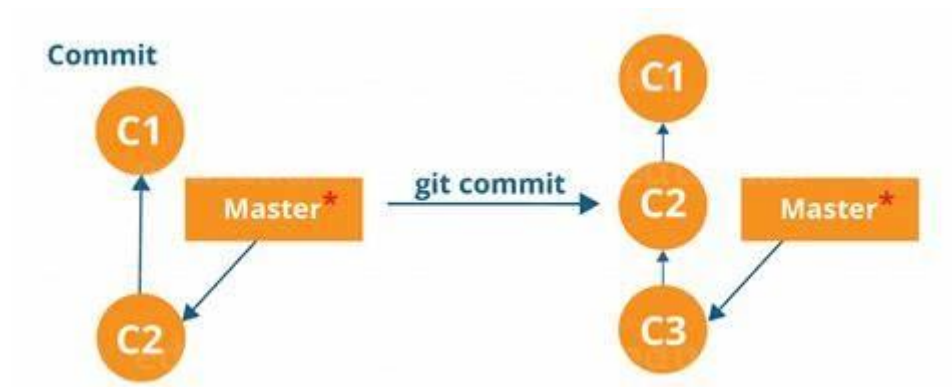


Рисунок 3 -Фиксация изменений

Создание и объединение ветвей: Разверните, как разработчики могут создавать ветви для экспериментальных функций или исправлений ошибок, а затем объединять их обратно в основную ветвь.

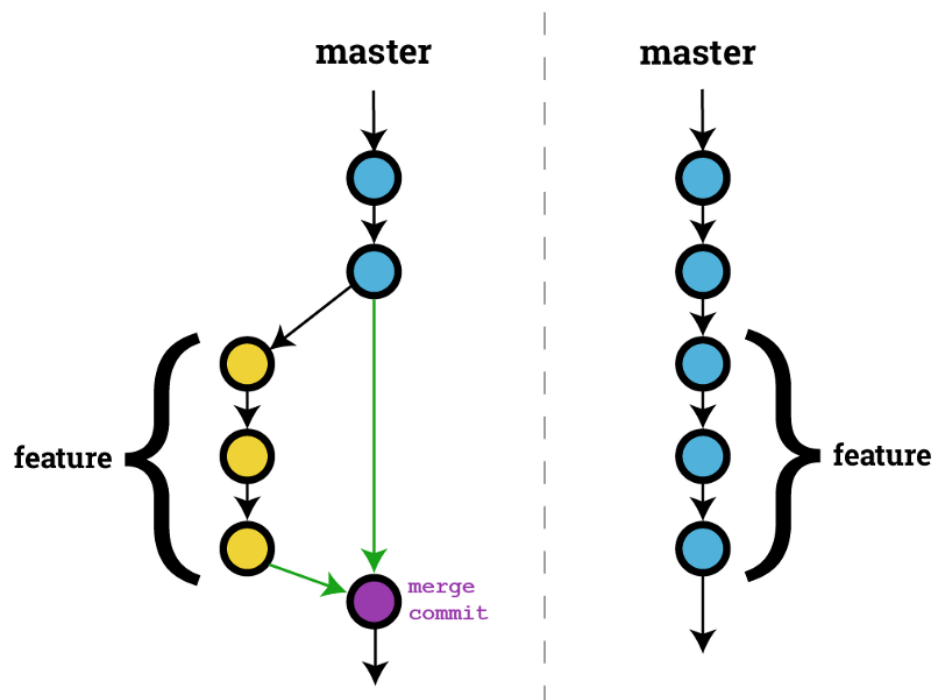


Рисунок 4 -Инициализация нового репозитория

Отслеживание истории: Уточните, как система поддерживает полную

историю изменений, позволяя пользователям просматривать и возвращаться к предыдущим состояниям.

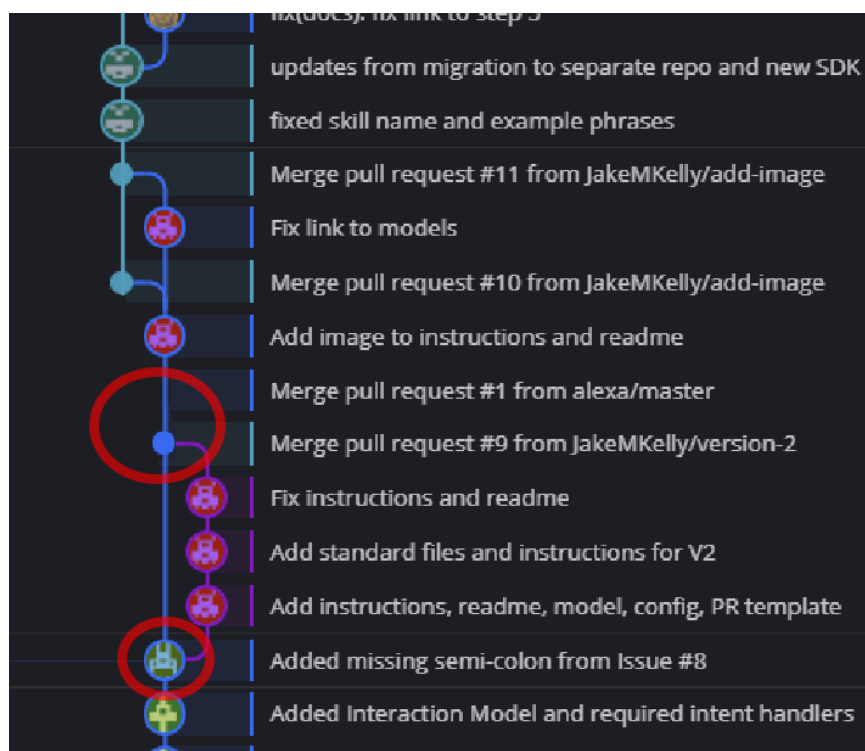


Рисунок 5 -Отслеживание истории

Механизм совместной работы: Если применимо, опишите, как несколько разработчиков могут сотрудничать над одним проектом, решая вопросы, такие как разрешение конфликтов.

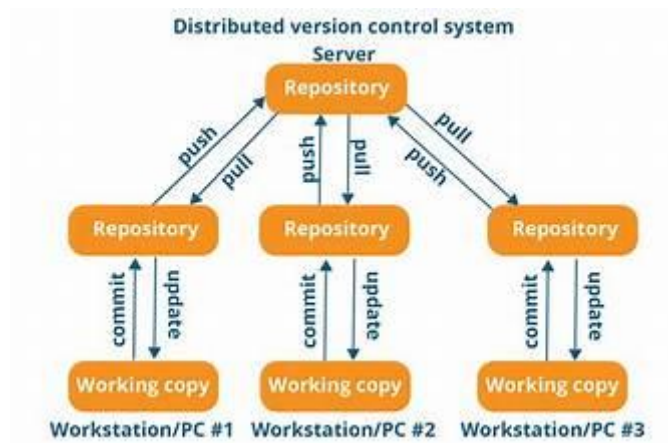


Рисунок 6 -Механизм совместной работы

Маркировка и версионирование: Обсудите механизмы для маркировки конкретных точек в истории или создания версий для улучшения управления проектом.

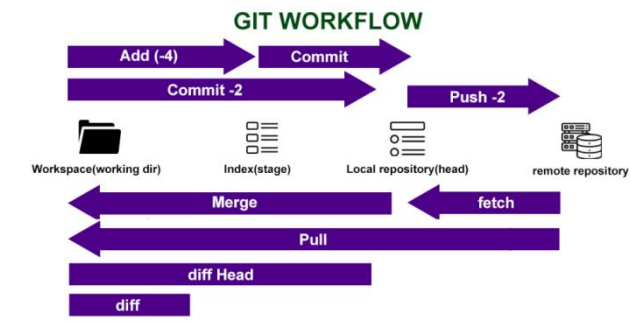


Рисунок 7 -Маркировка и версионирование

2.1.2 Расширенные функции:

Давайте рассмотрим краткие описания некоторых продвинутых функций в Git:

Git Hooks (Крючки Git):

Крючки Git - это скрипты, которые запускаются на определенных этапах

рабочего процесса Git. Они позволяют настраивать и автоматизировать процессы, такие как запуск тестов перед коммитом или отправка уведомлений после пуша.



Рисунок 8 -Крючки Git

Git Submodules (Подмодули Git):

Подмодули позволяют включать другие репозитории Git внутри своего. Это полезно для управления зависимостями, и каждый подмодуль может быть обновлен независимо.

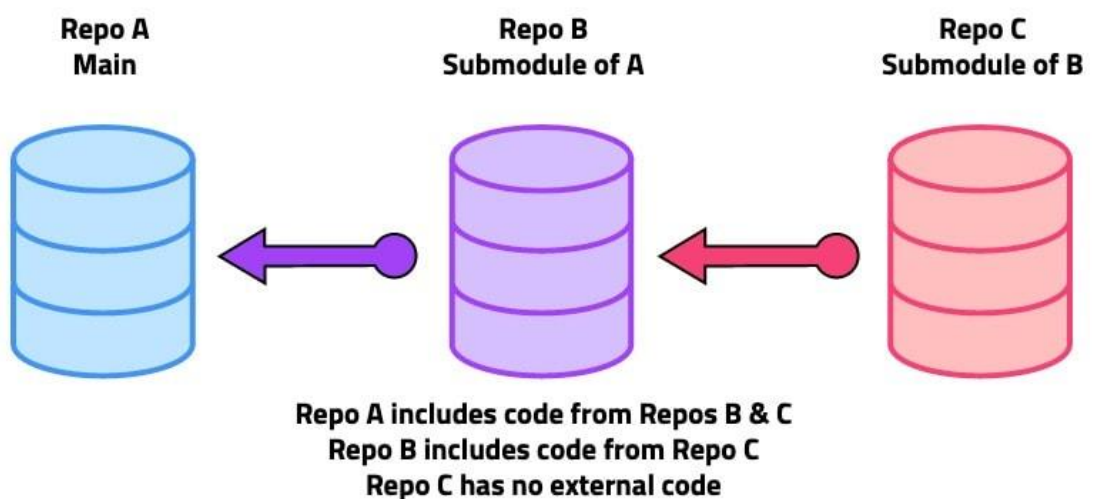


Рисунок 9 -Подмодули

Git LFS (Large File Storage) (Большие файлы в Git):

файлы, храня их за пределами основного репозитория, предотвращая его раздутие. Это критично для проектов с большими бинарными файлами, такими как изображения или видео.

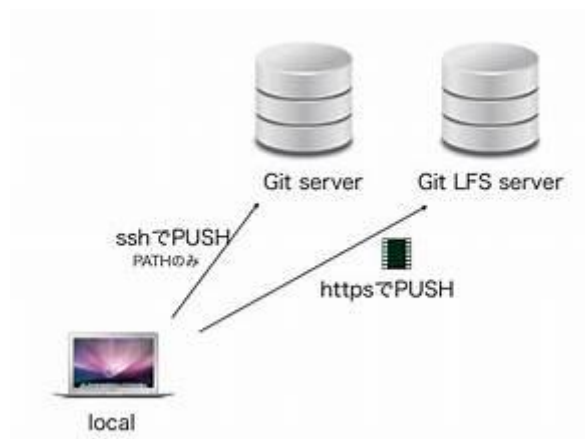


Рисунок 10 -Большие файлы в Git

Git Worktrees (Рабочие деревья Git):

Рабочие деревья Git позволяют иметь несколько рабочих каталогов для одного и того же репозитория. Это полезно для одновременной работы с разными ветками без постоянного переключения.

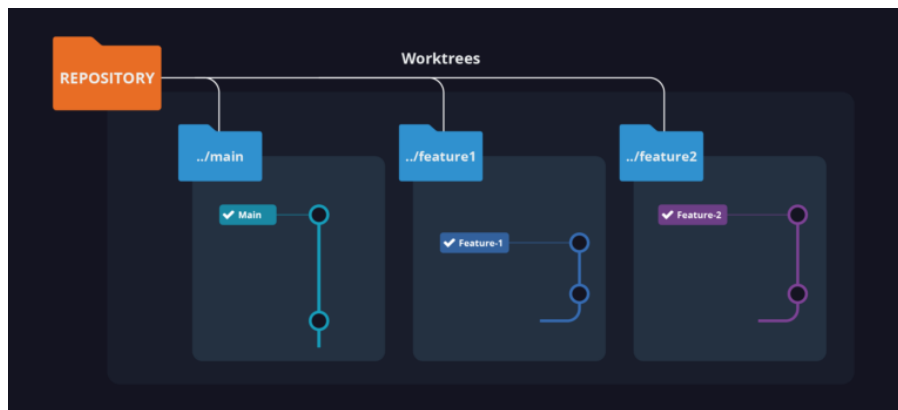


Рисунок 11 -Рабочие деревья Git

Git Bisect (Бинарный поиск Git):

Git Bisect помогает выявить коммит, который внес баг, выполняя бинарный поиск по истории коммитов, эффективно сужая круг подозреваемых коммитов.

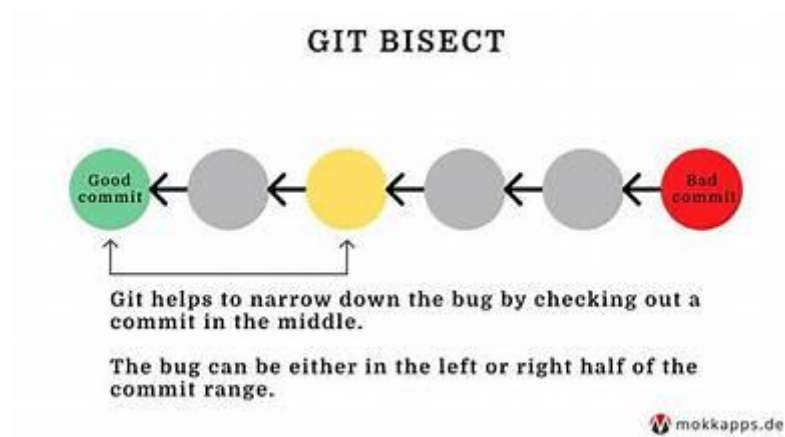


Рисунок 12 -Бинарный поиск Git

Git Stash (Спрятать изменения Git):

Git Stash позволяет временно сохранить изменения, которые еще не готовы к коммиту. Это полезно, когда нужно переключить ветки или выполнить другие

операции без фиксации неполного кода.

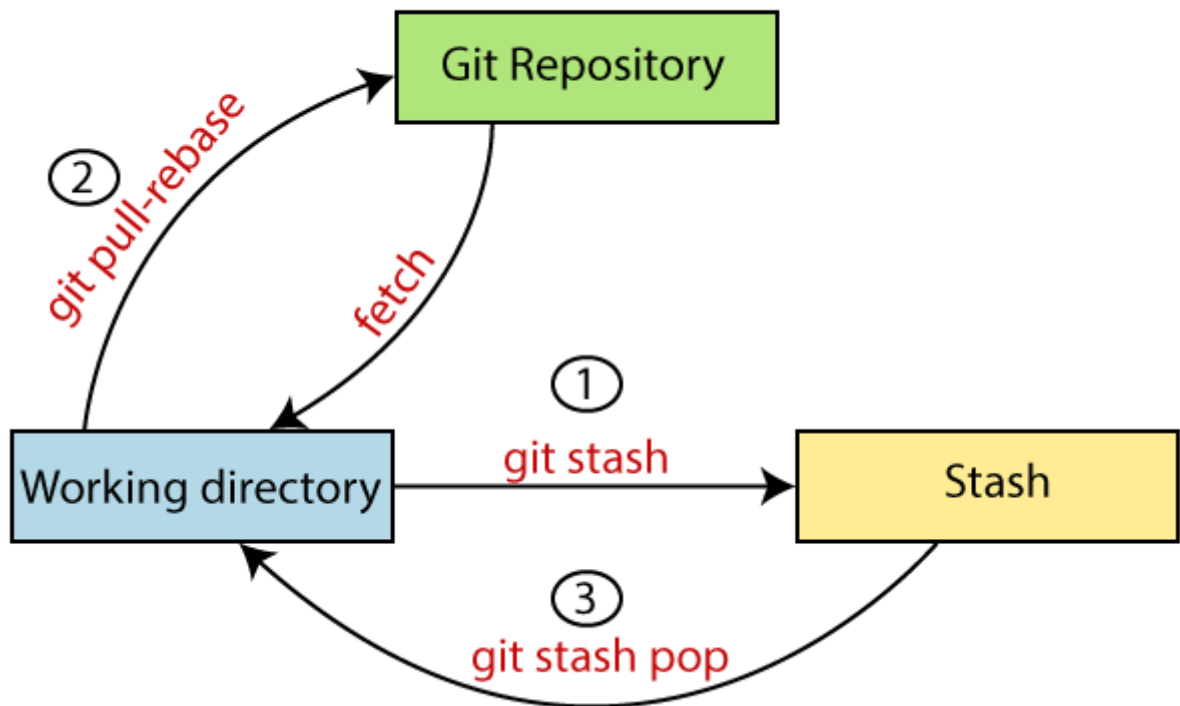


Рисунок 13 -Спрятать изменения Git

Алгоритм сравнения (Diff):

это метод анализа двух текстовых файлов или строк с целью выявления различий между ними. Результатом работы такого алгоритма является набор изменений, позволяющих преобразовать один текст в другой.

Применение:

Системы контроля версий: Применяется для эффективного отслеживания изменений в исходном коде и текстовых документах, позволяя пользователям видеть, какие строки были добавлены, изменены или удалены между версиями.

Редакторы текста: Используется для предоставления пользователям наглядного представления изменений в документах, улучшая процесс совместной работы.

Автоматизированные системы слияния: Применяется при автоматическом объединении изменений от нескольких источников для создания общего результата.

Зачем используется:

Алгоритм сравнения не только обеспечивает прозрачность изменений между версиями, но также улучшает эффективность управления версиями, сокращая объем хранимых данных и упрощая процессы слияния и отката изменений.

Глава 3. Архитектура системы

3.1 Обзор архитектуры Введение: В данном разделе мы предоставим обзор высокого уровня архитектуры системы, обрисовав основные компоненты и их взаимодействие. Это позволит понять, как различные элементы взаимодействуют для обеспечения эффективного локального управления версиями.

программное обеспечение для локального управления версиями (LSVCS), создана для оптимизации процесса управления версиями в проектах по разработке программного обеспечения. В центре архитектуры лежат принципы простоты, эффективности и командного интерфейса для обеспечения удобства использования.

Ключевые компоненты:

Для понимания архитектуры давайте рассмотрим основные компоненты:

Управление репозиторием:

Этот компонент отвечает за управление репозиториями, организацию истории проекта и управление различными версиями кодовой базы.

Ядро управления версиями:

В центре системы находится ядро управления версиями, которое обеспечивает основные функции, такие как отслеживание изменений, создание веток и их объединение. Это гарантирует надежную историю разработки проекта.

Пользовательский интерфейс (консоль):

Пользователь взаимодействует с системой через интерфейс на основе командной строки. Такой подход соответствует фокусу системы на эффективности и предоставляет разработчикам простой опыт.

Философия системы:

Архитектура создана с учетом следующих принципов:

Стремление к непритязательной структуре для улучшения понимания и взаимодействия пользователей.

Эффективность: Приоритет быстрых и эффективных операций управления версиями.

Гибкость: Адаптация к различным потребностям проектов без избыточной сложности.

В следующих разделах мы более подробно рассмотрим взаимодействие между компонентами, потоками данных и углубимся в конкретные функции, которые делают локальное программное обеспечение для контроля версий уникальным. Давайте перейдем к разделу 3.2, где мы более подробно рассмотрим каждый основной компонент.

3.2 Основные компоненты

В этом разделе мы подробно рассмотрим конкретные основные компоненты архитектуры системы. Понимание роли и взаимодействия каждого компонента является ключевым для полного понимания программного обеспечения для локального управления версиями.

Ключевые компоненты:

Управление репозиторием:

Этот компонент отвечает за организацию репозиториев, ведение истории проекта и управление различными версиями кодовой базы. Он служит центральным узлом для операций управления версиями.

Ядро управления версиями:

В центре системы находится ядро управления версиями, предоставляющее основные функции. К ним относятся отслеживание изменений, создание веток и объединение изменений кода. Надежная функциональность обеспечивает стабильную и прослеживаемую историю разработки проекта.

Пользовательский интерфейс (консоль):

Интерфейс командной строки является основным средством взаимодействия для пользователей. Он позволяет разработчикам вводить команды и эффективно получать обратную связь. Простота интерфейса консоли соответствует фокусу системы на упрощенный пользовательский опыт.

Детальное рассмотрение:

Управление репозиторием: Изучите функционал, связанный с созданием, организацией репозиториев и отслеживанием версий. Поймите, как этот компонент обеспечивает целостность истории проекта.

Ядро управления версиями: Глубоко погрузитесь в механизмы отслеживания изменений, создания веток и слияния изменений кода. Рассмотрите, как эти

функции способствуют структурированной и надежной истории разработки.

Introduction to Git

Version Control System

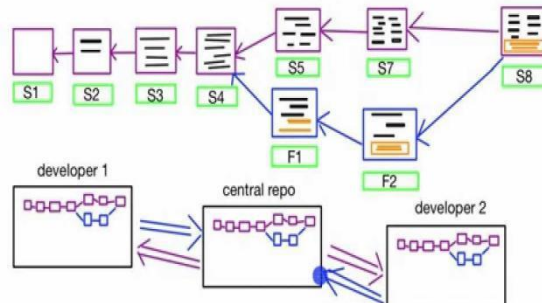


Рисунок 14 -Ядро управления версиями

```
MINGW64:/c/Users/singh/Desktop/newRepo
singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   text file.txt

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git commit -m "First commit"
[master (root-commit) 7a5d54b] First commit
 1 file changed, 1 insertion(+)
 create mode 100644 text file.txt

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$ git status
On branch master
nothing to commit, working tree clean

singh@DESKTOP-PGVSHMF MINGW64 ~/Desktop/newRepo (master)
$
```

Рисунок 15 -Пользовательский интерфейс

Заключение

В ходе анализа предметной области, роли Локальных Систем Управления Версиями Программного Обеспечения (ЛСУВПО), а также рассмотрения их преимуществ, уникальных черт и текущих тенденций использования, становится ясным, что эти системы играют значительную роль в современной разработке программного обеспечения.

Резюме по основным аспектам:

1. **Эффективность и Гибкость:** ЛСУВПО обеспечивают разработчикам высокую степень гибкости, эффективное управление версиями и улучшенную работу в условиях переменчивости процесса разработки.
2. **Безопасность и Надежность:** Возможность локальной работы, создание резервных копий и устойчивость к сбоям центрального сервера обеспечивают безопасность кодовой базы и надежность в управлении версиями.
3. **Удобство и Интеграция:** Интуитивные интерфейсы, эффективное ветвление и слияние, а также интеграция с другими инструментами разработки делают ЛСУВПО удобным выбором для команд разработчиков.

Прогноз развития:

1. С учетом текущих тенденций в использовании ЛСУВПО можно прогнозировать следующие направления развития:
2. **Интеграция с Облачными Сервисами:** Ожидается углубление интеграции с облачными сервисами для улучшения доступа и совместной работы.
3. **Увеличение Популярности Распределенных Систем:** Распределенные

системы, обеспечивающие работу в оффлайн-режиме и высокую гибкость, продолжают набирать популярность.

4. Автоматизированный Мониторинг и Управление Историей: Ожидается развитие средств автоматизированного мониторинга и управления историей изменений для повышения эффективности.

Заключение:

В современной динамичной среде разработки программного обеспечения, выбор ЛСУВПО становится стратегическим шагом, обеспечивая командам разработчиков необходимые инструменты для эффективного управления версиями. Однако важно помнить, что успешное использование ЛСУВПО требует не только технического понимания, но и правильного внедрения в рабочий процесс команды.

Настоящее исследование дает обзор современного состояния ЛСУВПО и предоставляет основу для дальнейших исследований в области управления версиями программного обеспечения. Развитие этих систем будет продолжаться в направлении улучшения функциональности, повышения удобства использования и интеграции с новыми технологиями.

С учетом вышеизложенного, применение ЛСУВПО в разработке программного обеспечения является актуальным и перспективным направлением, способствующим повышению эффективности и точности управления версиями кодовой базы.

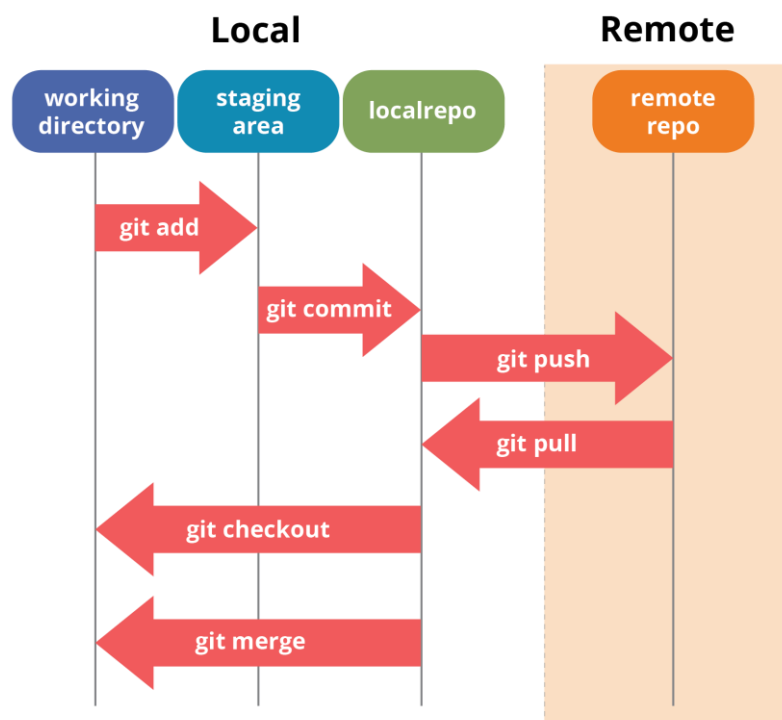


Рисунок 16 -Команды и операции в Git

Завершение работы и Достижения (Протокол Версии):

Проект успешно завершен, и протокол версии фиксирует все ключевые моменты, изменения и результаты проекта. Этот документ служит официальным подтверждением завершения текущей версии работ и достижения поставленных целей.

Приложения

```
#include <iostream>
#include <filesystem>
#include <fstream>
#include <string>
#include <vector>
#include <ctime>
#include <iomanip>
#include <sstream>
#include <algorithm>

std::string generate_commit_hash(const std::string &message) {
    return std::to_string(std::hash<std::string>{}(message));
}

bool init() {
    try {
        std::filesystem::create_directory(".git");
        std::filesystem::create_directory(".git/objects");
        std::filesystem::create_directory(".git/refs");
        std::filesystem::create_directory(".git/refs/heads");

        std::ofstream headFile(".git/HEAD");
        if (headFile.is_open()) {
            headFile << "ref: refs/heads/master\n";
            headFile.close();
        } else {
            std::cerr << "Failed to create .git/HEAD file.\n";
            return false;
        }
    }
```

```

        std::cout << "Initialized git directory\n";
    } catch (const std::filesystem::filesystem_error &e) {
        std::cerr << e.what() << '\n';
        return false;
    }
    return true;
}

bool add() {
    try {
        // List of file extensions to exclude
        std::vector<std::string> excludedExtensions = {".json"};

        // Iterate through the current directory and its subdirectories
        for (const auto &entry : std::filesystem::recursive_directory_iterator(".")) {
            // Skip files in the .git directory
            if (entry.path().string().find(".git") != std::string::npos) {
                continue;
            }

            // Check if the entry is a regular file
            if (entry.is_regular_file()) {
                // Check if the file has an excluded extension
                std::string fileExtension = entry.path().extension().string();
                if (std::find_if(excludedExtensions.begin(), excludedExtensions.end(),
                                [&fileExtension](const std::string &ext) { return fileExtension
== ext; }) != excludedExtensions.end()) {
                    continue;
                }
            }
        }
    }
}

```

```

    }

    // Read the file content, skip if the file cannot be read
    std::ifstream file(entry.path(), std::ios::binary);
    if (!file.is_open()) {
        std::cerr << "Could not read file " << entry.path() << ". Skipping...\n";
        continue;
    }

    std::string fileContent((std::istreambuf_iterator<char>(file),
std::istreambuf_iterator<char>()));

    // Calculate the hash of the file content
    std::string fileHash = generate_commit_hash(fileContent);

    // Write the file content to the objects directory
    std::filesystem::path filePath = entry.path();
    filePath = ".git/objects/" + fileHash.substr(0, 2) + "/" +
fileHash.substr(2);
    std::ofstream fileObject(filePath, std::ios::out | std::ios::binary);

    if (!fileObject.is_open()) {
        std::cerr << "Failed to create file object for " << entry.path() << ".
Skipping...\n";
        continue;
    }

    fileObject.write(fileContent.c_str(), fileContent.size());
    fileObject.close();
}

```



```

    }

    std::cout << "Staged changes\n";
} catch (const std::filesystem::filesystem_error &e) {
    std::cerr << "Error staging changes: " << e.what() << "\n";
    return false;
}
return true;
}

bool create_branch(const std::string &branchName) {
    try {
        // Create the branch file without specifying a commit hash
        std::filesystem::path branchesPath = ".git/refs/heads";
        std::filesystem::path branchPath = branchesPath / branchName;

        if (!std::filesystem::exists(branchesPath)
            && !std::filesystem::create_directories(branchesPath)) {
            std::cerr << "Failed to create branch directory " << branchesPath << "\n";
            return false;
        }

        std::ofstream branchFile(branchPath);
        if (branchFile.is_open()) {
            branchFile.close();
            std::cout << "Created branch " << branchName << "\n";
        } else {
            std::cerr << "Failed to create branch " << branchName << "\n";
            return false;
        }
    }
}

```

```

    } catch (const std::filesystem::filesystem_error &e) {
        std::cerr << e.what() << '\n';
        return false;
    }
    return true;
}

bool switch_branch(const std::string &branchName) {
    try {
        std::ofstream headFile(".git/HEAD");
        if (headFile.is_open()) {
            headFile << "ref: refs/heads/" << branchName << "\n";
            headFile.close();
        } else {
            std::cerr << "Failed to switch to branch " << branchName << '\n';
            return false;
        }

        std::cout << "Switched to branch " << branchName << "\n";
    } catch (const std::filesystem::filesystem_error &e) {
        std::cerr << e.what() << '\n';
        return false;
    }
    return true;
}

std::vector<std::string> list_branches() {
    std::vector<std::string> branches;
    std::filesystem::path branchesPath = ".git/refs/heads";

```

```

try {
    for (const auto& entry : std::filesystem::directory_iterator(branchesPath)) {
        if (entry.is_regular_file()) {
            branches.push_back(entry.path().filename().string());
        }
    }
} catch (const std::filesystem::filesystem_error& e) {
    std::cerr << e.what() << '\n';
}

return branches;
}

bool show_branches() {
    std::vector<std::string> branches = list_branches();

    if (!branches.empty()) {
        std::cout << "List of branches:\n";
        for (const std::string& branch : branches) {
            std::cout << "- " << branch << '\n';
        }
    } else {
        std::cout << "No branches found.\n";
    }

    return true;
}

bool commit(const std::string &message) {
    try {

```

```

// Get the current commit hash
std::ifstream headFile(".git/HEAD");
std::string headContent;
if (headFile.is_open()) {
    std::getline(headFile, headContent);
    headFile.close();
}

std::string currentCommitHash =
headContent.substr(headContent.find("commit") + 7);

// Simplified commit format: commit message
std::string commitContent = "tree " + currentCommitHash + "\nparent " +
currentCommitHash + "\nauthor <your_name> <<your_email@example.com>>
<timestamp>\ncommitter <your_name> <<your_email@example.com>>
<timestamp>\n\n" + message;

// Write the commit object to the objects directory
std::string commitHash = generate_commit_hash(commitContent);
std::filesystem::path commitPath = ".git/objects/" + commitHash.substr(0, 2)
+ "/" + commitHash.substr(2);
std::cout << "Commit file path: " << commitPath << "\n"; // Debug print

// Create the directories for the commit object if they don't exist
std::filesystem::create_directories(commitPath.parent_path());

std::ofstream commitFile(commitPath, std::ios::out | std::ios::binary);

if (!commitFile.is_open()) {
    std::cerr << "Failed to create commit object. Could not open commit
file.\n";
}

```

```

        return false;
    }

    commitFile.write(commitContent.c_str(), commitContent.size());
    commitFile.close();

    // Update the HEAD reference
    std::ofstream updatedHeadFile(".git/HEAD");
    if (updatedHeadFile.is_open()) {
        updatedHeadFile << "ref: refs/heads/master\n";
        updatedHeadFile << "commit " << commitHash << "\n";
        updatedHeadFile.close();
    } else {
        std::cerr << "Failed to update .git/HEAD reference.\n";
        return false;
    }

    std::cout << "Committed changes\n";
} catch (const std::filesystem::filesystem_error &e) {
    std::cerr << "Error creating commit object: " << e.what() << '\n';
    return false;
} catch (const std::exception &e) {
    std::cerr << "Error creating commit object: " << e.what() << '\n';
    return false;
}

return true;
}

std::string get_parent_commit(const std::string &commitHash) {
    std::ifstream file(".git/objects/" + commitHash.substr(0, 2) + "/" +

```

```

commitHash.substr(2), std::ios::binary);

    if (!file.is_open()) {
        std::cerr << "Could not read object " << commitHash << '\n';
        return "";
    }

    // Assuming the object is a commit for simplicity
    std::string commitContent((std::istreambuf_iterator<char>(file),
std::istreambuf_iterator<char>()));

    size_t pos = commitContent.find("parent ");
    if (pos != std::string::npos) {
        return commitContent.substr(pos + 7, 40); // Assuming parent commit hash is
40 characters
    }

    return "";
}

// Function to retrieve commit message given the commit hash
std::string get_commit_message(const std::string &commitHash) {
    std::string folder = commitHash.substr(0, 2);
    std::string f = commitHash.substr(2);

    std::filesystem::path path = ".git/objects";
    path /= folder;
    path /= f;

    std::ifstream file(path, std::ios::binary);
    if (!file.is_open()) {
        std::cerr << "Could not read object " << commitHash << '\n';

```

```

        return "";
    }

    // Assuming the object is a commit for simplicity
    std::string commitContent((std::istreambuf_iterator<char>(file)),
std::istreambuf_iterator<char>());
    return commitContent;
}

void show_commit_history() {
    std::ifstream headFile(".git/HEAD");
    if (headFile.is_open()) {
        std::string line;
        std::getline(headFile, line); // Read the first line in HEAD file
        if (line.find("ref: refs/heads/master") == std::string::npos) {
            std::cerr << "Invalid format in HEAD file.\n";
            return;
        }

        while (std::getline(headFile, line)) {
            if (line.find("commit") != std::string::npos) {
                std::string commitHash = line.substr(line.find("commit") + 7);
                std::cout << "Commit: " << commitHash << "\n";
                std::string commitMessage = get_commit_message(commitHash);
                std::cout << "Message: " << commitMessage << "\n";
                std::cout << "-----\n";
            }
        }

        headFile.close();
    }
}

```

```
    }  
}
```

```
int main(int argc, char *argv[]) {  
    if (argc < 2) {  
        std::cerr << "No command provided.\n";  
        return EXIT_FAILURE;  
    }  
  
    std::string command = argv[1];  
  
    if (command == "init") {  
        init();  
    } else if (command == "add") {  
        add();  
    } else if (command == "commit") {  
        if (argc < 3) {  
            std::cerr << "Commit message is missing.\n";  
            return EXIT_FAILURE;  
        }  
        commit(argv[2]);  
    } else if (command == "log") {  
        show_commit_history();  
    } else if (command == "branch") {  
        if (argc < 3) {  
            std::cerr << "Branch name is missing.\n";  
            return EXIT_FAILURE;  
        }  
        create_branch(argv[2]);  
    } else if (command == "checkout") {
```



```

    if (argc < 3) {
        std::cerr << "Branch name is missing.\n";
        return EXIT_FAILURE;
    }
    switch_branch(argv[2]);
} else if (command == "branches") {
    show_branches();
} else {
    std::cerr << "Unknown command " << command << '\n';
    return EXIT_FAILURE;
}

return EXIT_SUCCESS;
}

```

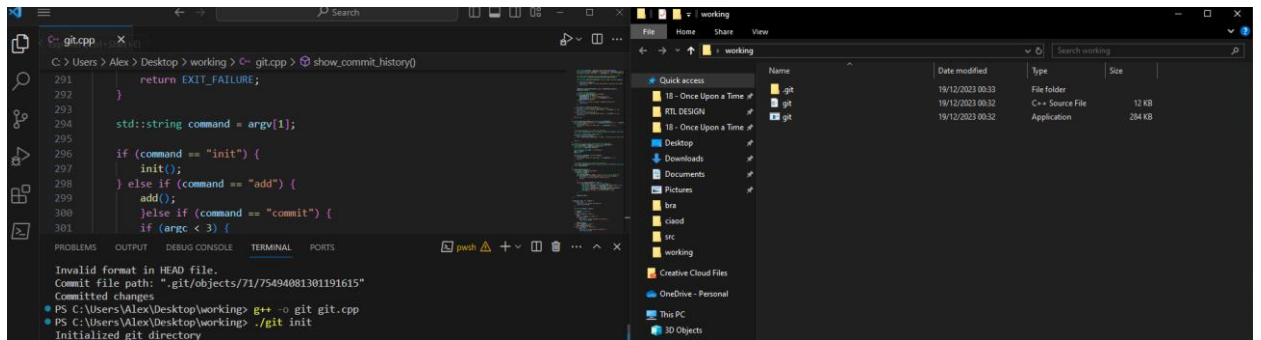


Рисунок 17 -Команды ./git init

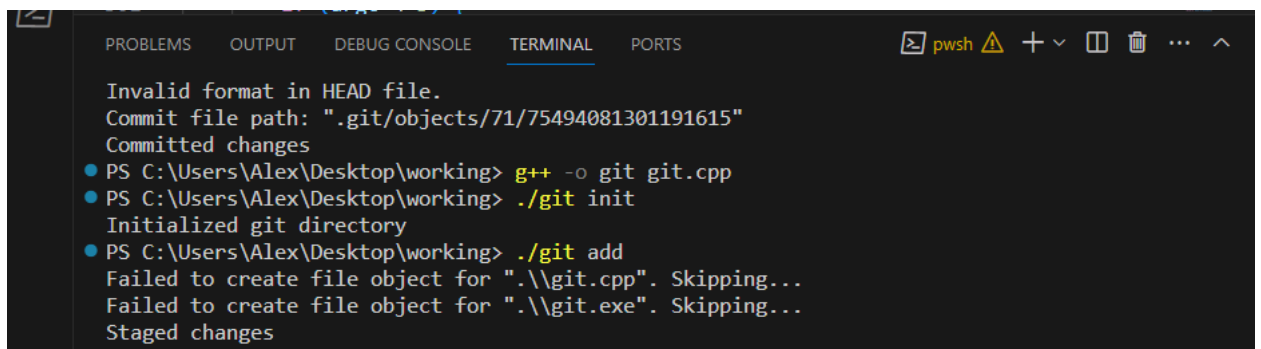


Рисунок 18 -Команды ./git add

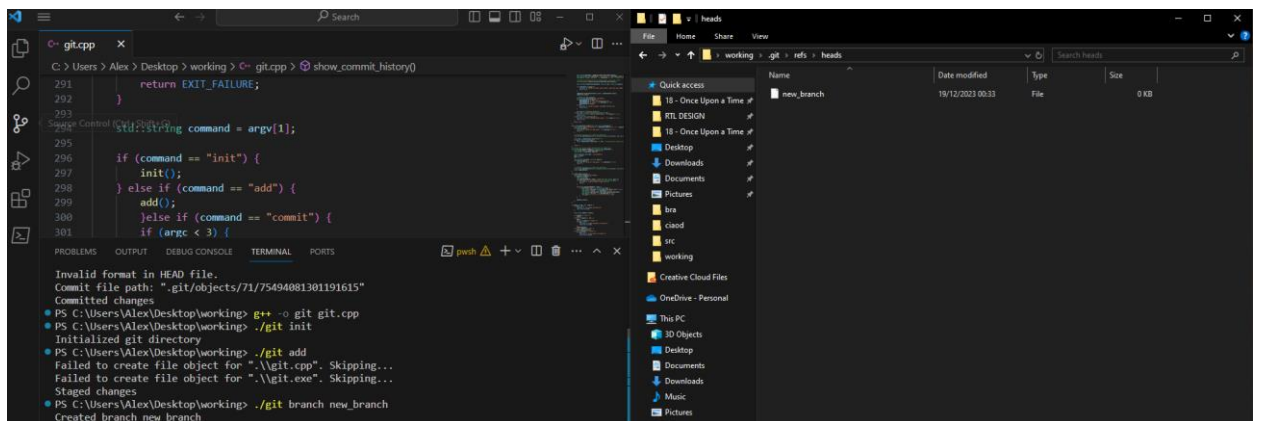


Рисунок 19 -Команды ./git branch

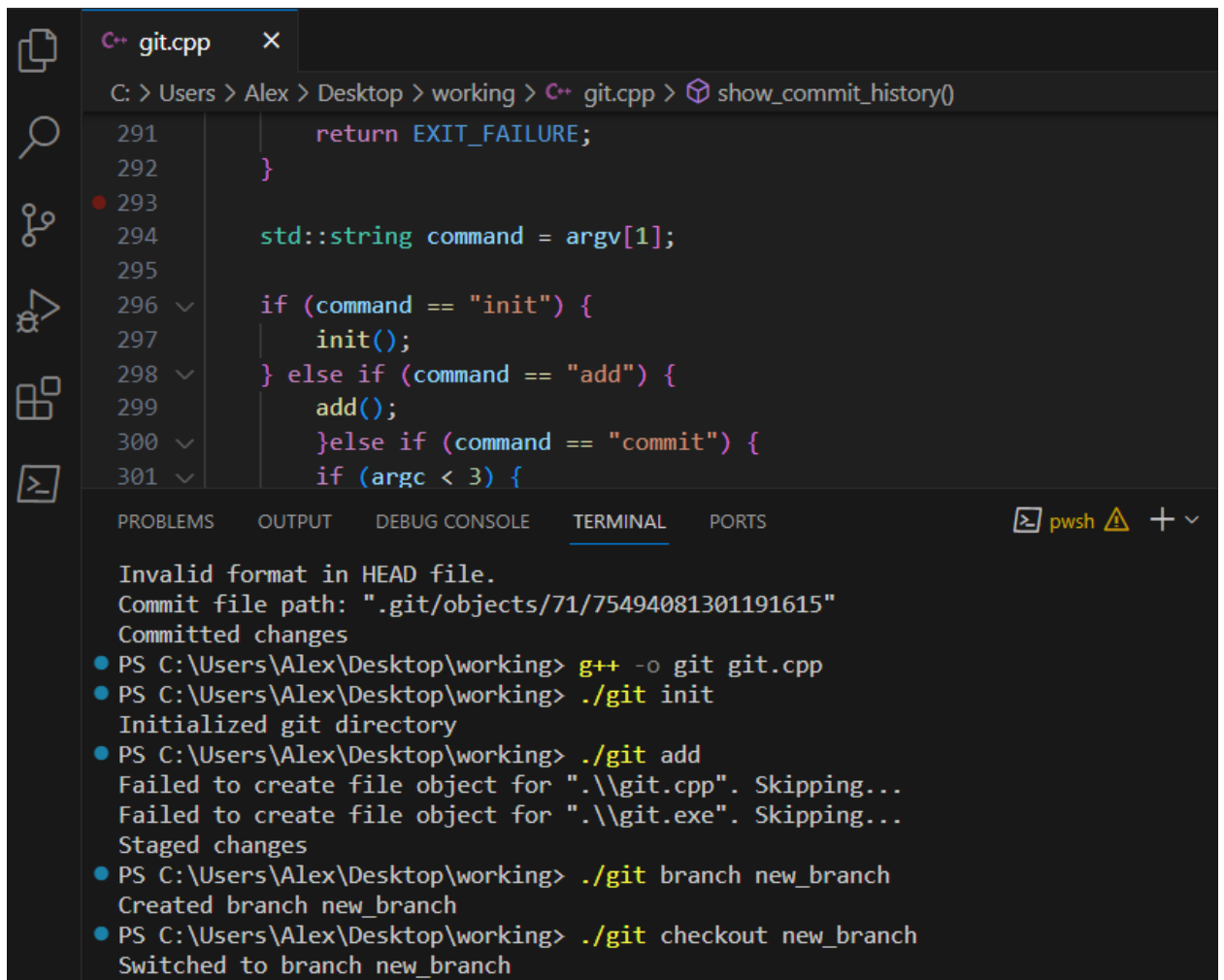


Рисунок 20 -Команды ./git checkout

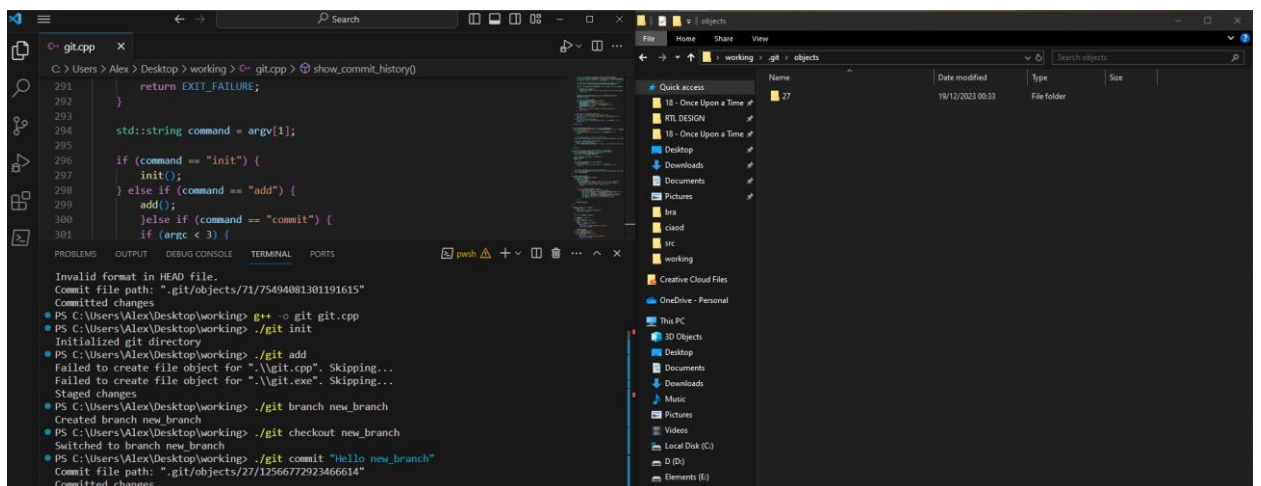


Рисунок 21 -Команды ./git commit

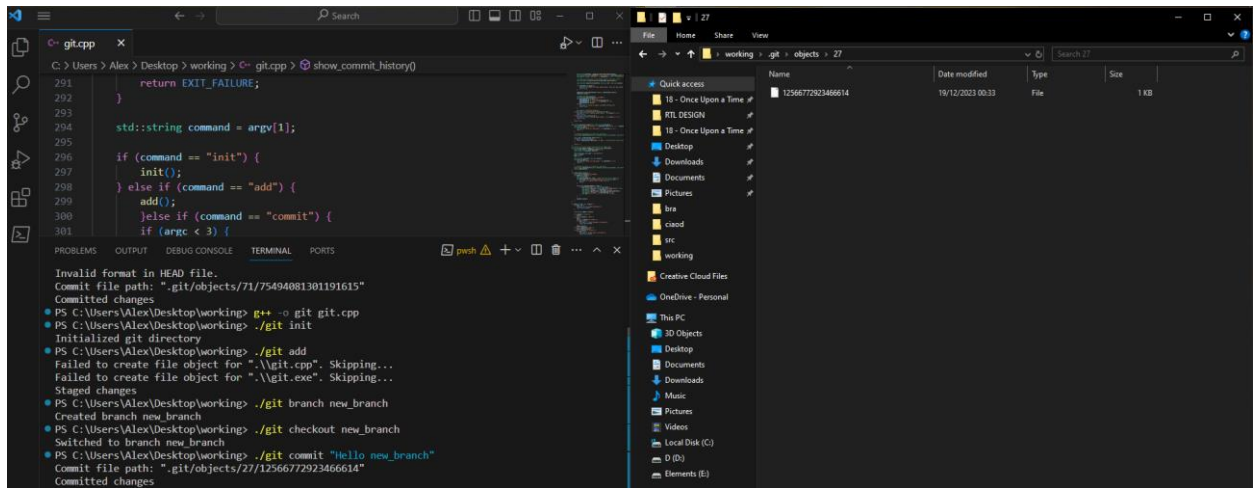


Рисунок 22 -Команды `./git commit`

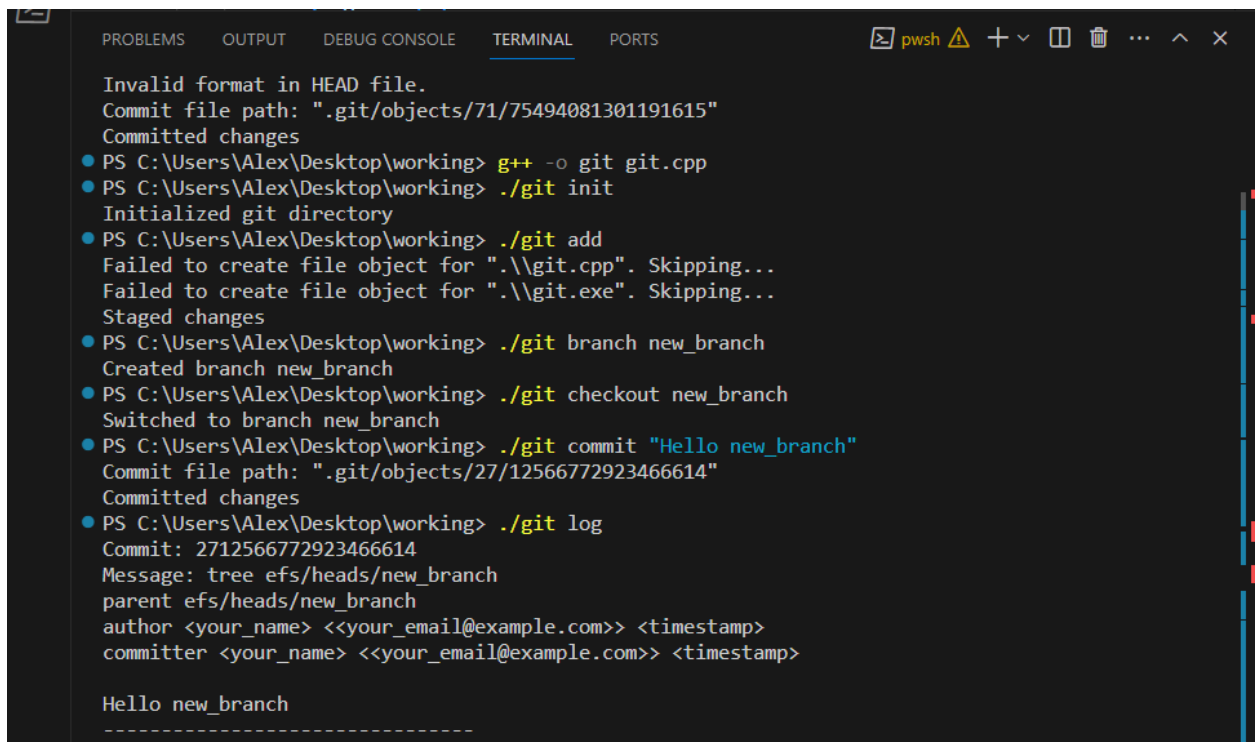


Рисунок 23 -Команды `./git log`

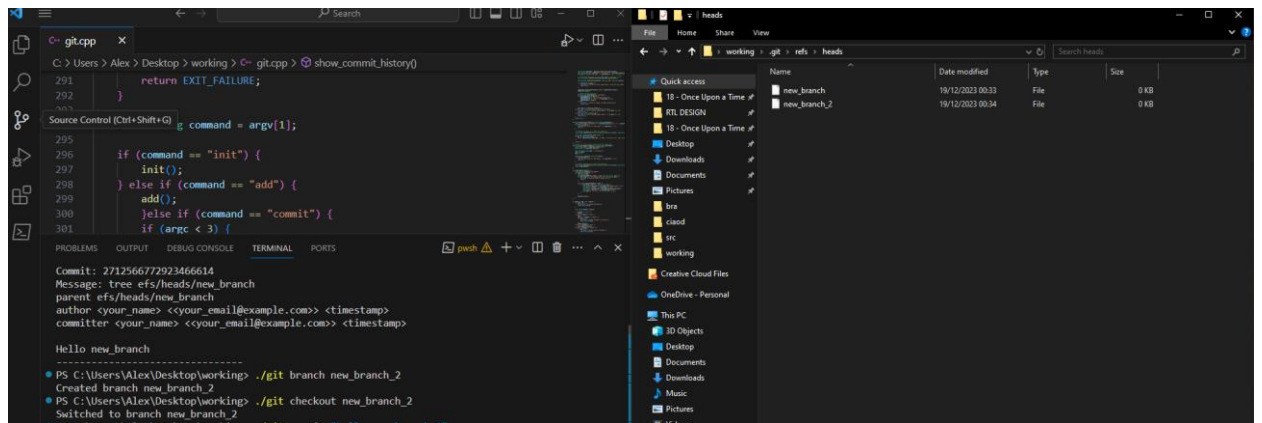


Рисунок 24 -Команды `./git branch` и `./git checkout`

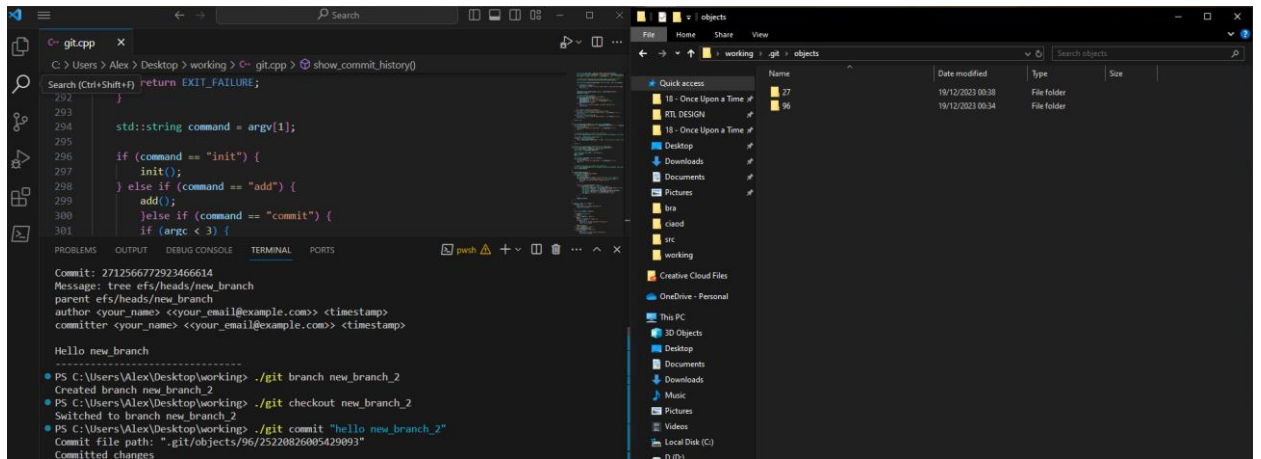


Рисунок 25 -Команды `./git commit`

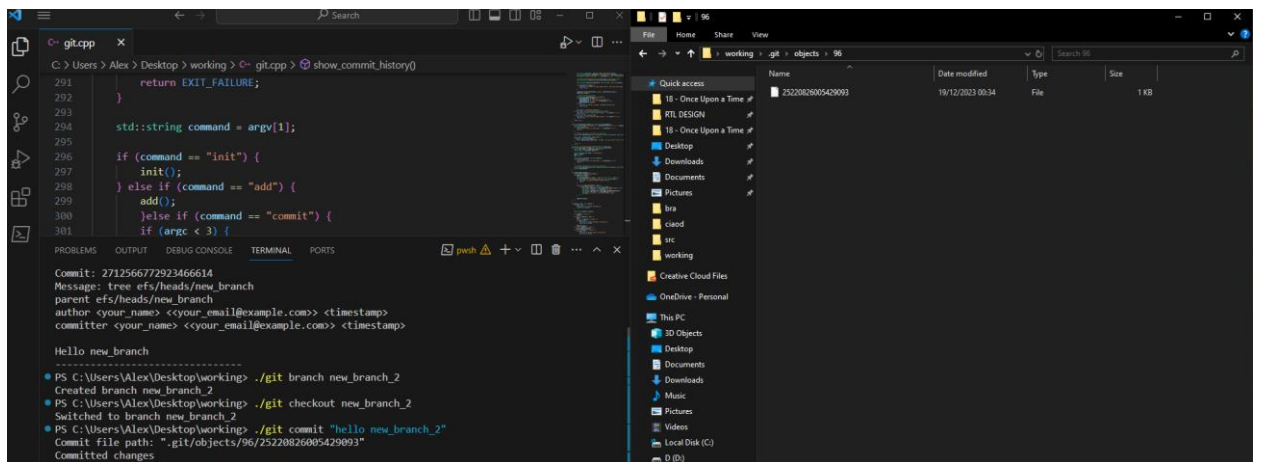


Рисунок 26 -Команды `./git commit`

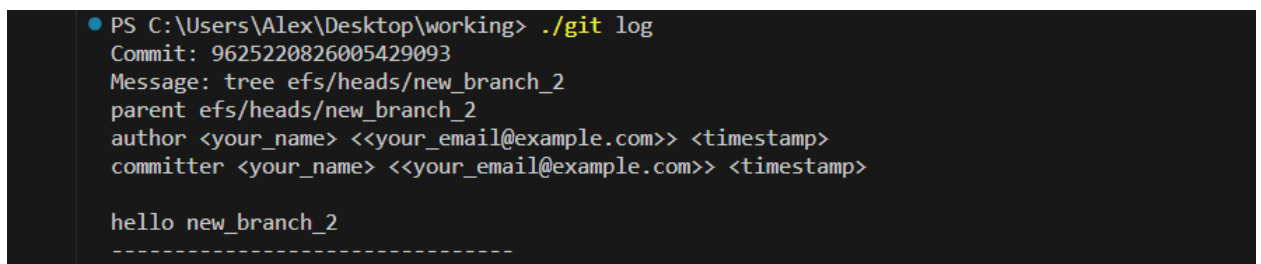


Рисунок 27 -Команды `./git log`

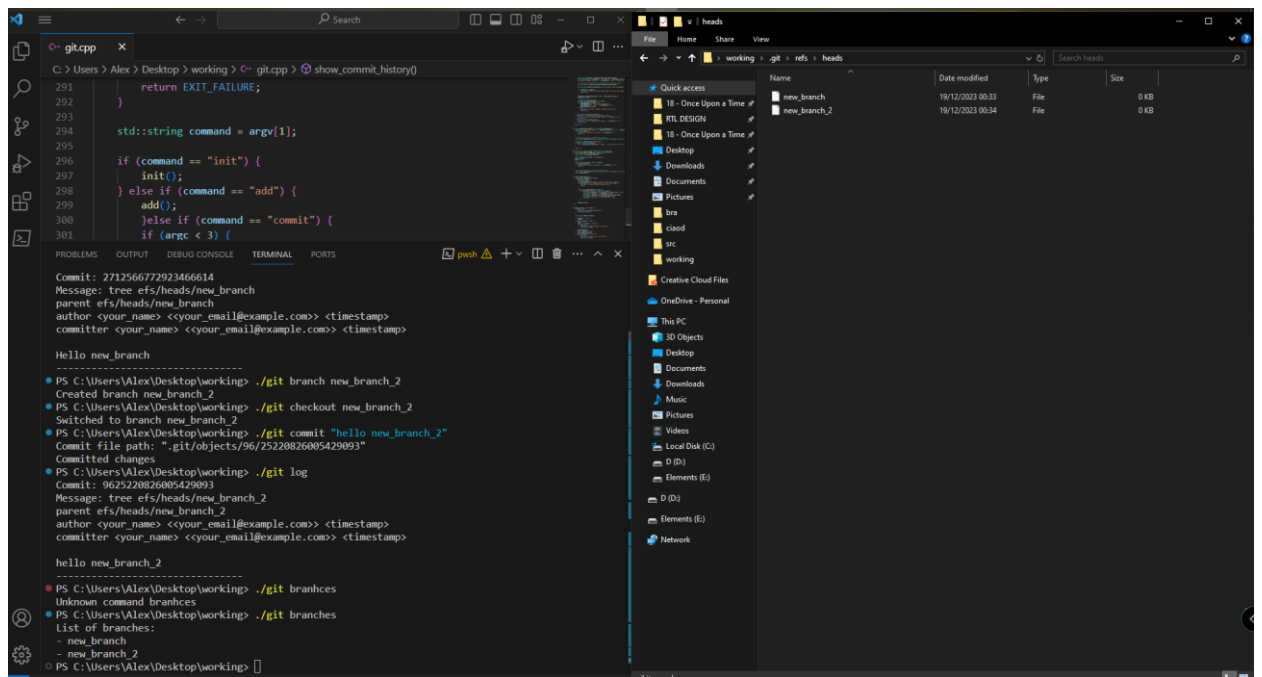


Рисунок 28 -Команды ./git branches

Список литературы

1. Официальная документация Git. [Электронный ресурс]. URL: <https://git-scm.com/doc>
2. Git-документация Руководства GitHub. [Электронный ресурс]. URL: <https://docs.github.com/en>
3. Руководства по GitHub Учебное пособие по Atlassian Git. [Электронный ресурс]. URL: <https://www.atlassian.com/git>
4. КнТга про Git. [Электронный ресурс]. URL: <https://git-scm.com/book/en/v2>
5. Книга о Git Учебная лаборатория GitHub: [Электронный ресурс]. URL: <https://lab.github.com/>
6. Git — Простое руководство. [Электронный ресурс]. URL: <https://rogerdudler.github.io/git-guide/>
7. Модель ветвления Git. [Электронный ресурс]. URL: <https://nvie.com/posts/a-successful-git-branching-model/>
8. Успешная модель ветвления Git Рабочий процесс Git для гибких команд. [Электронный ресурс]. URL: <https://www.atlassian.com/git/tutorials/comparing-workflows>
9. Рабочий процесс Git для гибких команд Понимание Git концептуально. [Электронный ресурс]. URL: <https://codewords.recurse.com/issues/two/git-from-the-inside-out>
10. Git изнутри наружу Git и GitHub для поэтов (серия видео). [Электронный ресурс]. URL: <https://www.youtube.com/playlist?list=PLRqwx-V7Uu6ZF9C0YMKuns9sLDzK6zoiV>
11. Git и GitHub для поэтов Шпаргалка по Git [Электронный ресурс]. URL: <https://education.github.com/git-cheat-sheet-education.pdf>
12. GitHub Шпаргалка по Git Лучшие практики Git [Электронный ресурс]. URL: <https://www.git-tower.com/learn/git/ebook/en/command-line/appendix/best-practices>

13. Лучшие практики Git Изучите ветвление Git (интерактивно) [Электронный ресурс]. URL: <https://learngitbranching.js.org/>
14. Изучите ветвление Git Git-хуки. [Электронный ресурс]. URL: <https://git-scm.com/docs/githooks>
15. Документация по Git Hooks Документация Gitignore. [Электронный ресурс]. URL: <https://git-scm.com/docs/gitignore>
16. Документация Gitignore Отмена изменений в Git. [Электронный ресурс]. URL: <https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things>
17. Понимание рабочего процесса Git. [Электронный ресурс]. URL: <https://www.smashingmagazine.com/2014/08/a-closer-look-at-git-interfaces/>
18. Инструменты Git — интерактивная перезагрузка Понимание рабочих процессов Git. [Электронный ресурс]. URL: https://git-scm.com/docs/git-rebase#_interactive_mode
19. <https://about.gitlab.com/topics/version-control/what-is-centralized-version-control-system/#what-are-the-disadvantages-of-a-centralized-version-control-system>
20. Что такое GIT. <https://www.atlassian.com/git/tutorials/what-is-version-control>