МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет по лабораторной работе № 1**

по дисциплине: “Конструирование программного обеспечения”

на тему: ***“*Система контроля версий GIT GUI*”***

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | студент группы 10701122  Бульбочка В.Д. |
| Принял: |  | пр. Станкевич С.Н. |

Минск 2024

# Лабораторная работа №1. Система контроля версий GIT GUI.

**Цель работы:** Ознакомиться с системами контроля версий для разработки программного обеспечения. Освоить способы организации работы в системах контроля версий. Получить навыки командной работы.

**Выполнение работы:**

**Создание хранилища**

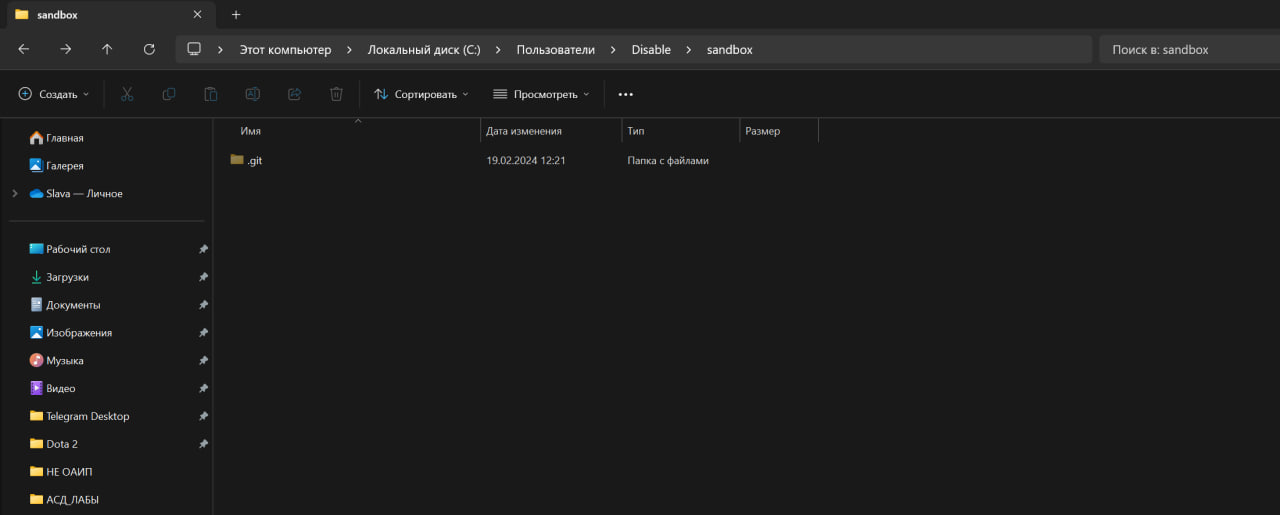


Рисунок 1 – Созданное хранилище

Этот процесс включает создание папки для проекта, запуск Git GUI, создание нового репозитория, что создает скрытую папку .git, и настройку информации о пользователе в Git GUI. Точка в начале имени файла (.git) указывает на то, что он скрыт от обычного просмотра в проводнике или файловой системе.

**Заполнение имя пользователя и почты**

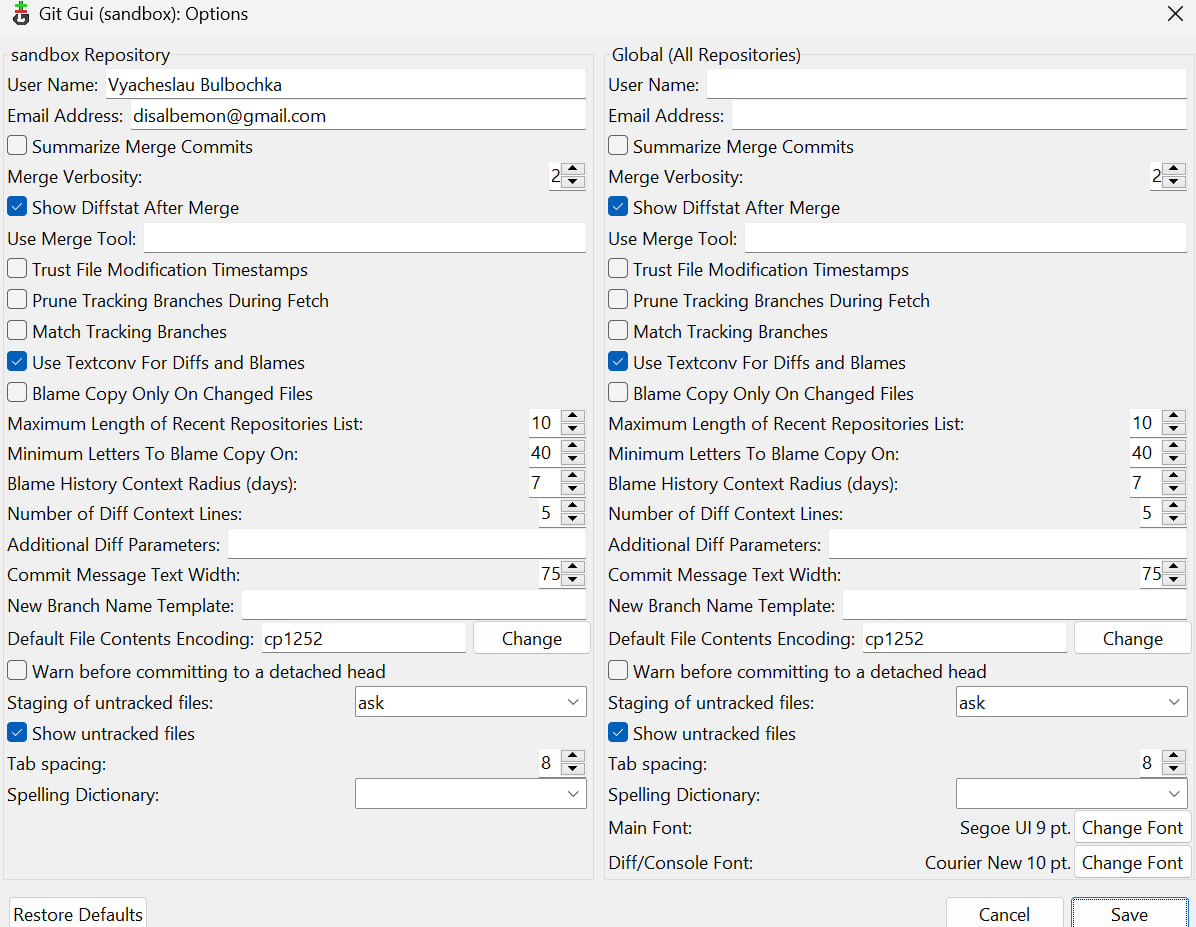


Рисунок 2 – Заполненные данные

**Закрепление изменений (commiting)**

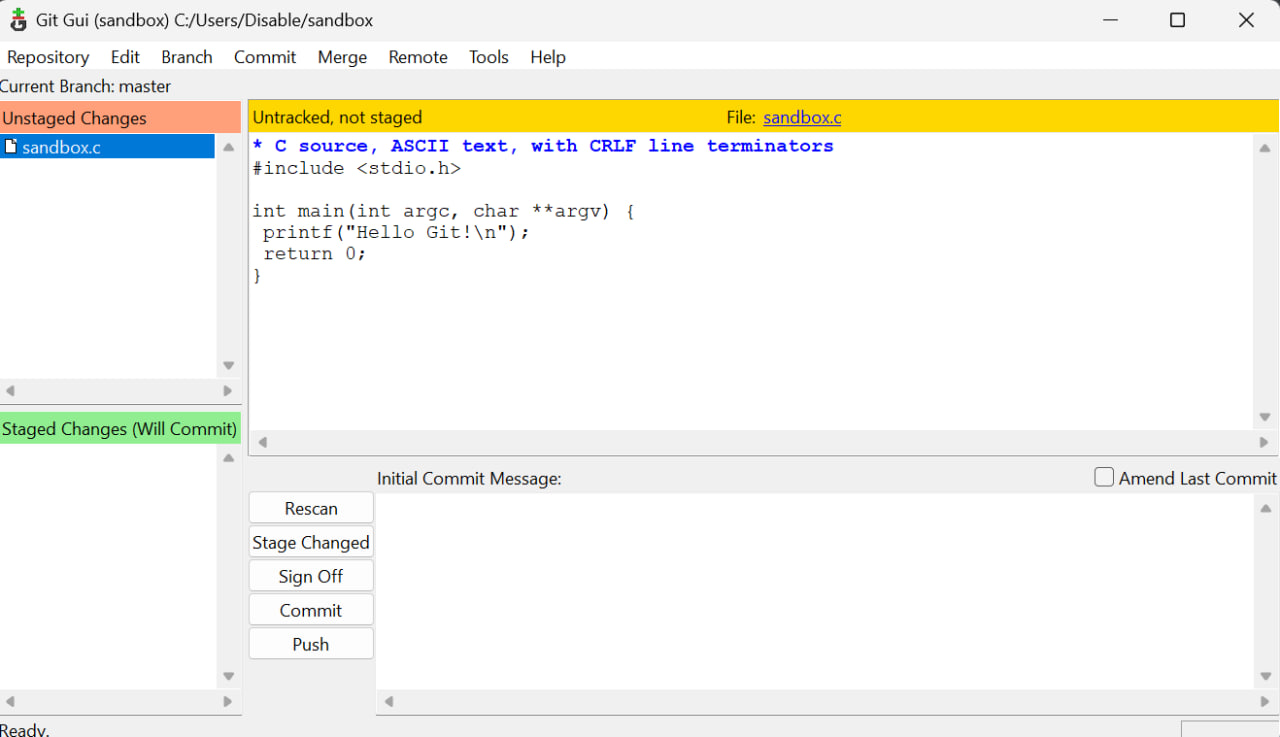


Рисунок 3 – Созданный файл

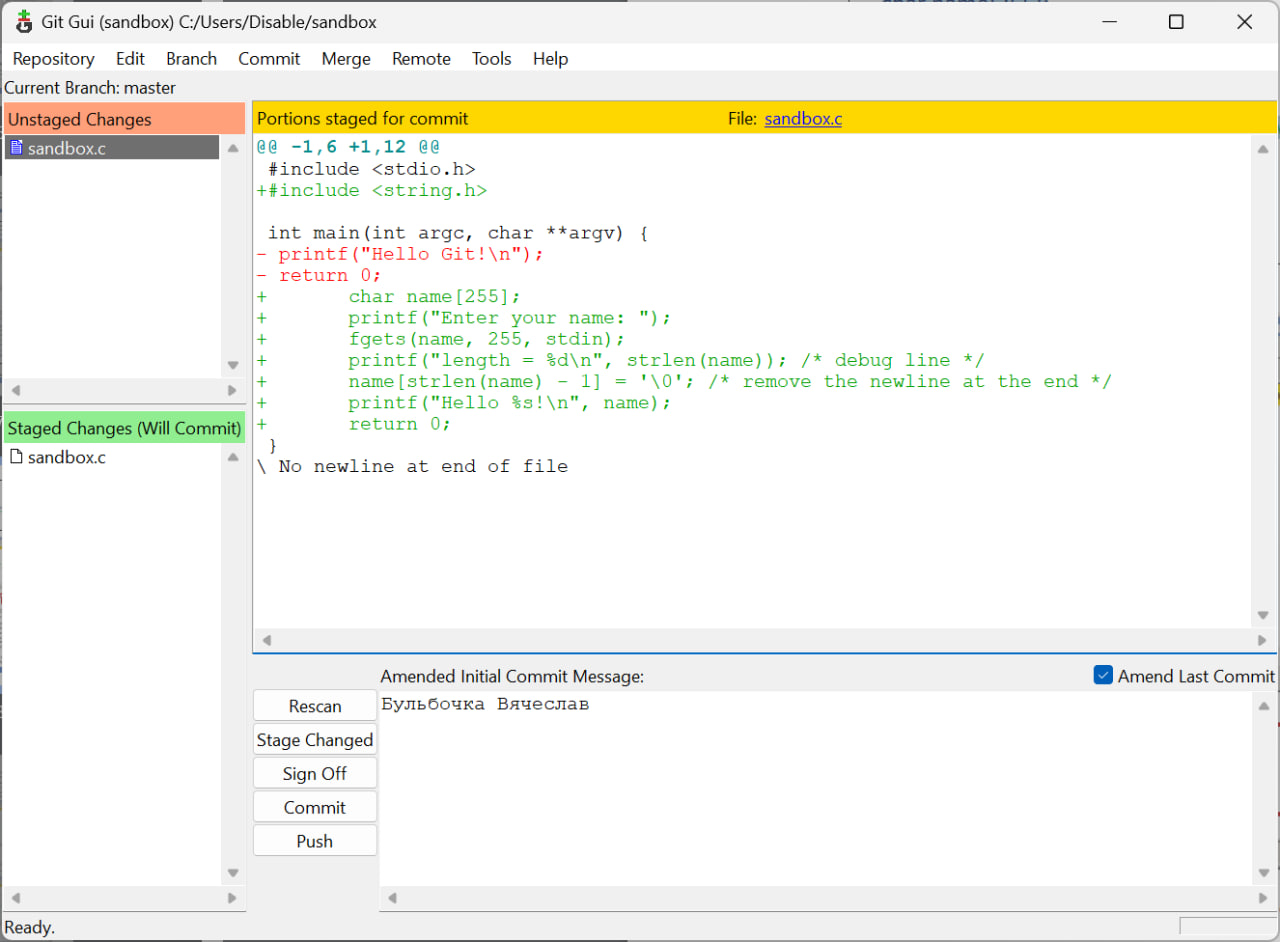


Рисунок 4 – Фиксация изменений

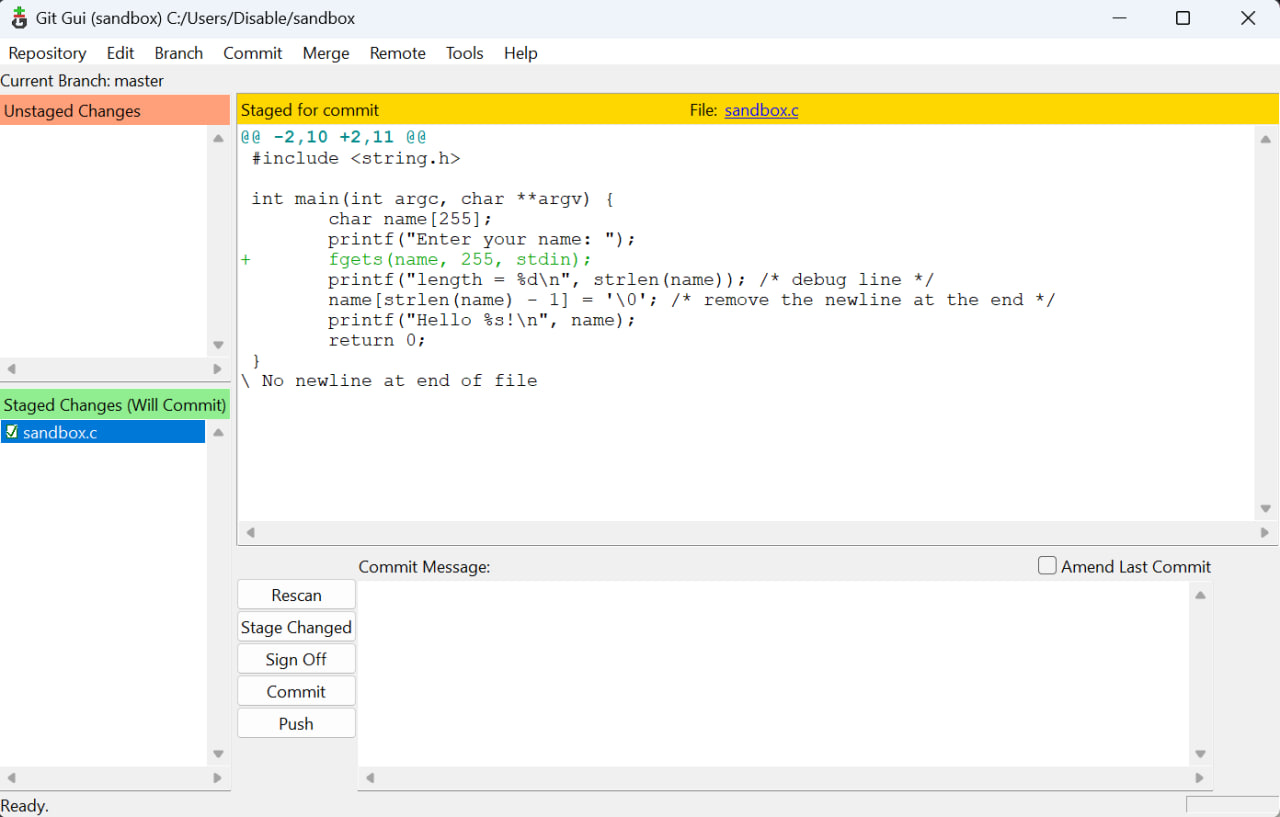


Рисунок 5 – Подготовка отладочной линии

В данном разделе выполнены следующие шаги: создание файла для фиксации изменений в репозитории, добавление этого файла в фиксацию через Git GUI, указание сообщения о фиксации и сохранение изменений. Затем произведена персонализация программы и добавление отладочной строки кода. Для исключения отладочной строки из фиксации, она была убрана из подготовленных изменений, после чего снова были указаны сообщение о фиксации и сохранение изменений.

**Ветвление**

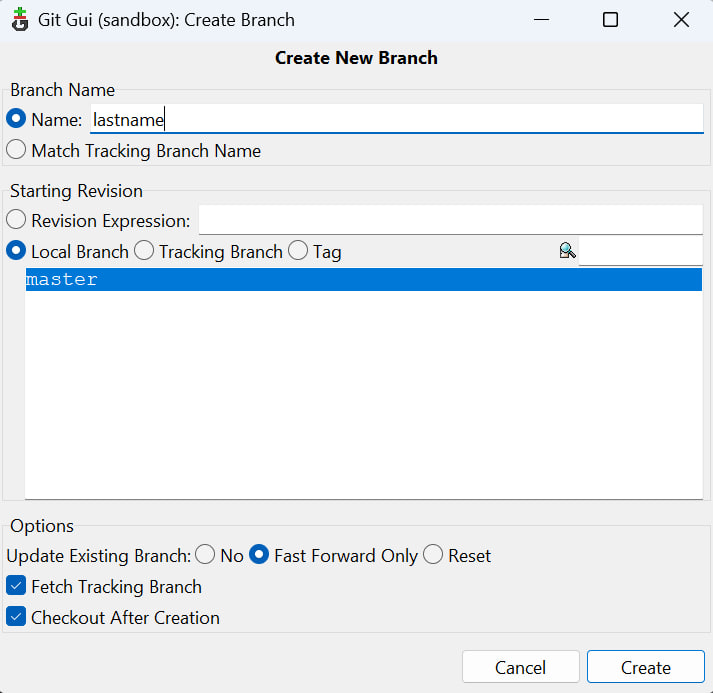


Рисунок 6 – Создание ветки

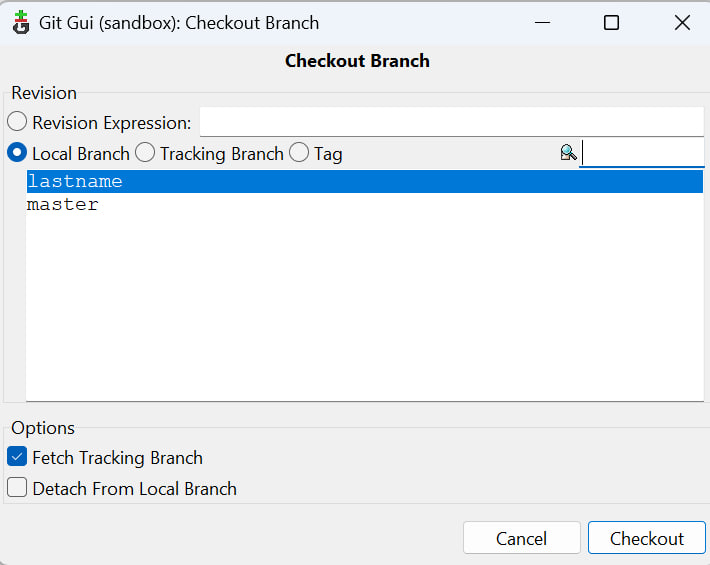


Рисунок 7 – Созданная ветка

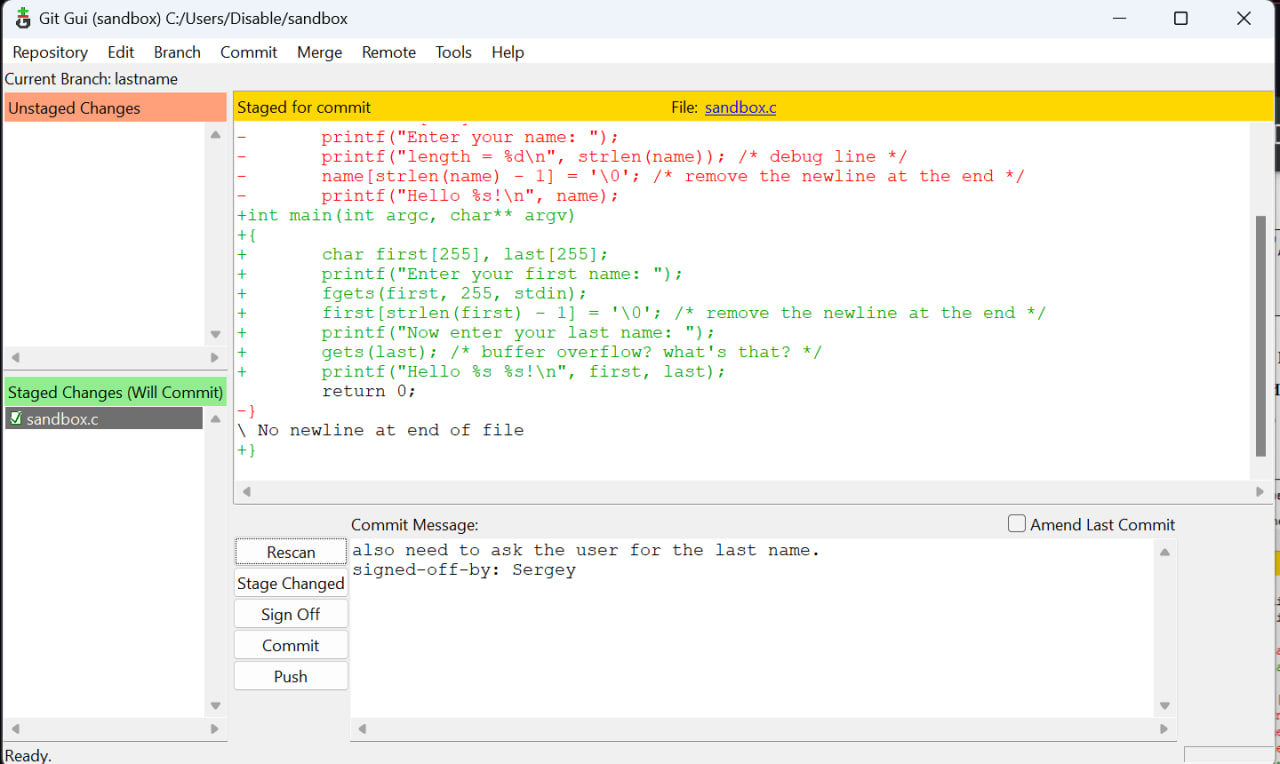


Рисунок 8 – Фиксация изменения в новой ветки

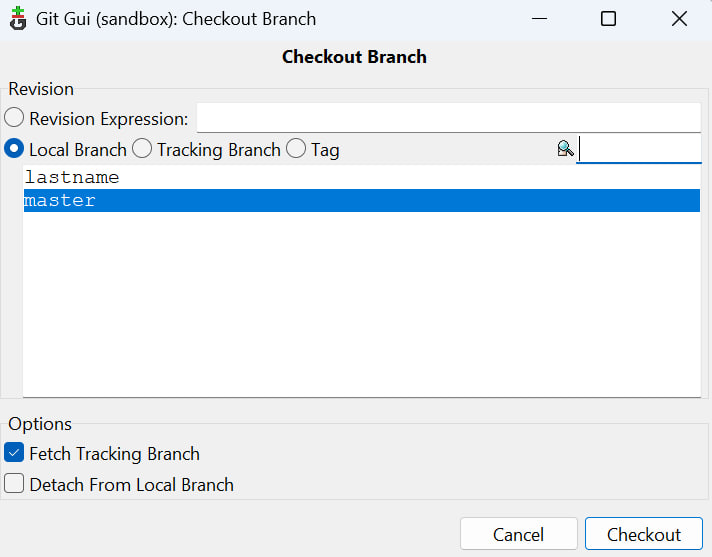


Рисунок 9 – Переход на основную ветку

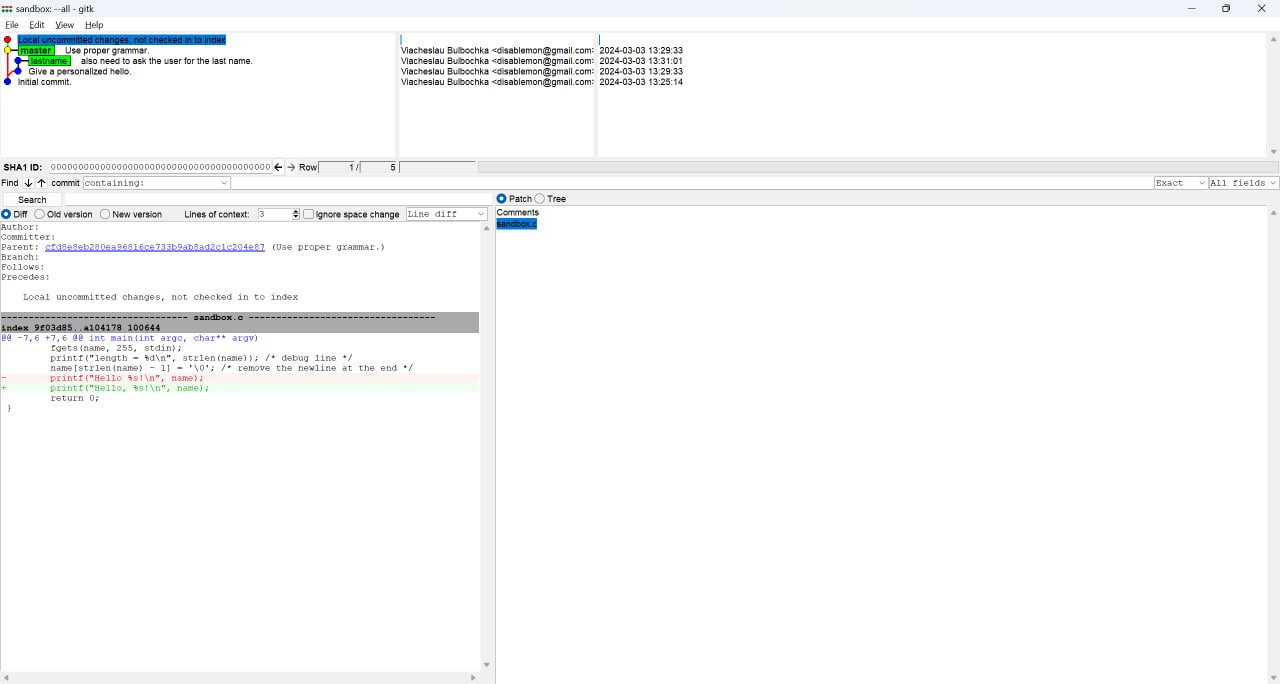


Рисунок 10 – Демонстрация веток после изменений

В данном разделе мы добавляли новые функции в следующую версию программы. Для обеспечения стабильности основной версии и возможности исправления ошибок создали отдельную ветку разработки. Создание ветки производилось через Git GUI, где мы задали название ветки и зафиксировали изменения. Для исправления ошибки в основной ветке сначала переключились на неё, а затем внесли необходимые изменения. Историю изменений можно просмотреть через опцию "Visualize All Branch History", где она представлена в виде графа и таблицы.

**Слияние**

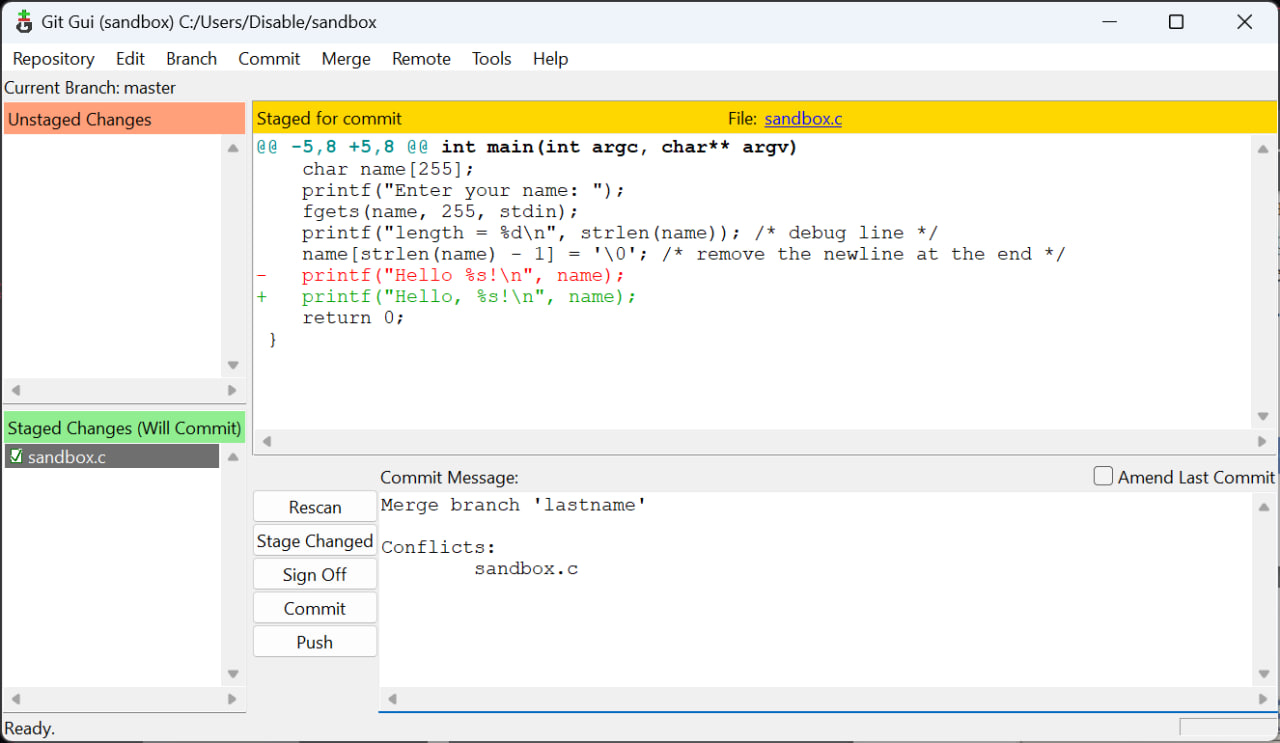


Рисунок 11 – Сохранение слияния веток

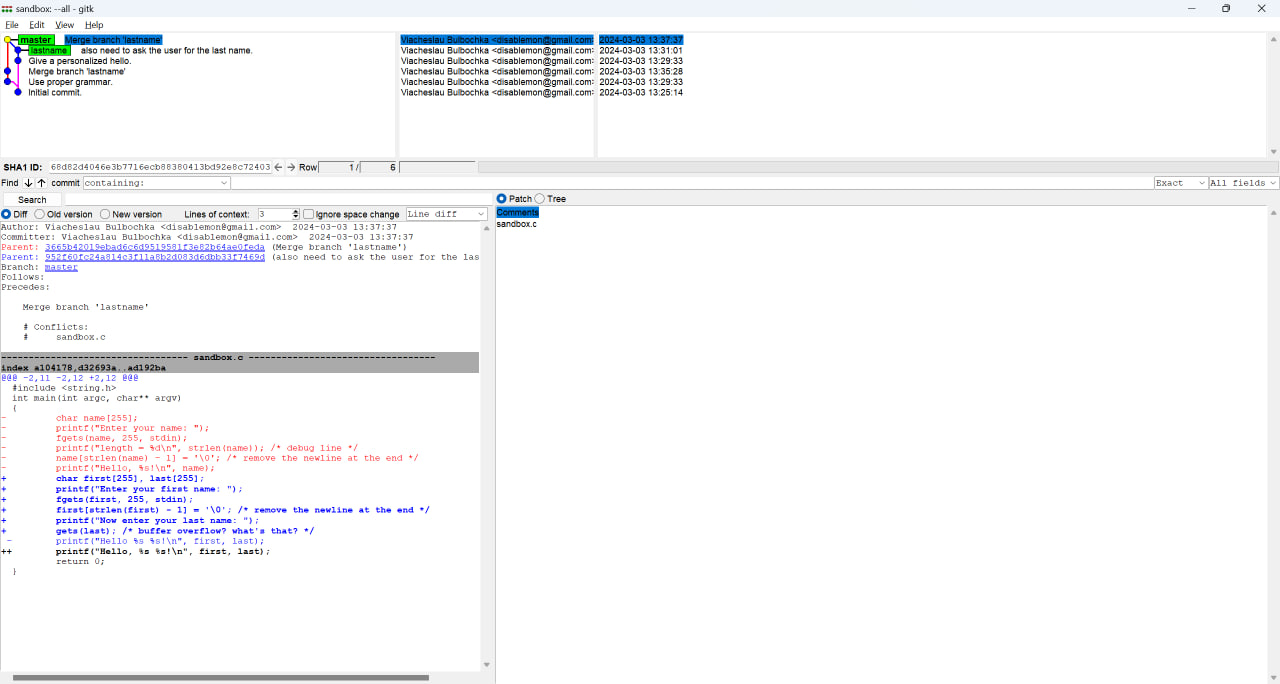


Рисунок 12 – Демонстрация слияния веток

В данном разделе мы выполняли слияние ветки lastname в основную ветку master. Для этого использовалась опция "Merge → Local Merge". Поскольку две разные фиксации внесли изменения в одну и ту же строку, возник конфликт, который необходимо было разрешить вручную, используя текстовый редактор. После разрешения конфликта изменения были подготовлены и зафиксированы слиянием с помощью кнопки "Commit".

**Просмотр истории**

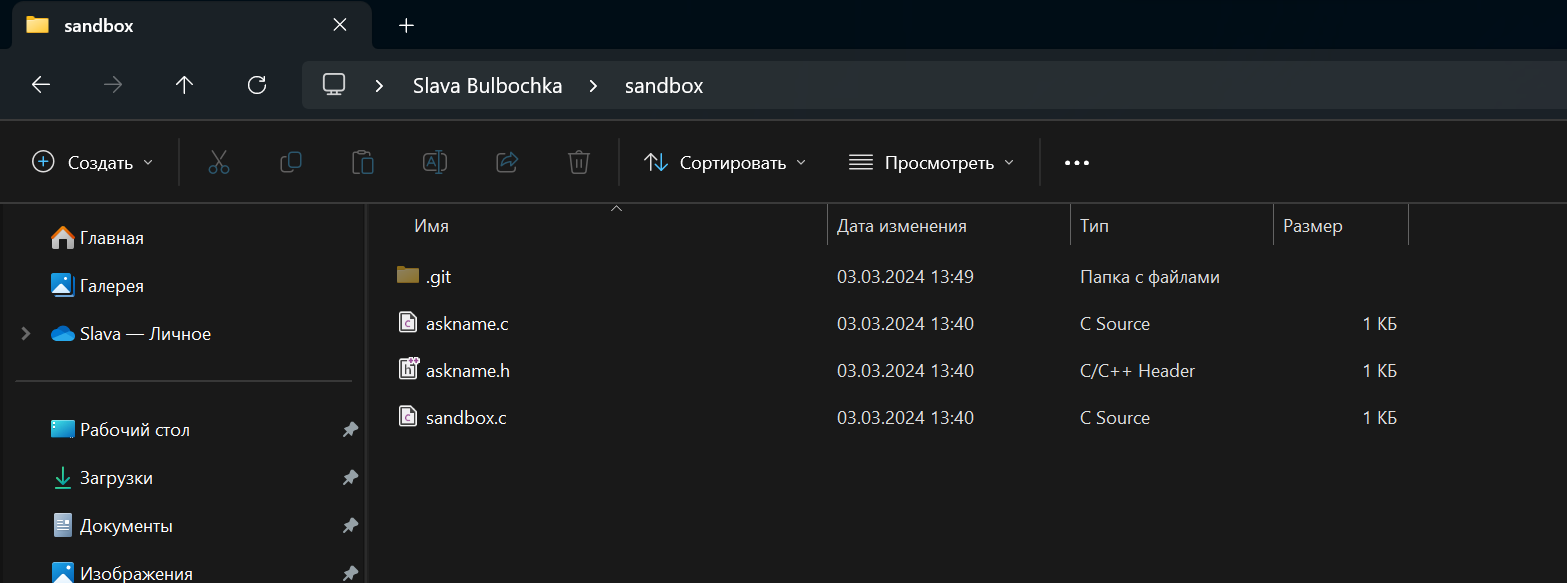


Рисунок 13 – Создание файлов

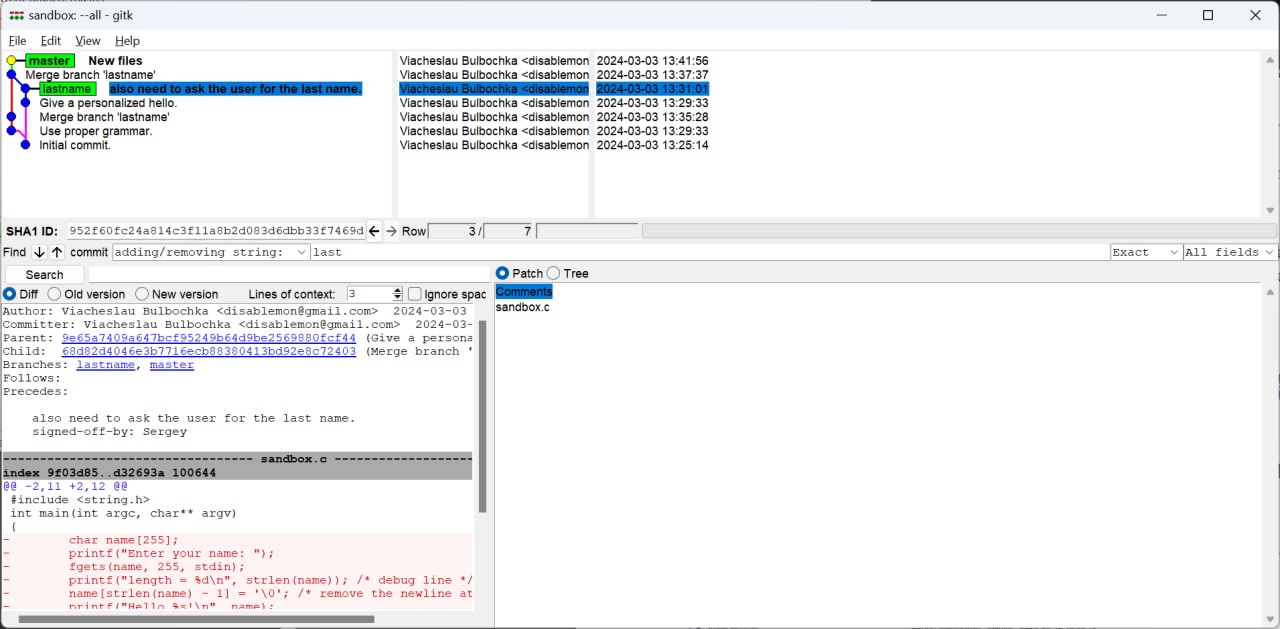


Рисунок 14 – Поиск в истории добавления last переменной.

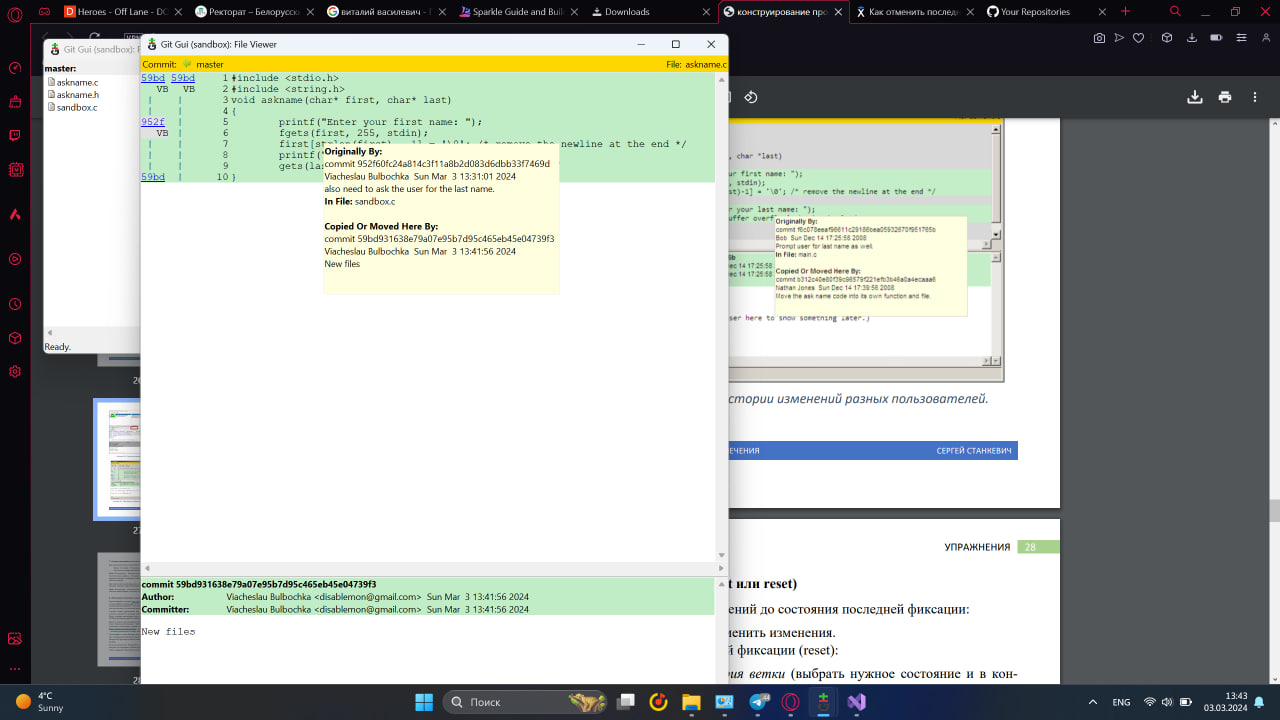


Рисунок 15 - Просмотр истории изменений разных пользователей

В данном разделе мы вынесли код, спрашивающий имя пользователя, в отдельную функцию и, возможно, в отдельный файл. Теперь репозиторий содержит файлы main.c, askname.c и askname.h, которые были созданы в редакторе. После этого мы перечитали репозиторий, подготовили все изменения и сохранили их.

Для изучения истории хранилища мы использовали опцию "Repository → Visualize All Branch History". Мы пытались найти, в какой фиксации была добавлена переменная last, а для этого искали все фиксации, в которых добавлялось или убиралось слово last. Подходящие фиксации были отмечены жирным шрифтом для быстрого обнаружения, и мы могли просмотреть старую и новую версии кода, выделенные цветом.

Мы также узнали о возможности использования git blame для определения автора последнего изменения строки кода. Этот инструмент позволяет отследить изменения даже через несколько дней после их внесения, а также определить, кто и когда последний раз редактировал определенную строку кода.

**Публикация изменений**

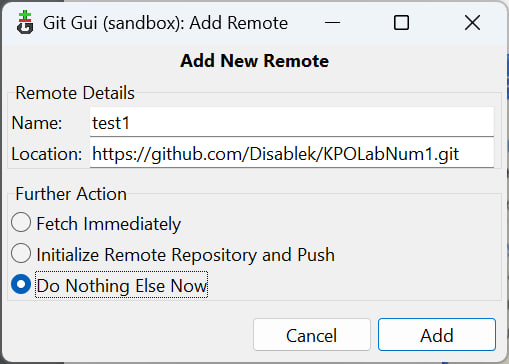


Рисунок 16 – Добавление внешнего репозитория

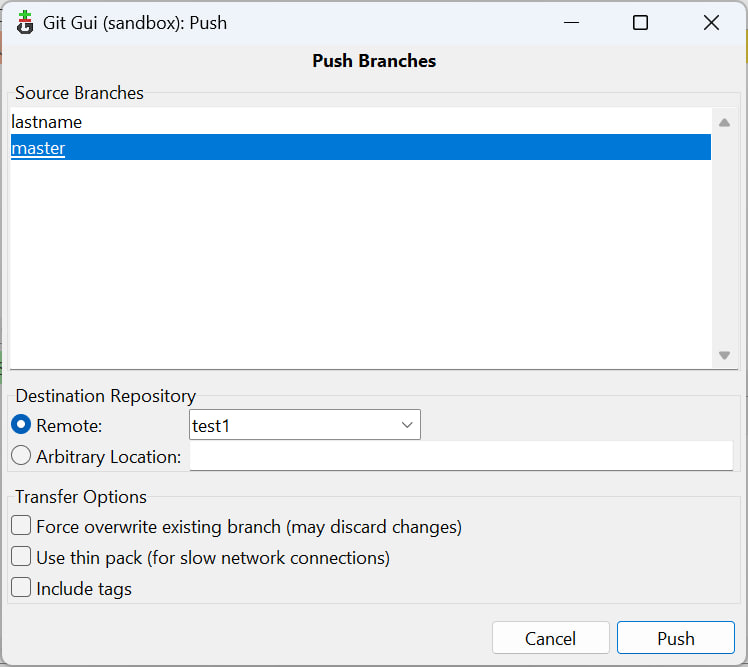


Рисунок 17 – Отправка изменений в ветвях.

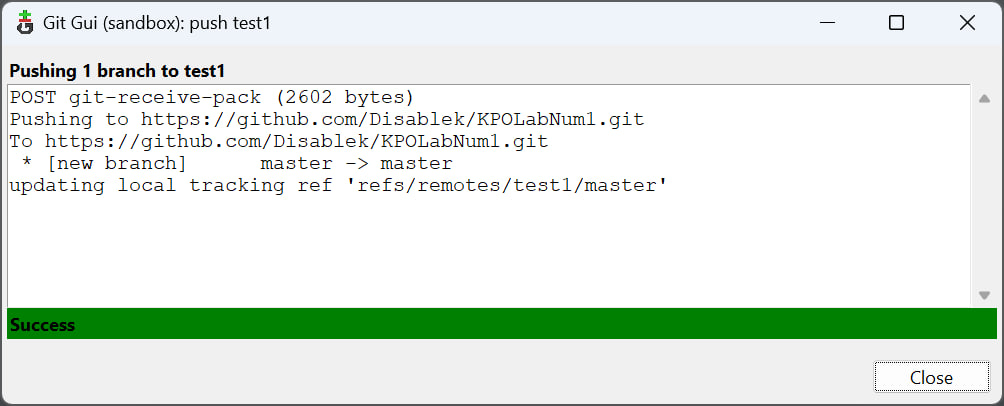


Рисунок 18 – Подтверждение отправки

В этом разделе мы зарегистрировались на сайте GitHub, создали новый публичный репозиторий, указали его название и получили адрес репозитория. Затем мы добавили удаленный репозиторий в Git Gui, указав его псевдоним и адрес. После этого отправили изменения на удаленный репозиторий, введя логин и пароль для подтверждения. Проверили успешность операции на сайте GitHub, где можно просматривать содержимое репозитория, историю изменений, а также другую информацию.

## Контрольные вопросы

## Что такое система контроля версий?

Система контроля версий – это программное обеспечение для облегчения работы с изменяющейся информацией.

1. **В каких сферах деятельности могут использоваться системы контроля**

**версий?**

В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM).

1. **Назовите наиболее популярные СКВ для разработки ПО.**

Git, Mercurial, Bazaar или Darcs

1. **Какую СКВ вы выбрали для работы и почему?**

Git представляет собой популярную систему управления версиями с активной поддержкой и открытым исходным кодом. Разработанная Линусом Торвальдсом, она обеспечивает эффективное управление версиями кода, поддерживает различные операционные системы и интегрируется с различными IDE. Git предоставляет гибкость выбора между работой через командную строку или графический интерфейс, что делает его удобным и доступным для разработчиков всех уровней опыта. Кроме того, базовая комплектация включает инструменты как для продвинутых пользователей, так и для новичков, обеспечивая широкие возможности использования.

1. **Какие виды и типы контроля версий существуют?**

Существуют три основных типа систем контроля версий: локальные, централизованные и распределённые. Каждый из них имеет свои особенности и преимущества в управлении версиями файлов и совместной работе над проектами.

1. **Что такое GIT?**

Git - это распределённая система контроля версий, разработанная Линусом Торвальдсом в 2005 году, позволяющая эффективно управлять версиями файлов и совместно работать над проектами на различных операционных системах.

1. **Знаете ли вы историю создания GIT?**

Нет.

1. **Кто является создателем GIT?**

Линус Торвальдс.

1. **Как слово “git” переводится с английского языка?**

Подлец, негодяй.

1. **Укажите основные команды, компоненты и процедуры работы в СКВ.**

Основные команды, компоненты и процедуры работы в системе контроля версий (СКВ) включают в себя команды для управления версиями файлов, такие как добавление (add), фиксация (commit) и слияние (merge), а также для работы с удаленными репозиториями, например, загрузка (push) и скачивание (pull). Компоненты включают рабочую копию (working copy), область индексирования (staging area) и репозиторий (repository). Процедуры работы включают в себя создание репозитория, добавление файлов, фиксацию изменений и обновление удаленного репозитория для совместной работы с другими разработчиками.

1. **Что такое система управления версиями?**

Система управления версиями (VCS) - это программное обеспечение, которое помогает разработчикам отслеживать изменения в файлах и коде проекта со временем.

1. **Как создать репозиторий?**

Для создания репозитория с использованием системы контроля версий, такой как Git, нужно выполнить команду "git init" в папке проекта. Это инициализирует новый локальный репозиторий, который будет отслеживать изменения в файлах этого проекта.

1. **Что такое commit, и почему Git называют системой управления коммитами?**

"Commit" в Git означает сохранение изменений в файлы и добавление этих изменений в локальный репозиторий. Коммит фиксирует определённое состояние файлов в определённый момент времени, включая все внесённые изменения. Git часто называют системой управления коммитами, потому что основной его функционал связан с процессом коммитов: добавление, отмена, просмотр и слияние коммитов. Каждый коммит в Git содержит информацию о изменениях, авторе коммита, дате и времени, что делает Git мощным инструментом для управления версиями файлов.

1. **Как создать ветку?**

Чтобы создать новую ветку в Git, используйте команду "git checkout -b <branch\_name>". Эта команда создаст новую ветку с указанным именем и автоматически переключится на неё, чтобы вы могли начать работу в новой ветке.

1. **Как провести слияние? Как разрешить конфликт и что это такое?**

Для слияния изменений из одной ветки в другую в Git используется команда "git merge <branch\_name>". Однако, если в процессе слияния возникают конфликты, то Git не может автоматически решить их, и необходимо вмешательство пользователя. Конфликт возникает, когда Git обнаруживает различия между изменениями в сливаемых ветках, которые не может разрешить самостоятельно. Для разрешения конфликтов необходимо вручную отредактировать файлы, разрешив противоречия, затем добавить изменения в индекс с помощью команды "git add", а затем завершить слияние командой "git merge --continue".

1. **Как зафиксировать изменения?**

Чтобы зафиксировать изменения в Git, используйте команду "git commit -m 'Ваше сообщение о коммите'". Эта команда фиксирует текущие изменения в вашем рабочем каталоге и добавляет их в историю версий вашего репозитория с указанным сообщением о коммите.

1. **Как провести откат? Различия в reset и revert, мягкий и жесткий reset.**

Для отката изменений в Git можно использовать команды "git reset" и "git revert". "git reset" удаляет коммиты из истории, возвращая указатель ветки к определенному коммиту, при этом изменения остаются в рабочем каталоге. Он может быть мягким (soft), сохраняя изменения в индексе, или жестким (hard), удаляя изменения из рабочего каталога. "git revert" создает новый коммит, отменяющий изменения, введенные определенным коммитом, при этом сохраняя историю изменений.

1. **Какова последовательность действий при работе с локальным репозиторием?**

Последовательность действий при работе с локальным репозиторием включает: инициализацию репозитория с помощью "git init", добавление файлов в индекс командой "git add", фиксацию изменений с помощью "git commit", работу с ветками (создание, переключение, слияние), отслеживание изменений с удаленным репозиторием (загрузка и отправка изменений с помощью "git pull" и "git push"), а также управление историей изменений (откат, просмотр и т. д.).

1. **Какова последовательность действий при работе с удаленным репозиторием?**

Последовательность действий при работе с удаленным репозиторием включает: синхронизацию локального репозитория с удаленным с помощью "git pull", выполнение необходимых изменений локально, добавление изменений в индекс с помощью "git add", фиксацию изменений с помощью "git commit", отправку изменений в удаленный репозиторий с помощью "git push", а также управление ветками и слиянием изменений между локальным и удаленным репозиториями.

1. **Каковы возможности при работе с удаленным репозиторием? Как его**

**клонировать, получать и отправлять данные?**

При работе с удаленным репозиторием можно осуществлять следующие действия: клонировать удаленный репозиторий на локальную машину с помощью команды "git clone <URL\_удаленного\_репозитория>", получать данные из удаленного репозитория с помощью команды "git pull", отправлять локальные изменения в удаленный репозиторий с помощью команды "git push". Эти возможности позволяют эффективно синхронизировать изменения между локальным и удаленным репозиториями при совместной работе над проектом.

1. **Что будет если в процессе работы изменить название папки с рабочим**

**проектом, у нас это «песочница».**

Изменение названия папки с рабочим проектом (в данном случае "песочница") не повлияет на функционирование системы контроля версий (VCS), такой как Git. Однако, если папка "песочница" уже была инициализирована как Git репозиторий (с помощью `git init`), изменение её названия может привести к тому, что Git перестанет распознавать эту папку как репозиторий. В таком случае, после переименования папки, вам нужно будет перейти в новую папку с помощью команды `cd` и инициализировать Git заново с помощью `git init`, если вы хотите сохранить историю версий вашего проекта.

**22) Почему перед в названии “.git” в папке-песочнице стоит точка, к какому**

**типу файла относится эта папка?**

Папка с названием ".git" в вашей песочнице является скрытой папкой в операционной системе. На UNIX-подобных системах (таких как Linux, macOS), имена файлов и папок, начинающиеся с точки, считаются скрытыми и по умолчанию не отображаются в обычном списке файлов. Папка ".git" содержит все внутренние данные Git-репозитория, такие как история коммитов, информация о ветках и настройки репозитория. Она не является отдельным типом файла, а представляет собой внутреннюю структуру Git-репозитория.

## Вывод:

В данной лабораторной работе мы изучили основные понятия и инструменты систем контроля версий (VCS), а также научились выполнять основные операции с локальным и удаленным репозиториями с использованием Git. Мы познакомились с понятиями коммитов, веток, слияний, а также научились проводить различные действия, такие как клонирование, получение и отправка данных в удаленный репозиторий. Понимание и использование систем контроля версий является важной частью процесса разработки программного обеспечения, позволяя эффективно управлять и отслеживать изменения в коде проекта.