Отчёт по лабораторной работе №1

Дисциплина: Математическое моделирование

Ганина Таисия Сергеевна, НФИбд-01-22

Содержание

# 1 Цель работы

Целью данной работы является приобретение навыков работы с системой контроля версий git. Работа с репозиториями, внесение изменений, индексация, коммиты.

# 2 Задание

1. Подготовить пространство для работы.
2. Создать проект.
3. Внести изменения в проект.
4. Провести индексацию изменений
5. Работа с тегами версий.
6. Отменить локальные изменения (до индексации).
7. Отменить проиндексированные изменения (перед коммитом).
8. Отменить коммит.
9. Удалить коммиты из ветки.
10. Удаление тега.
11. Внести изменения в коммиты.
12. Переместить файлы.
13. Рассмотреть Git внутри: Каталог .git.
14. Работа непосредственно с объектами git.
15. Навигация по веткам. Изменения в ветке master.
16. Слияние веток. Создание конфликта.
17. Разрешение конфликтов.
18. Сброс ветки style. Сброс ветки master.
19. Перебазирование.
20. Слияние в ветку master.
21. Клонирование репозиториев. Просмотр клонированного репозитория.
22. Что такое origin? Удаленные ветки. Изменение оригинального репозитория.
23. Слияние извлеченных изменений.
24. Извлечение общих изменений.
25. Написание отчета по лабораторной работе в формате Markdown.

# 3 Теоретическое введение

Система контроля версий (Version Control System) – это инструмент, который используется для отслеживания, внесения и управления изменениями в программном коде. Это также можно назвать просто контролем версий.

Системы контроля версий помогает разработчикам сохранять изменения, внесенные в файл, на разных этапах, чтобы и они сами, и их коллеги могли их увидеть позже.

Существует три типа систем контроля версий:

* Локальные системы контроля версий
* Централизованные системы контроля версий
* Распределенные системы контроля версий

**Что такое локальная система контроля версий (LVCS)?**

Этот тип системы контроля версий очень распространен и прост в использовании. Однако этот метод может выдавать ошибки и подвержен атакам, потому что файлы хранятся в вашей локальной системе.

Это означает, что вы можете потерять системный файл или случайно забыть каталог, с которым вы работаете (и затем записать в другой каталог). Что такое централизованная система контроля версий (CVCS)?

В этом типе контроля версий сервер работает как общее хранилище, в котором находятся все версии кода. CVCS помогает разработчикам работать совместно.

Однако, несмотря на то, что такой метод позволяет разработчикам сотрудничать, если сервер отключится на несколько секунд или будет поврежден, то есть шанс, что вы потеряете все файлы. Это является серьезной проблемой при работе с CVCS.

В CVCS только несколько разработчиков могут работать совместно над проектом. Что такое распределенная система контроля версий (DVCS)?

В настоящее время это новый и наиболее часто используемый тип системы контроля версий.

В DVCS у каждого разработчика есть полная резервная копия всех данных на сервере. Это означает, что всякий раз, когда сервер не будет работать или будет неисправен, то вы все равно сможете работать над своим проектом, а также копировать или создавать резервные копии своих хранилищ на сервере, чтобы восстановить их.

При работе с DVCS над одним проектом может работать много разработчиков. Одной из популярнейших DVCS является Git, о которой мы сейчас будем говорить подробнее.

**Что такое Git?**

Git – это бесплатная распределенная система контроля версий с открытым исходным кодом, которую можно использовать для отслеживания изменений в ваших файлах. В Git можно работать над всеми типами и размерами проектов.

С помощью Git вы можете добавлять изменения в свой код, а затем фиксировать их (или сохранять), когда это необходимо. Это означает, что у вас есть возможность вернуться к ранее внесенным изменениям. Git работает рука об руку с GitHub. А что же такое GitHub? Что такое GitHub?

GitHub – это веб-интерфейс, в котором можно хранить свои репозитории Git, а также эффективно отслеживать и управлять своими изменениями. С его помощью разные разработчики имеют доступ к коду одного проекта. У вас есть возможность вносить свои собственные изменения в проект одновременно с другими разработчиками.

Например, если вы вдруг допустили какую-то ошибку во время внесения изменений, вы можете легко вернуться к предыдущему этапу, где ошибки еще нет.

**Для чего нужно использовать GitHub?**

Есть множество причин для использования GitHub. Давайте рассмотрим некоторые из них.

**Эффективное управление проектами**

GitHub – это своего рода хранилище ваших репозиториев. GitHub позволяет разработчикам работать над одним проектом, находясь в разных местах.

С помощью GitHub вы можете легко отслеживать внесенные вами изменения и управлять ими, а также проверять ход вашей работы над проектом. Простое сотрудничество

С GitHub разработчики со всего мира могут работать вместе на одним проектом без каких-либо проблем.

Команды разработчиков могут оставаться на одной странице во время совместной работы над проектом и могут легко организовывать и эффективно управлять проектом.

**Открытый исходный код**

GitHub – это бесплатная система с открытым исходным кодом. Это означает, что разработчики могут легко получить доступ к различным типам кода/проектов, которые они могут использовать для обучения и развития своих навыков.

**Универсальность.**

Это свойство GitHub очень важно. GitHub – это веб-интерфейс не только для разработчиков. Его также могут использовать дизайнеры, писатели и все, кто хочет отслеживать историю своих проектов.

# 4 Выполнение лабораторной работы

1. Подготовить пространство для работы.

Я уже ранее использовала гит, поэтому сейчас выполнила несколько команд для того, чтобы вывести на экран имя и электронную почту, а после поработала с параметрами установки окончаний строк и с установкой отображения unicode. (рис. [1](#fig:001)).

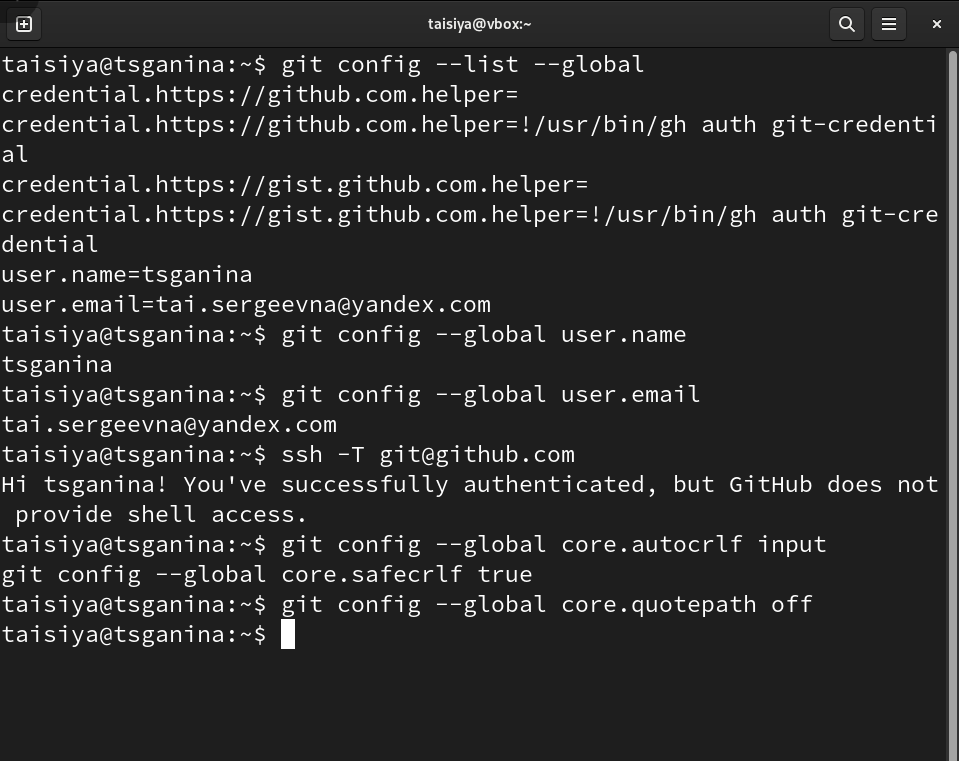


Figure 1: Подготовить пространство для работы

1. Создание проекта.

Я начала работу в пустом рабочем каталоге с создания пустого каталога с именем hello, затем вошла в него и создала там файл с именем hello.html.

Потом я создала git репозиторий из этого каталога. Добавила файл в репозиторий, проверила его состояние. (рис. [2](#fig:002), [3](#fig:003), [4](#fig:004), [5](#fig:005)).

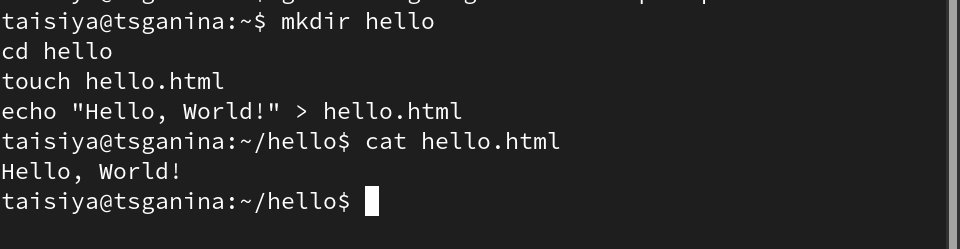


Figure 2: Создание проекта

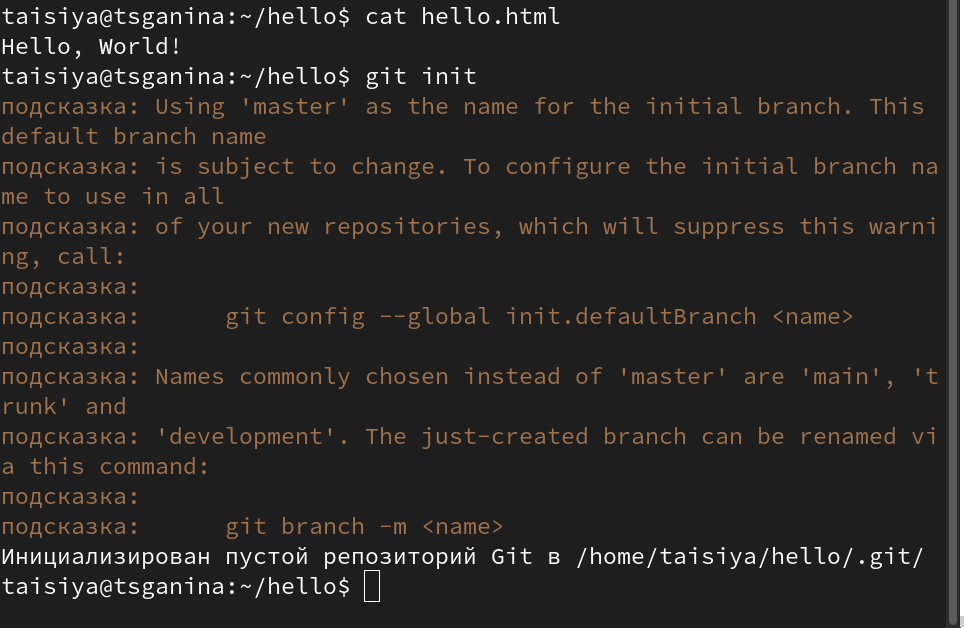


Figure 3: Создать git репозиторий из этого каталога

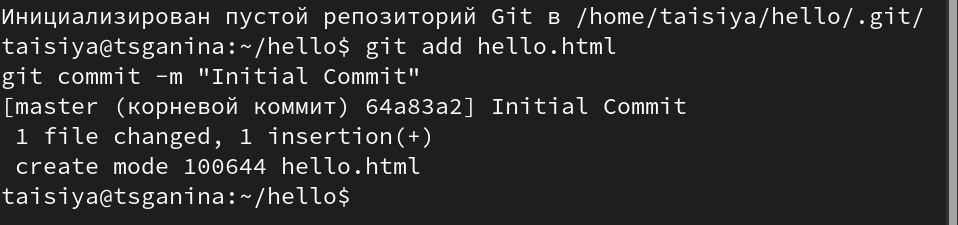


Figure 4: Добавим файл в репозиторий

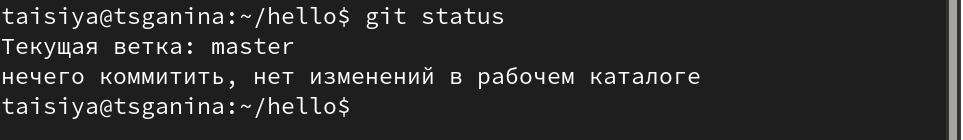


Figure 5: Команда git status, чтобы проверить текущее состояние репозитория

1. Внесение изменений в проект.

Я добавила HTML-теги к нашему приветствию. Проверила состояние рабочего каталога, видно, что git знает, что файл hello.html был изменен, но при этом эти изменения еще не зафиксированы в репозитории. (рис. [6](#fig:006)).

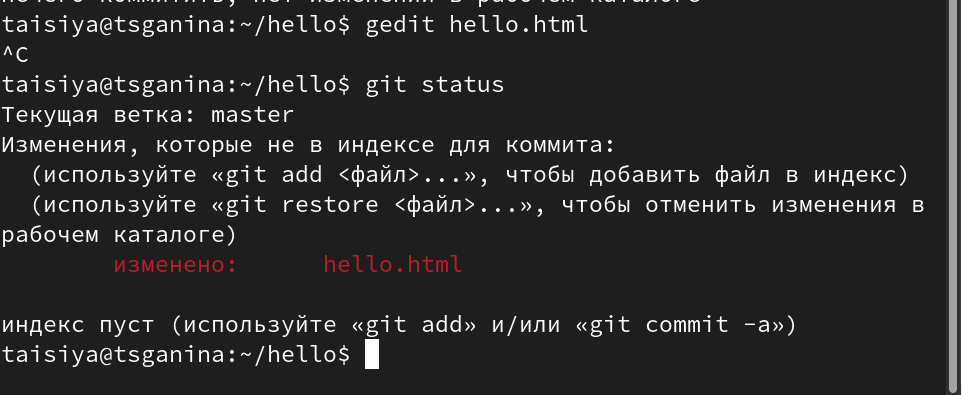


Figure 6: Изменила файл и проверила состояние каталога

1. Провести индексацию изменений. Выполнила следующие команды, чтобы проиндексировать изменения и опять проверить состояние:

git add hello.html  
git status

(рис. [7](#fig:007)).

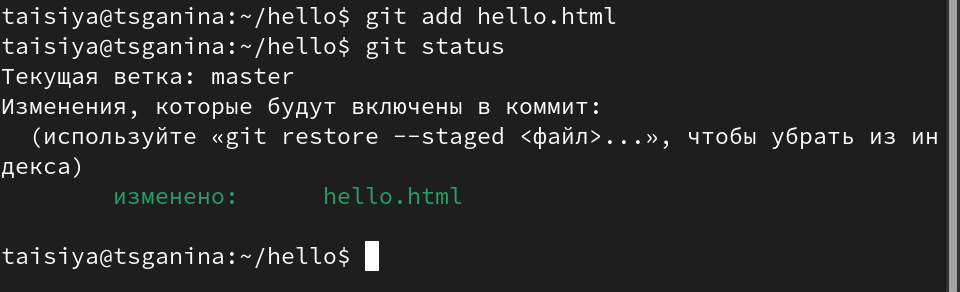


Figure 7: Провести индексацию изменений

После этого я сделала коммит, но не писала метку -m в строке, поэтому в редакторе смогла интерактивно отредактировать комментарии для коммита. (рис. [8](#fig:008), [9](#fig:009)).

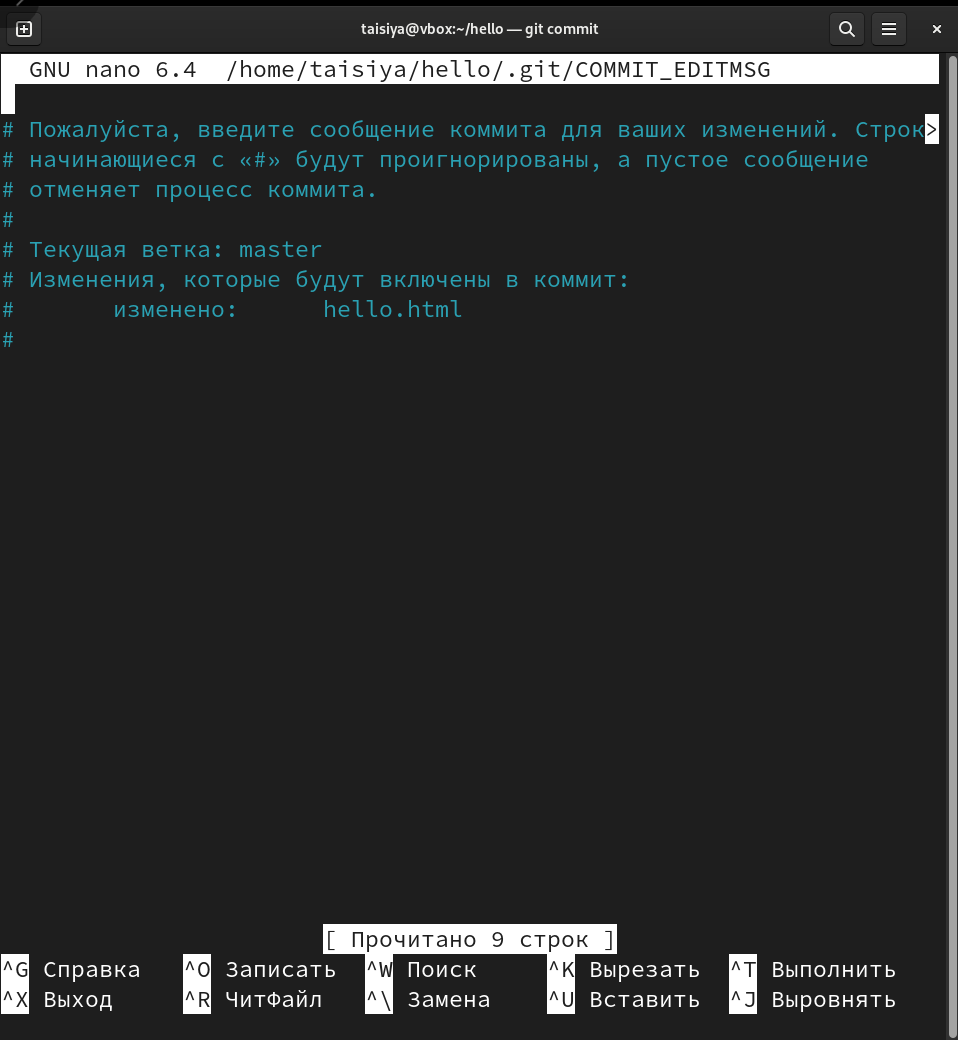


Figure 8: Редактор коммита

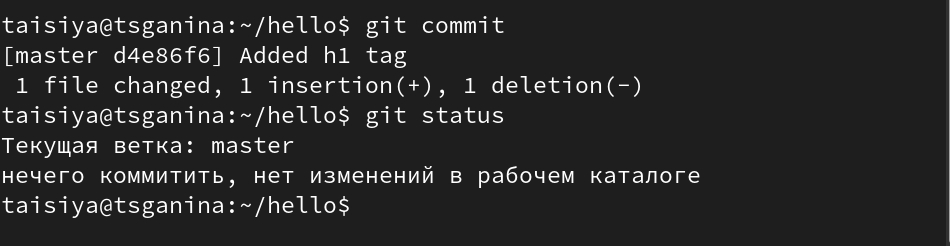


Figure 9: Запись коммита, проверка состояния

Потом я опять изменила файл html и добавила туда стандартные теги страницы. При этом часть изменений я проиндексировала, а часть (заголовок) – нет. Произвела коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверила состояние. Потом последовательно проиндексировала второе изменение и закоммитила. (рис. [10](#fig:010), [11](#fig:011), [12](#fig:012), [13](#fig:013), [14](#fig:014)).

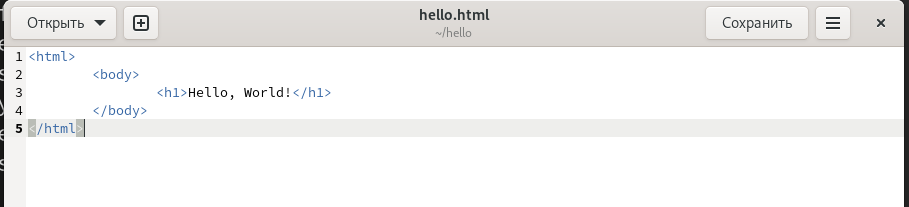


Figure 10: Код файла



Figure 11: Продолжение кода

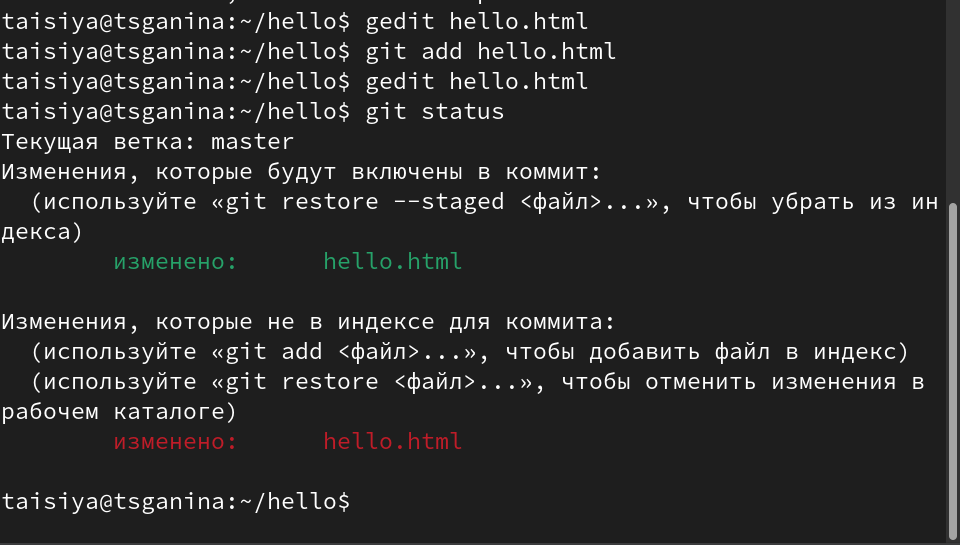


Figure 12: Одно изменение проиндексировано, второе нет

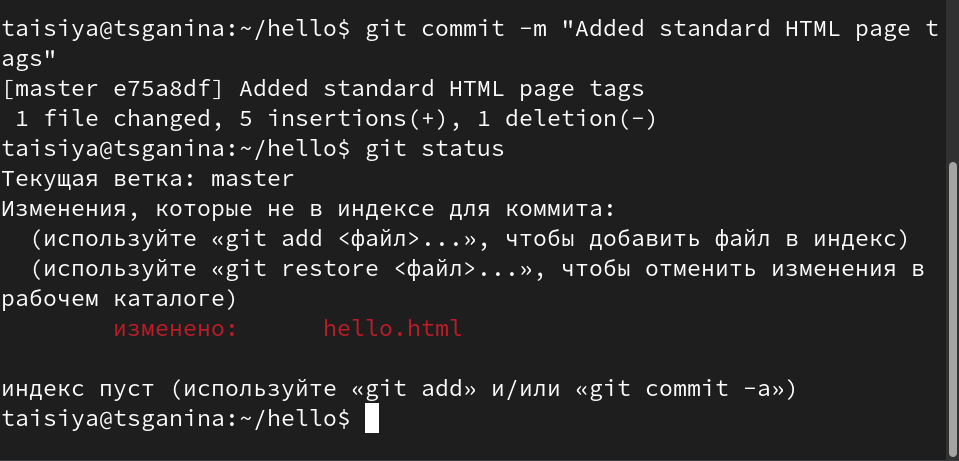


Figure 13: Закоммитила проиндексированное

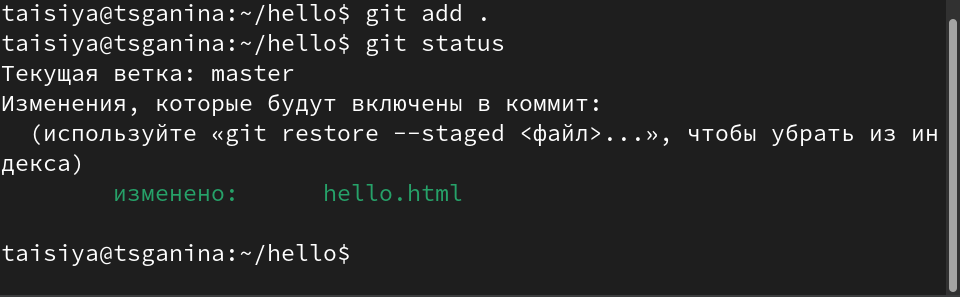


Figure 14: Проиндексировала второе изменение

Далее была работа с историей и просмотр версий. (рис. [15](#fig:015), [16](#fig:016)).

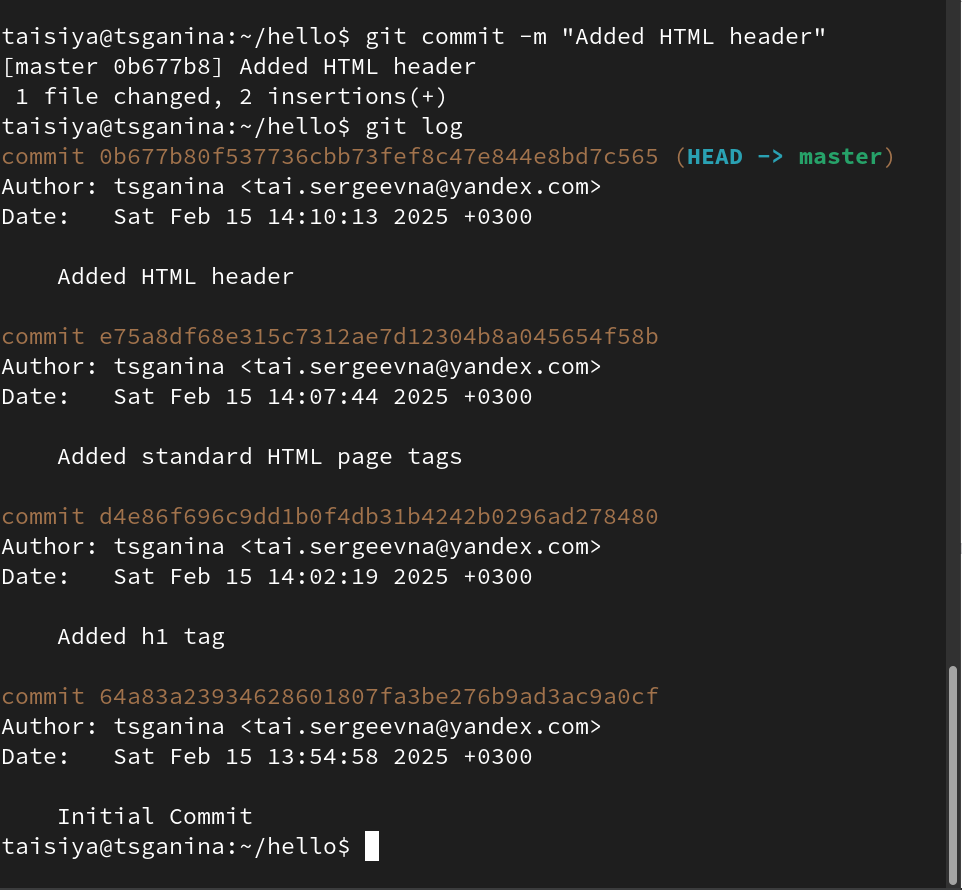


Figure 15: git log

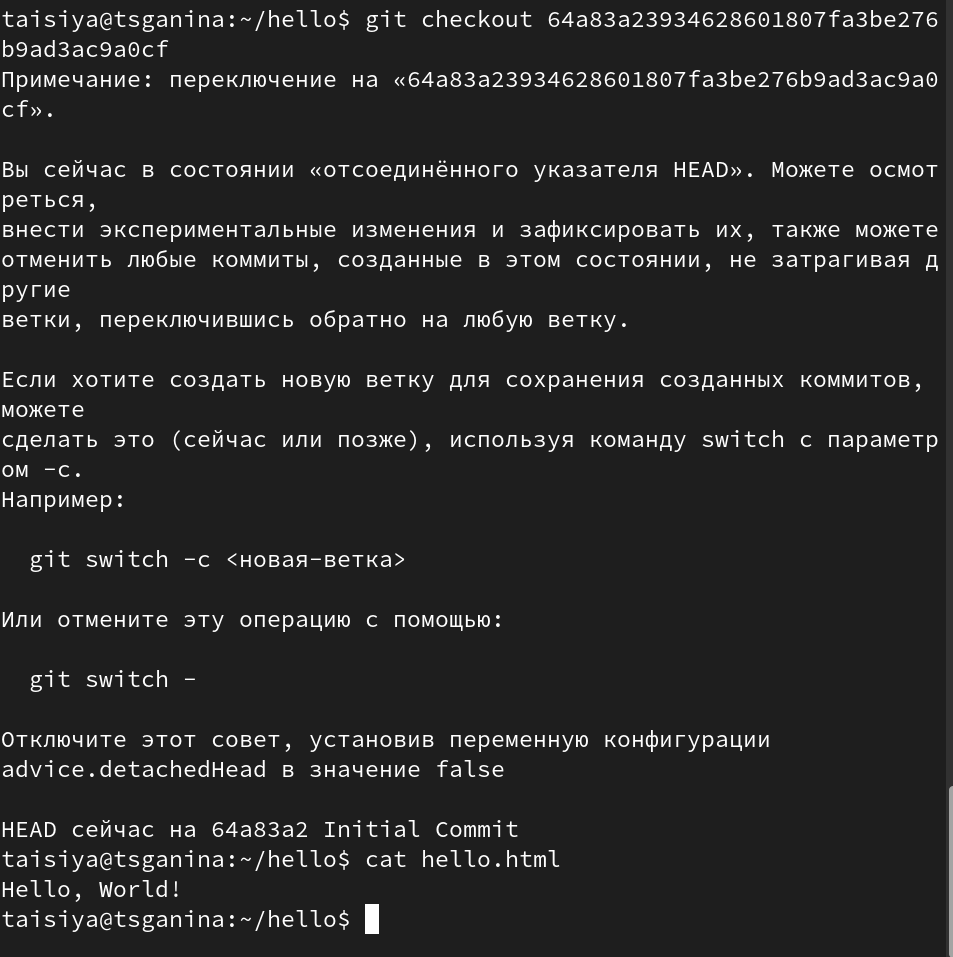


Figure 16: Первый коммит

1. Работа с тегами версий. Вместо поиска до хэш, использовала ^, обозначающее «родитель v1». Вместо обозначения v1^ можно использовать v1~1. Это обозначение можно определить как «первую версию предшествующую v1». Создала тег первой версии и также тег для предыдущей версии при помощи команд:

git tag v1  
git checkout v1^  
cat hello.html  
git tag v1-beta

Потом переключалась между двумя отмеченными версиями и смотрела теги в логе. (рис. [17](#fig:017), [18](#fig:018), [19](#fig:019)).

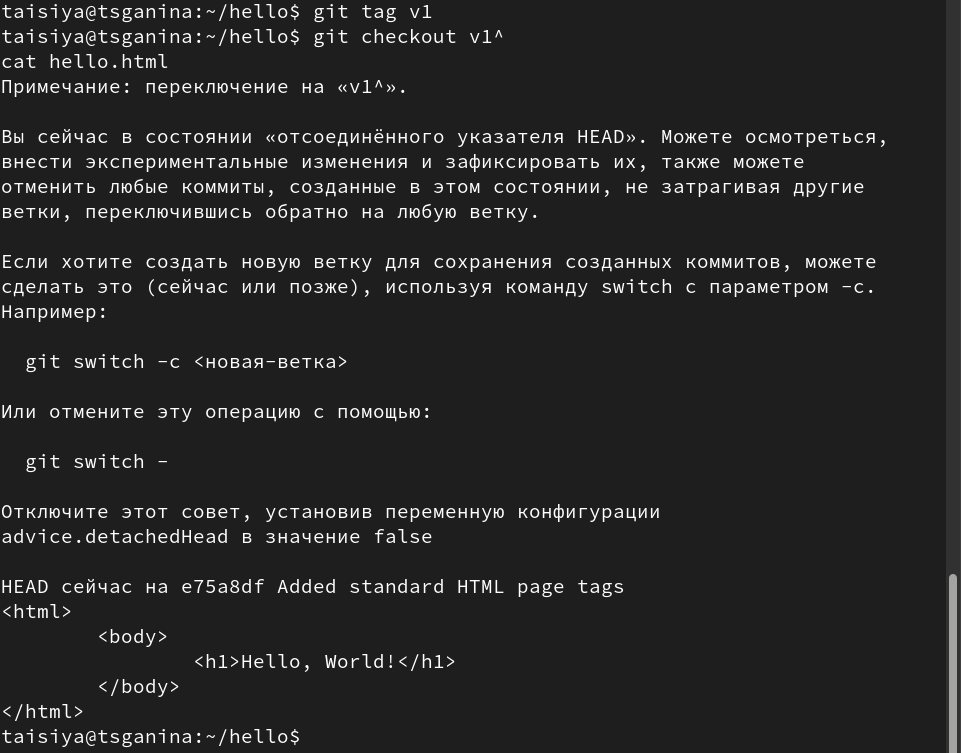


Figure 17: git tag v1

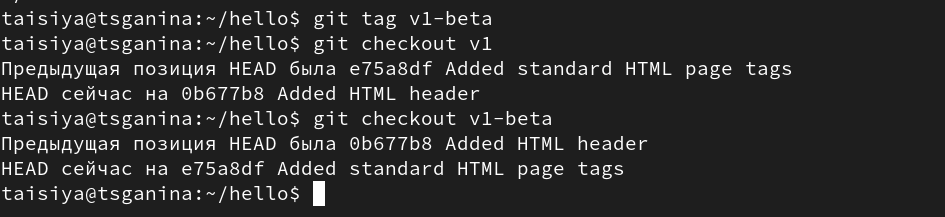


Figure 18: git tag v1-beta

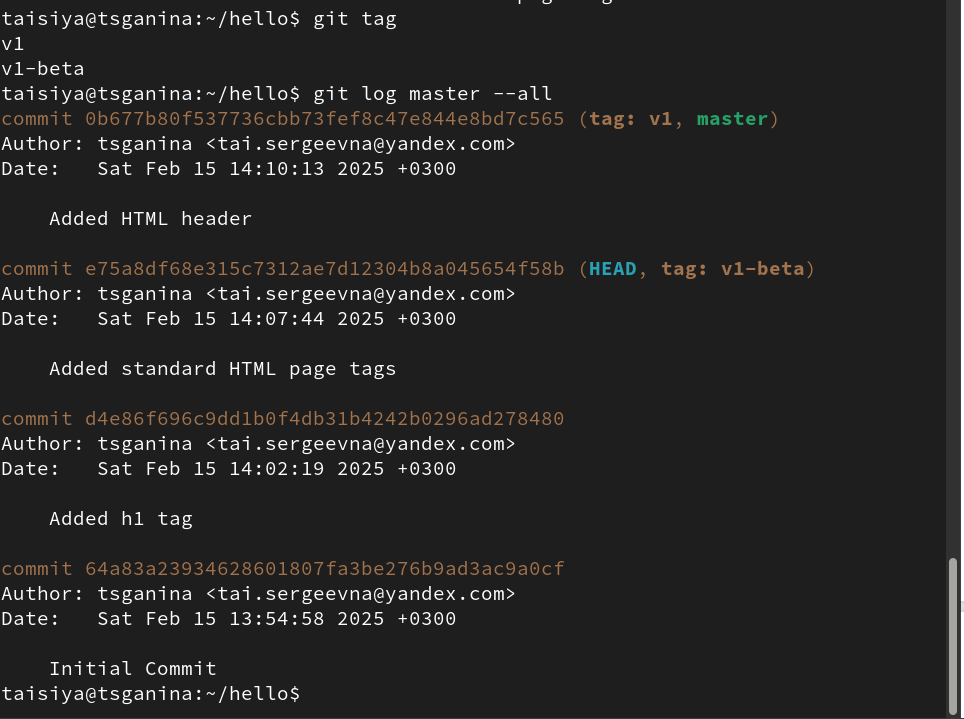


Figure 19: git log master –all

1. Отменить локальные изменения (до индексации). Убедилась, что нахожусь на последнем коммите ветки master, прежде чем продолжить работу. Потом внесла нежелательный комментарий, но не индексировала изменения и отменила их при помощи git checkout (рис. [20](#fig:020))

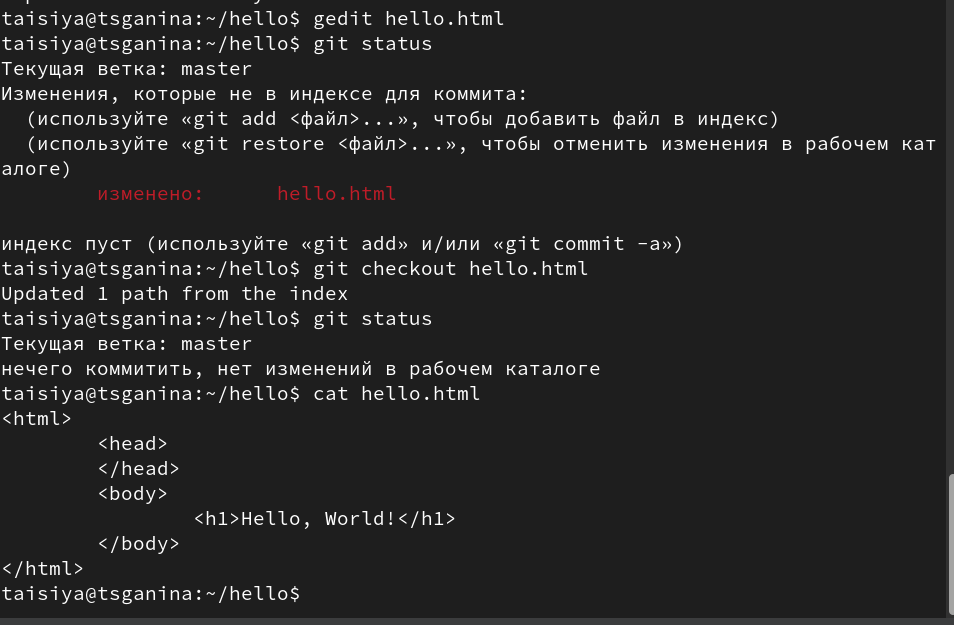


Figure 20: Отменить локальные изменения (до индексации)

1. Отменить проиндексированные изменения (перед коммитом). Я опять внесла нежелательные изменения в файл, но теперь уже проиндексировала их, а после – отменила. (рис. [21](#fig:021)).

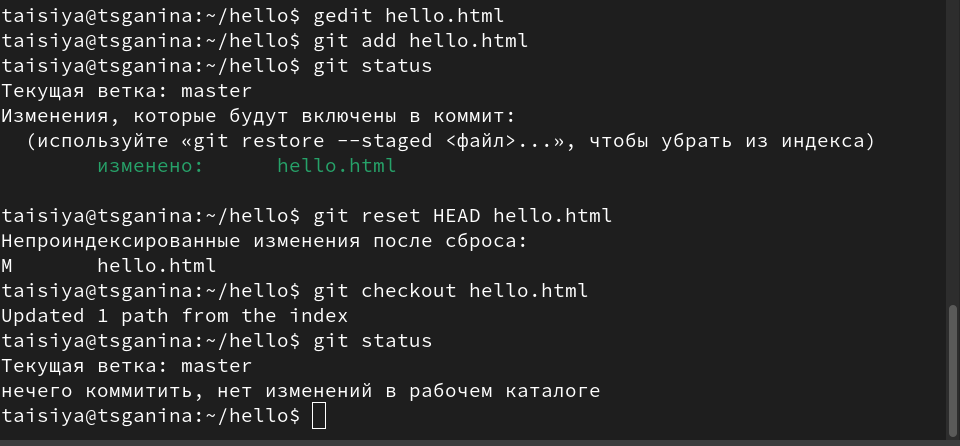


Figure 21: Отменить проиндексированные изменения (перед коммитом)

1. Отменить коммит. (рис. [22](#fig:022), [23](#fig:023), [24](#fig:024)).

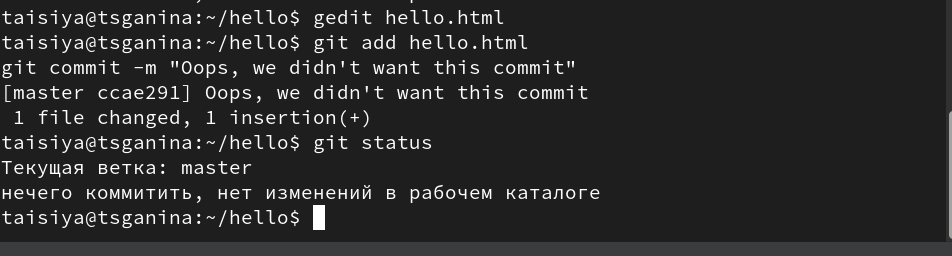


Figure 22: Делаем коммит

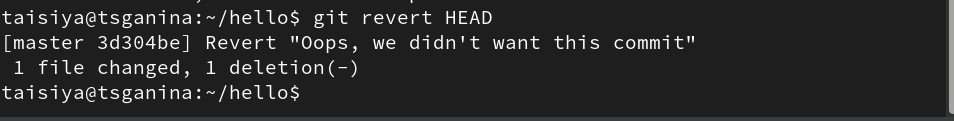


Figure 23: git revert HEAD

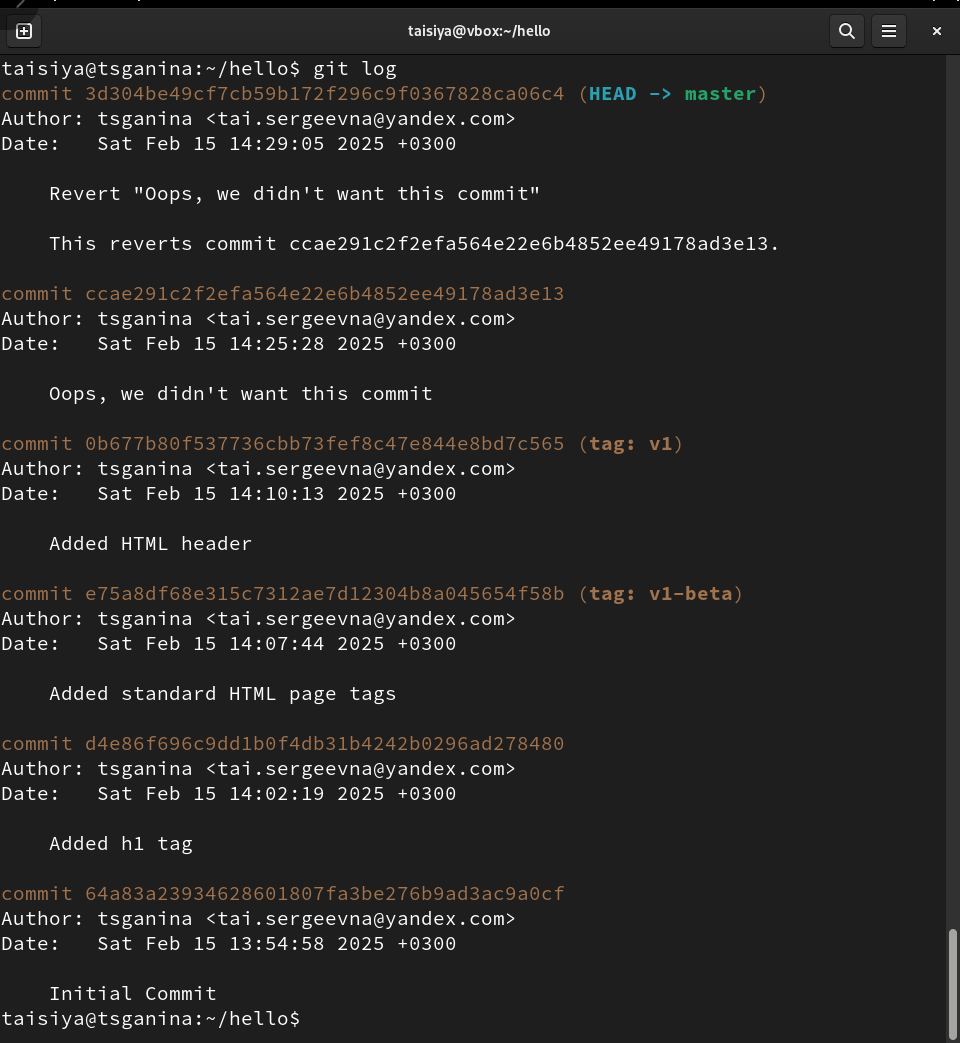


Figure 24: git log

1. Удалить коммиты из ветки. Часто мы делаем коммит, и сразу понимаем, что это была ошибка. Было бы неплохо иметь команду «возврата», которая позволила бы нам сделать вид, что неправильного коммита никогда и не было. Команда «возврата» даже предотвратила бы появление нежелательного коммита в истории git log. (рис. [25](#fig:025), [26](#fig:026))

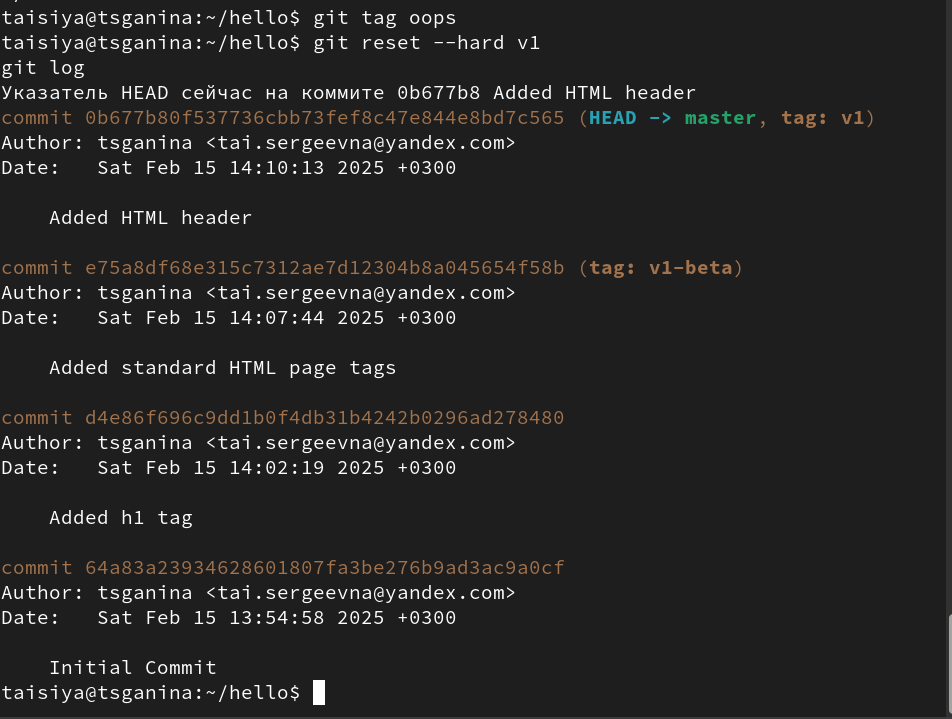


Figure 25: Давайте отметим последний коммит тегом, и выполним сброс коммитов к предшествующим коммиту oops

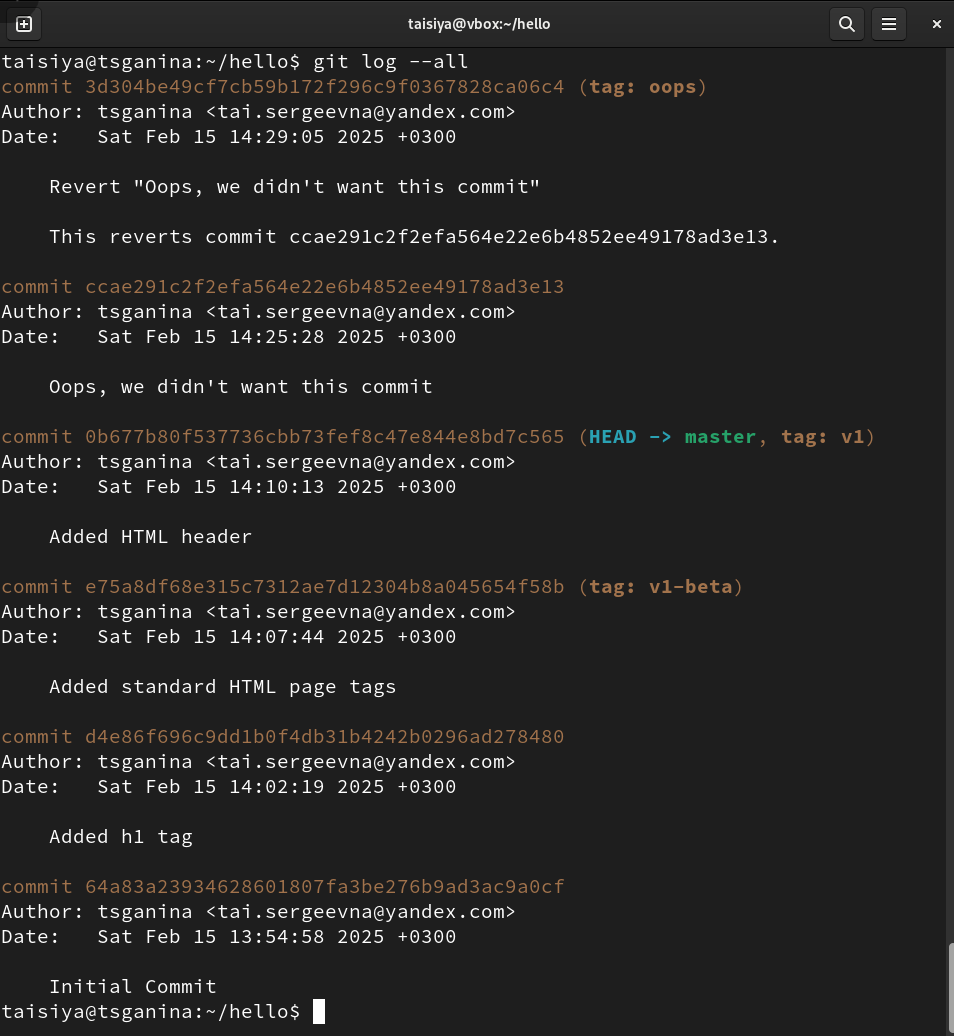


Figure 26: Оказывается, что коммиты все еще находятся в репозитории

1. Удаление тега. Сброс в локальных ветках, как правило, безопасен. Последствия любой «аварии» как правило, можно восстановить простым сбросом с помощью нужного коммита. Однако, если ветка «расшарена» на удаленных репозиториях, сброс может сбить с толку других пользователей ветки. (рис. [27](#fig:027)).

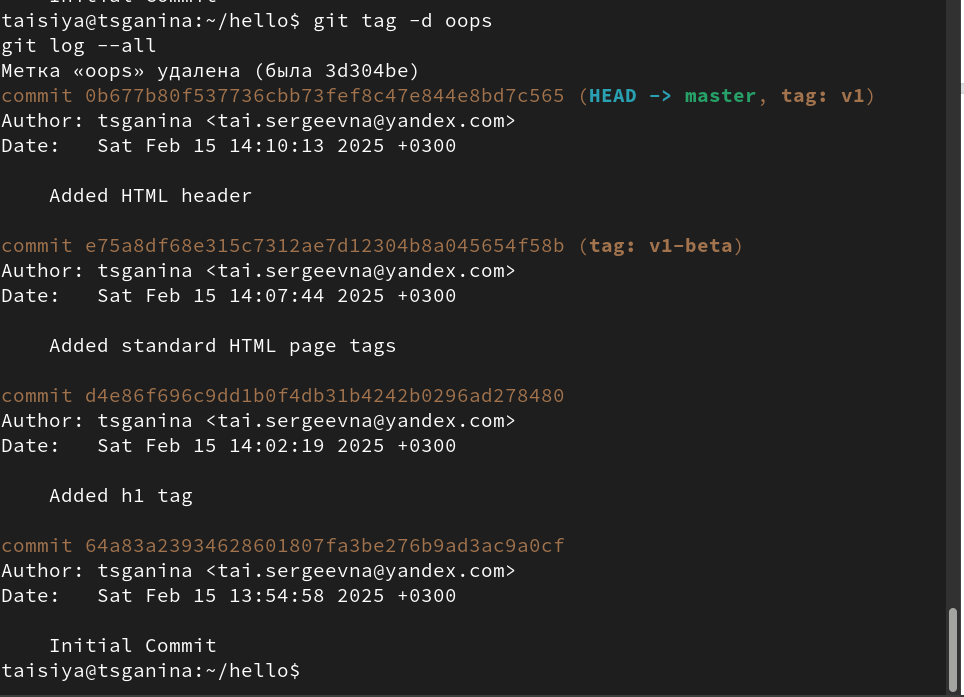


Figure 27: Удаление тега oops

1. Внести изменения в коммиты. Добавим в страницу комментарий автора. После совершения коммита мы понимаем, что любой хороший комментарий должен включать электронную почту автора. Обновим страницу hello, включив в нее email. Давайте изменим предыдущий коммит, включив в него адрес электронной почты. (рис. [28](#fig:028), [29](#fig:029))



Figure 28: Добавим в страницу комментарий автора

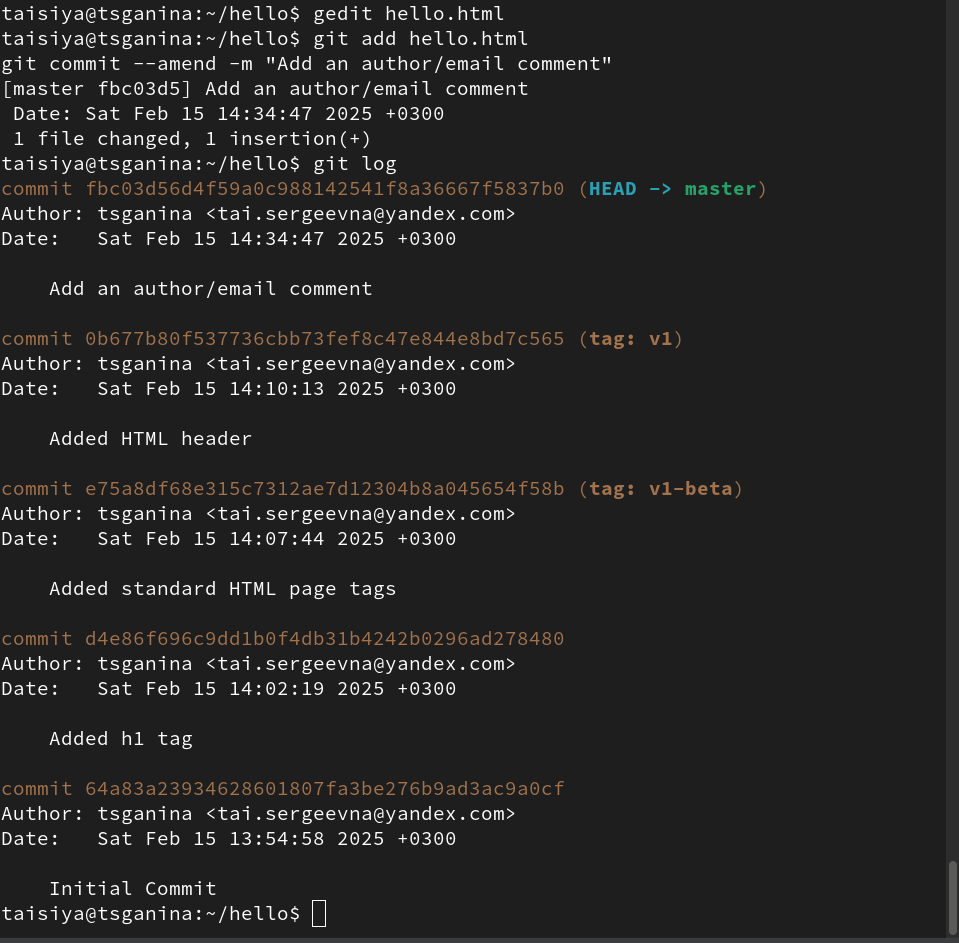


Figure 29: Изменим предыдущий коммит

1. Переместить файлы. Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесем страницу в каталог lib, сделаем коммит этого перемещения. Добавим файл index.html в наш репозиторий. Теперь при открытии index.html, мы должны увидеть кусок страницы hello в маленьком окошке. (рис. [30](#fig:030), [31](#fig:031), [32](#fig:032))

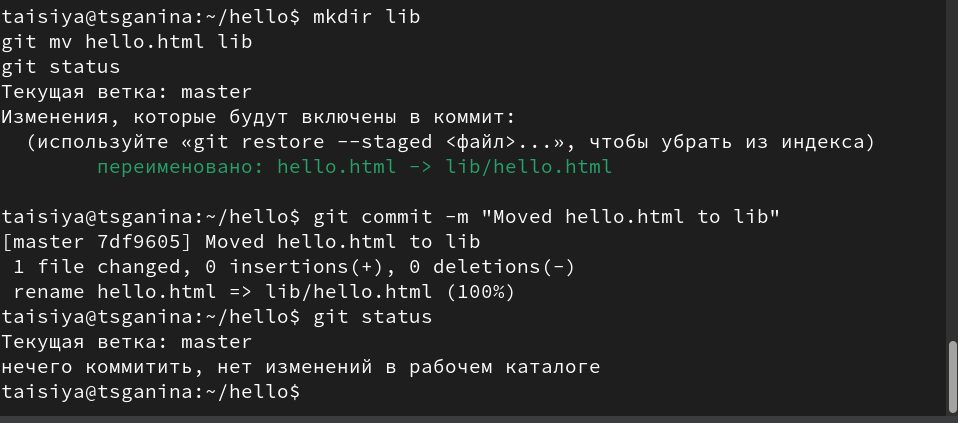


Figure 30: Структура нашего репозитория

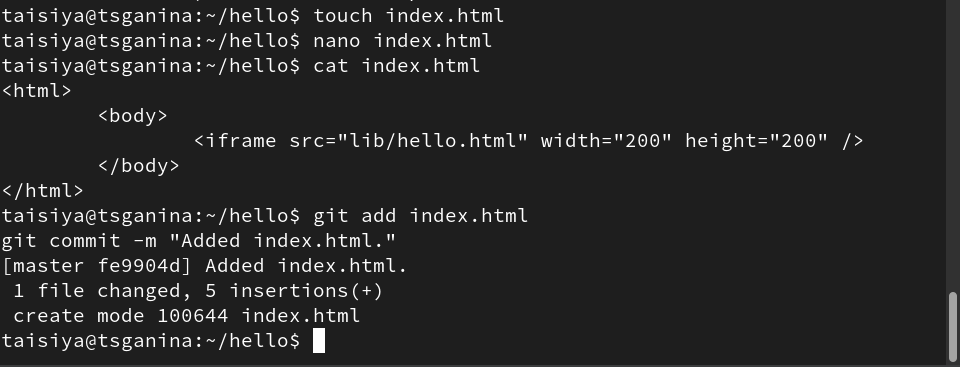


Figure 31: Добавим файл index.html

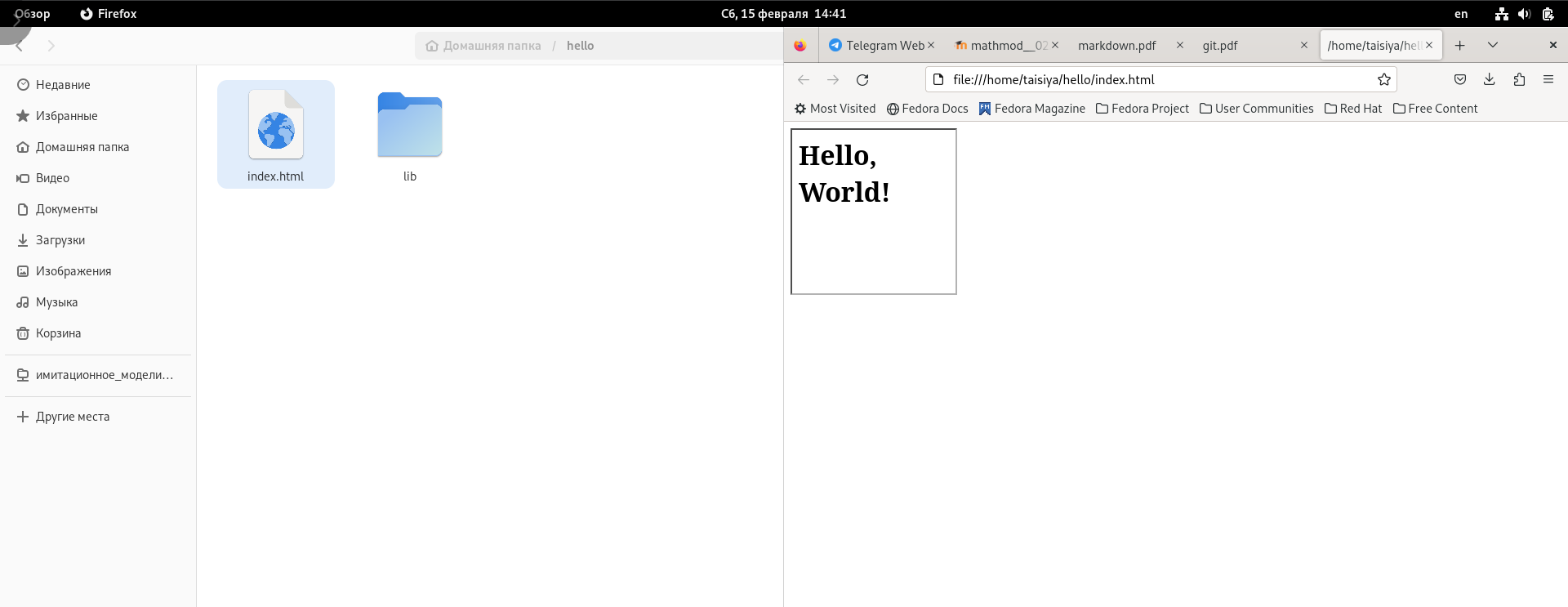


Figure 32: Кусок страницы hello в маленьком окошке

1. Рассмотреть Git внутри: Каталог .git. Посмотрим каталог, в котором хранится вся информация git.

Выполним:

ls -C .git/objects

Мы должны увидеть набор каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git. После углубимся в базу данных объектов, посмотрим файл конфигурации, создающийся для каждого конкретного проекта.

Выполним:

ls .git/refs  
ls .git/refs/heads  
ls .git/refs/tags  
cat .git/refs/tags/v1

Мы узнаём файлы в подкаталоге тегов. Каждый файл соответствует тегу, ранее созданному с помощью команды git tag. Его содержание — это всего лишь хэш коммита, привязанный к тегу. Посмотрим файл HEAD содержит ссылку на текущую ветку, в данный момент это должна быть ветка master. (рис. [33](#fig:033), [34](#fig:034), [35](#fig:035)).

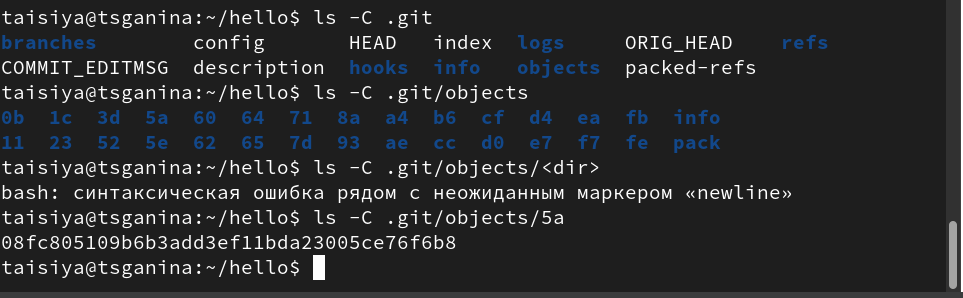


Figure 33: Набор каталогов, имена которых состоят из 2 символов

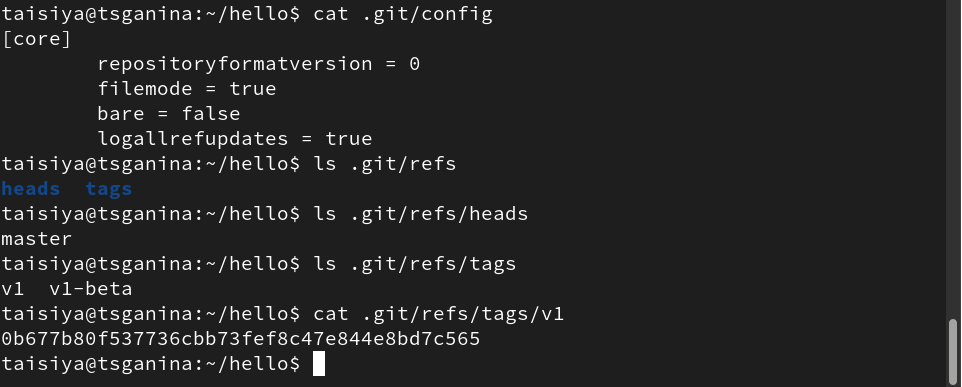


Figure 34: Посмотрим файл конфигурации

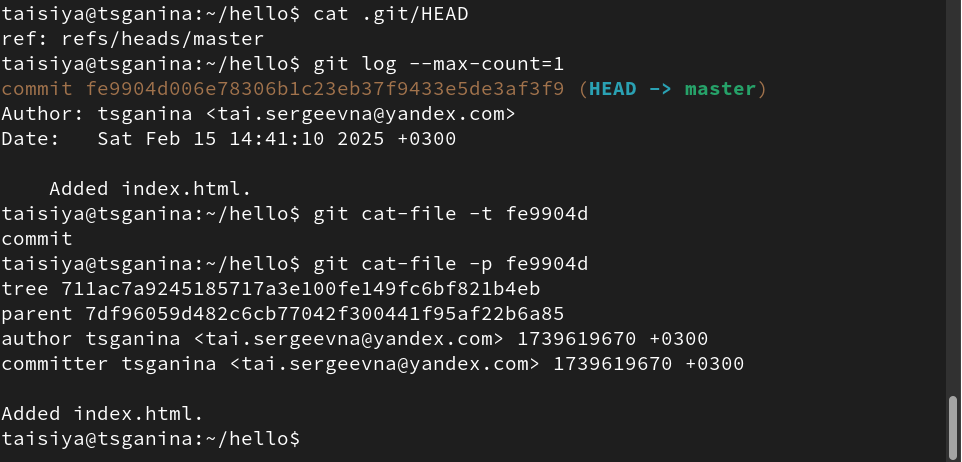


Figure 35: Файл HEAD

1. Работа непосредственно с объектами git. Посмотрим последний коммит в репозиторий. SHA1 хэш в нашей системе. Мы можем вывести дерево каталогов, ссылка на который идет в коммите. Это должно быть описание файлов (верхнего уровня) в нашем проекте (для конкретного коммита). Вывод каталога lib, вывод файла hello.html.(рис.@fig:036)

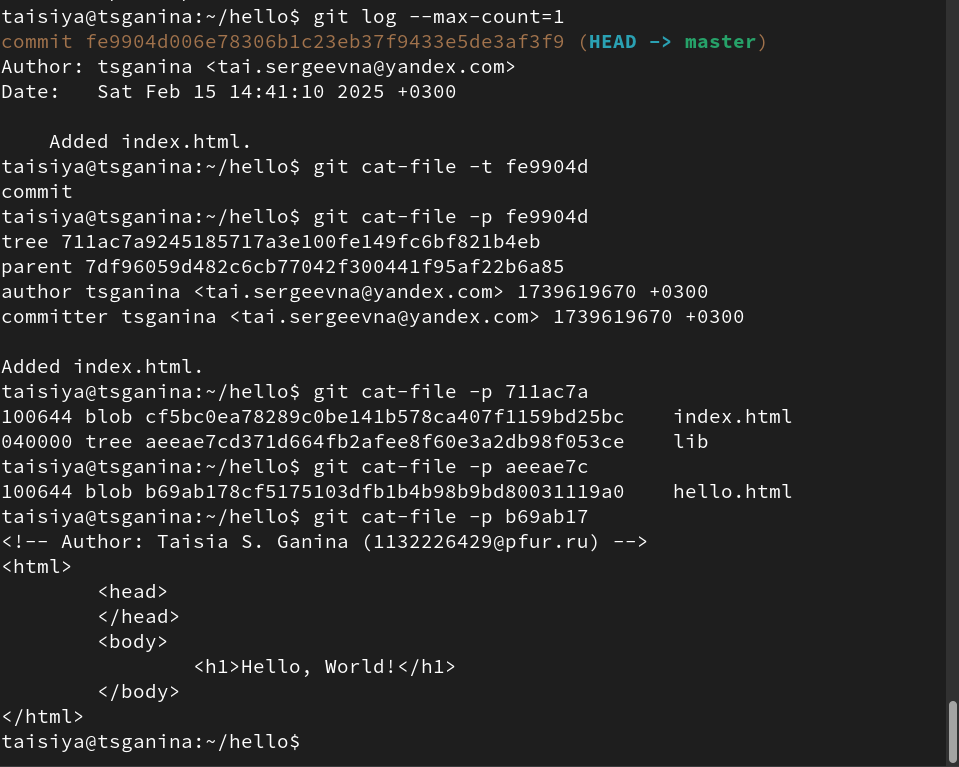


Figure 36: Работа непосредственно с объектами git

**Самостоятельное исследование.** Исследуем git репозиторий вручную самостоятельно. Смотрим, удастся ли нам найти оригинальный файл hello.html с самого первого коммита вручную по ссылкам SHA1 хэша в последнем коммите. (рис. [37](#fig:037), [38](#fig:038))

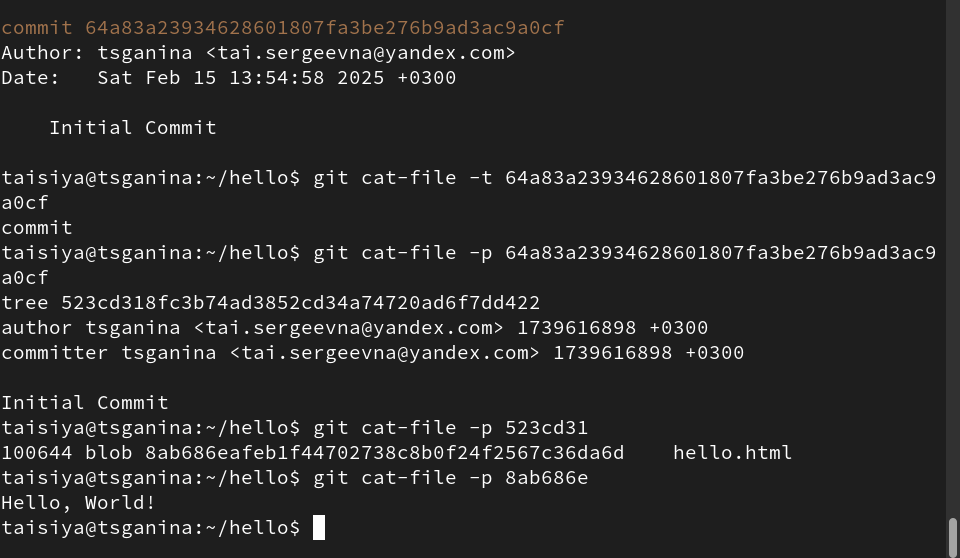


Figure 37: Самостоятельное исследование 1

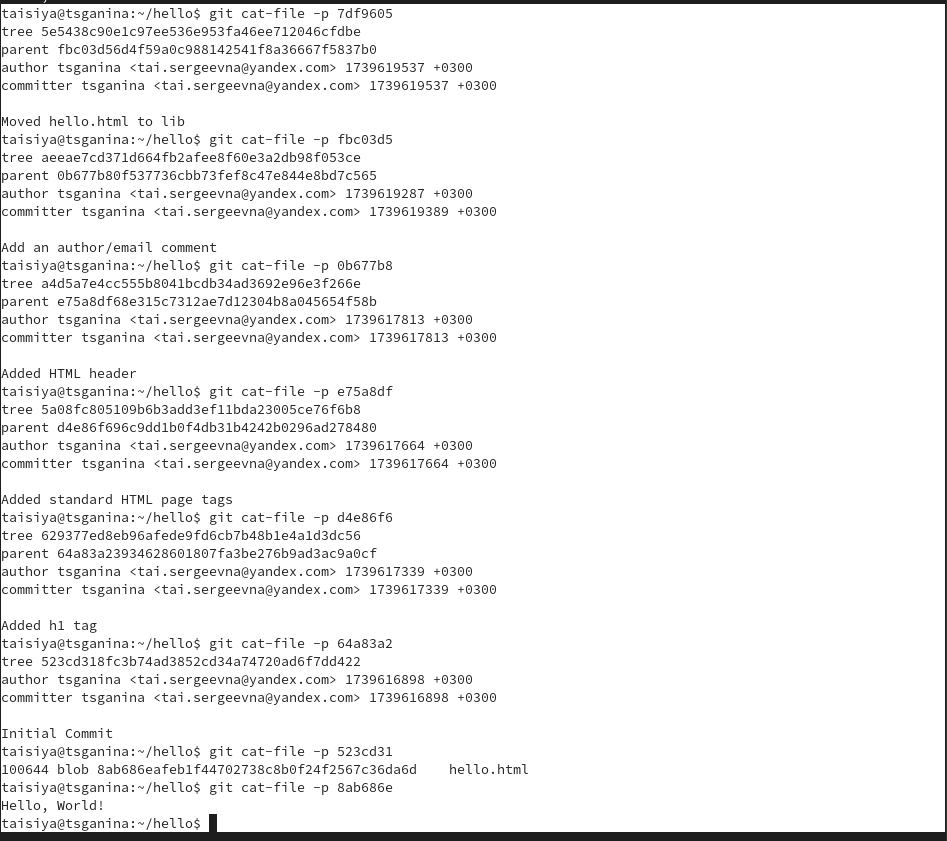


Figure 38: Самостоятельное исследование 2

1. Навигация по веткам. Изменения в ветке master.

Пора сделать наш hello world более выразительным. Так как это может занять некоторое время, лучше переместить эти изменения в отдельную ветку, чтобы изолировать их от изменений в ветке master. Давайте назовем нашу новую ветку «style». Добавим файл стилей style.css. Обновим файл hello.html, чтобы использовать стили style.css. Обновим файл index.html, чтобы он тоже использовал style.css. (рис. [39](#fig:039), [40](#fig:040)).

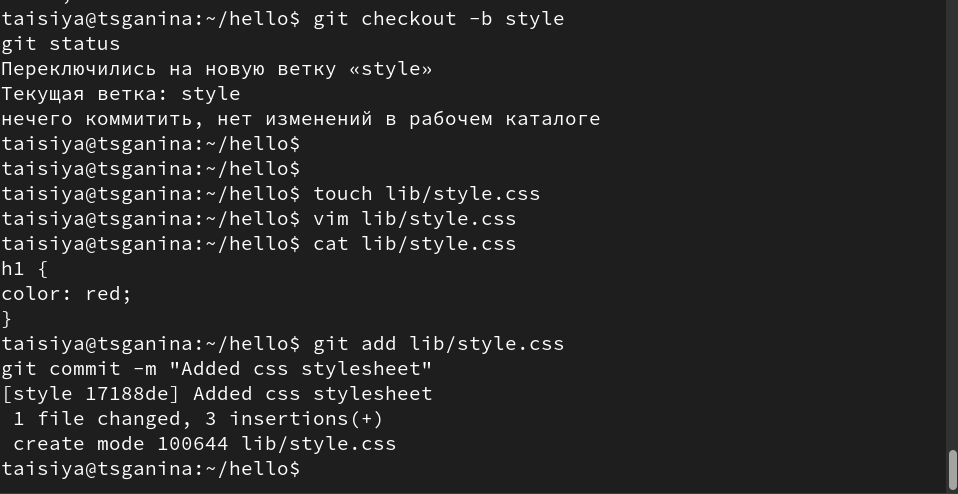


Figure 39: Создайте ветку, файл стилей style.css

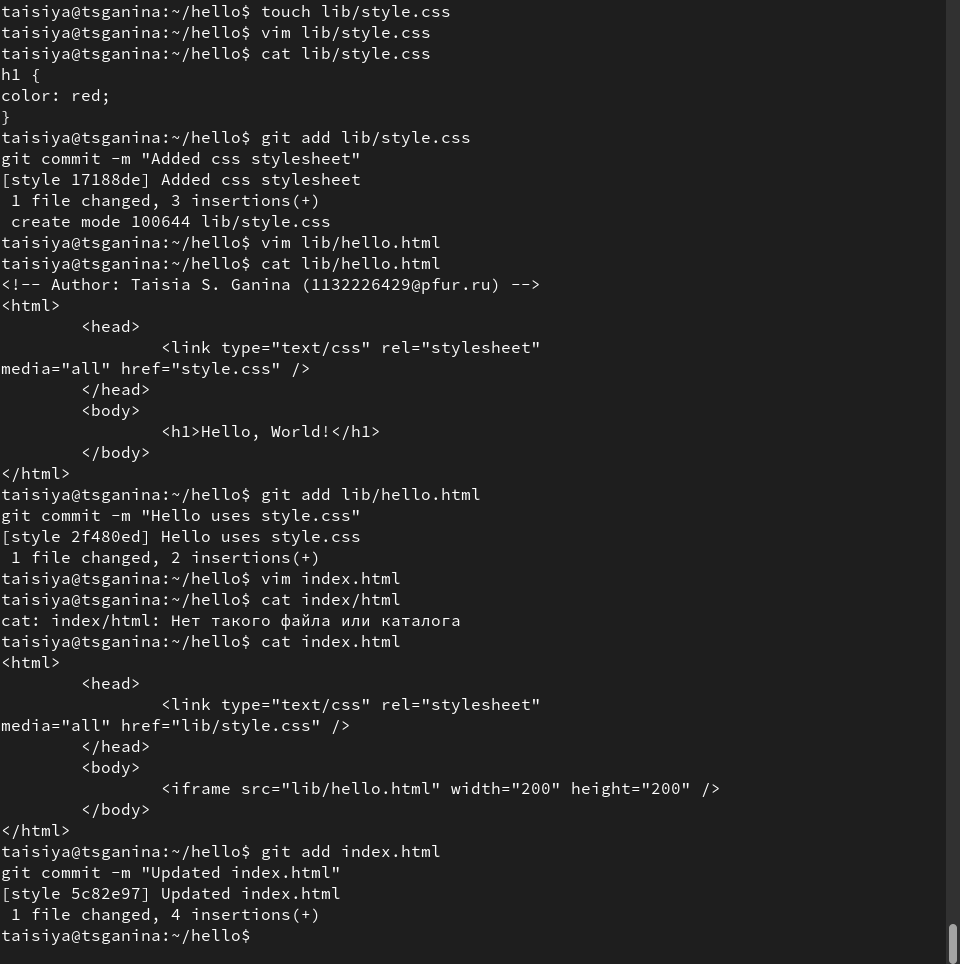


Figure 40: Обновим файл hello.html. Обновим файл index.html

Потом используем команду git checkout для переключения между ветками. (рис. [41](#fig:041)).

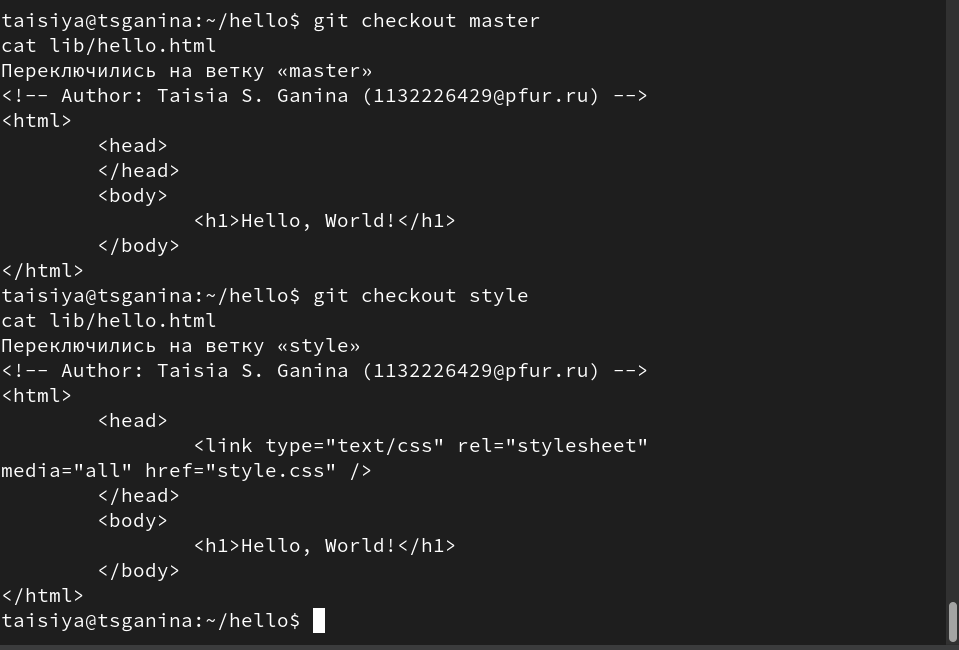


Figure 41: Переключения между ветками

Создадим файл README в ветке master. Сделаем коммит изменений README.md в ветку master. Просмотр отличающихся веток. (рис. [42](#fig:042)).

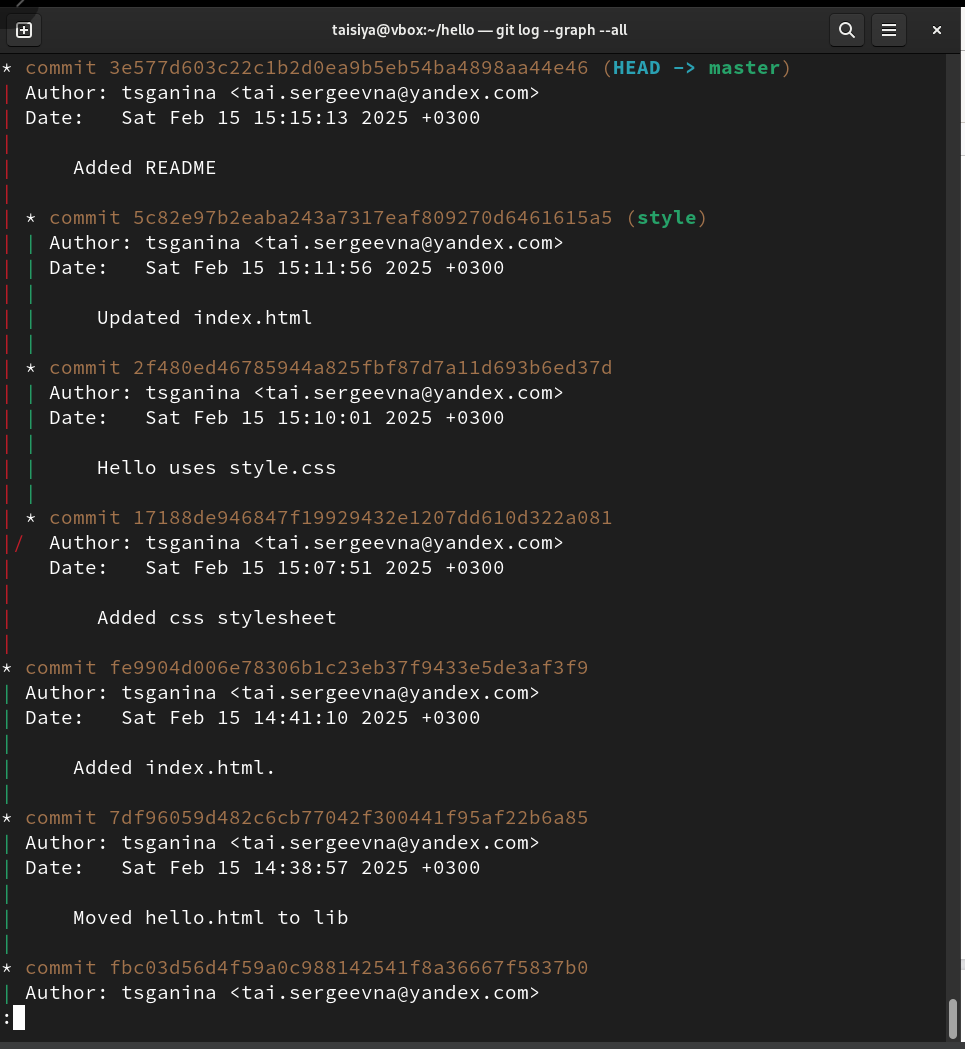


Figure 42: git log –graph –all

1. Слияние веток. Создание конфликта.

Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сольем master с style. Путем периодического слияния ветки master с веткой style мы можем переносить из master любые изменения и поддерживать совместимость изменений style с изменениями в основной ветке. Но что если изменения в ветке master конфликтуют с изменениями в style? (рис. [43](#fig:043), [44](#fig:044), [45](#fig:045)).

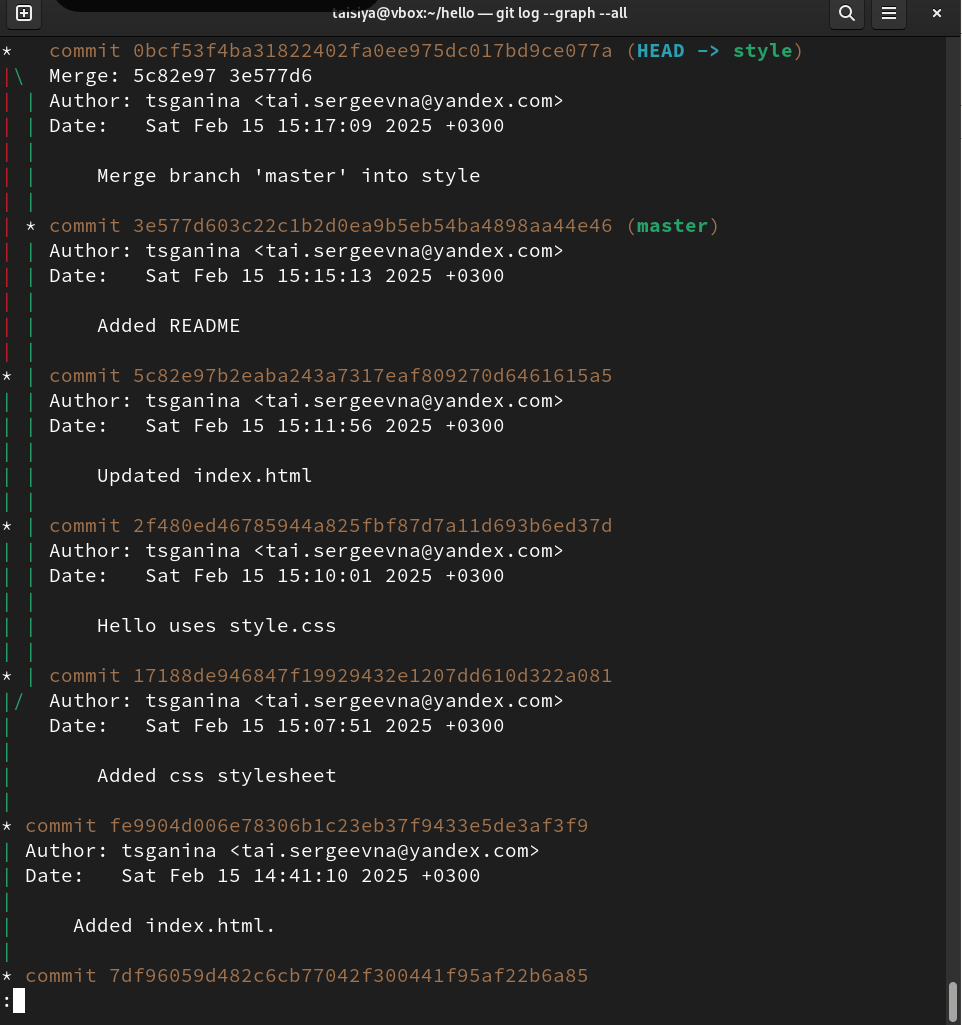


Figure 43: Слияние веток

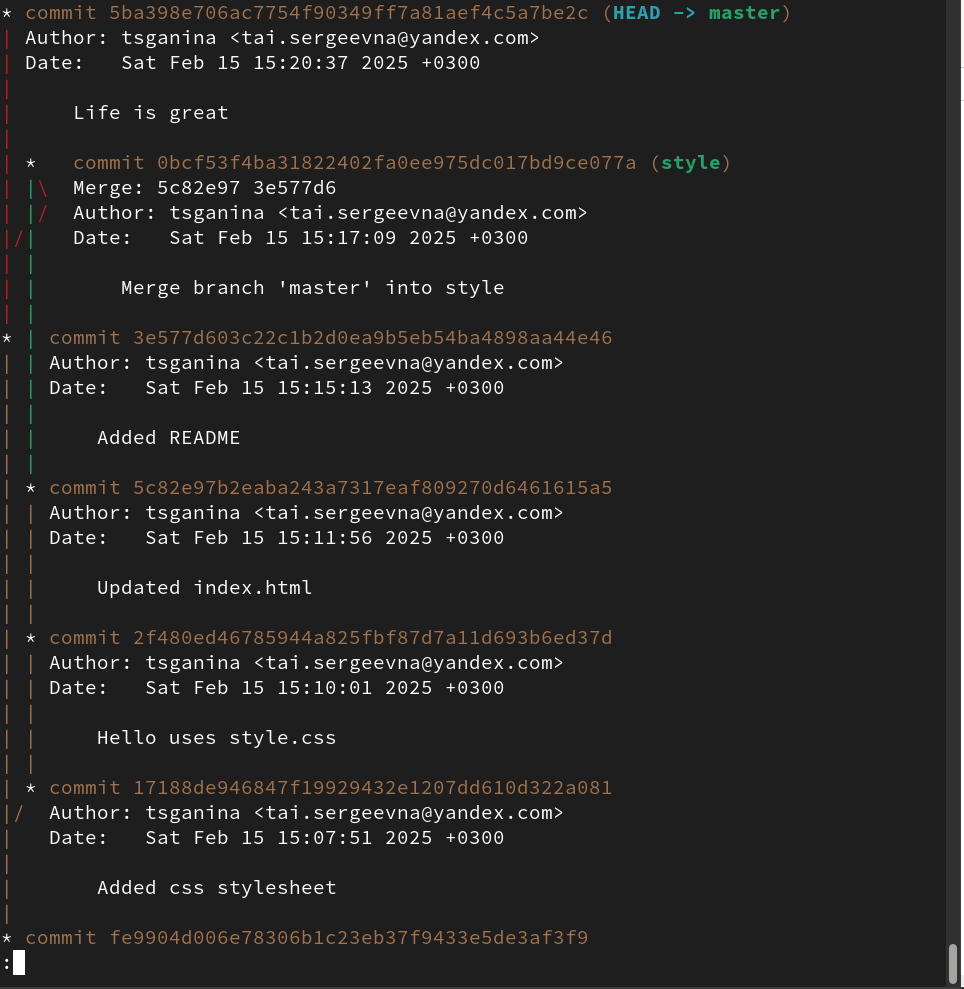


Figure 44: Создание конфликта

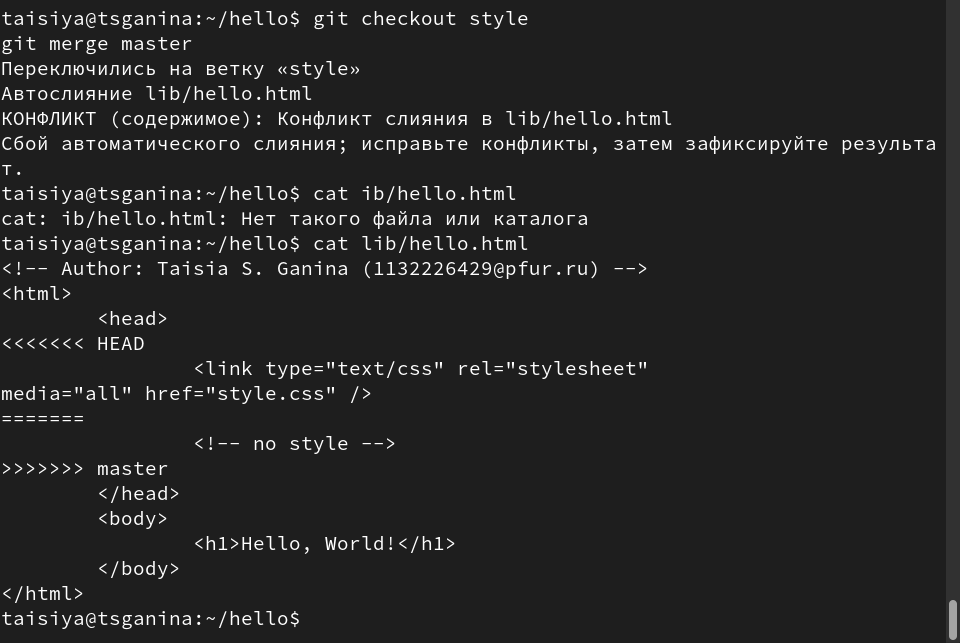


Figure 45: Вернемся к ветке style и попытаемся объединить ее с новой веткой master

1. Разрешение конфликтов. (рис. [46](#fig:046)).

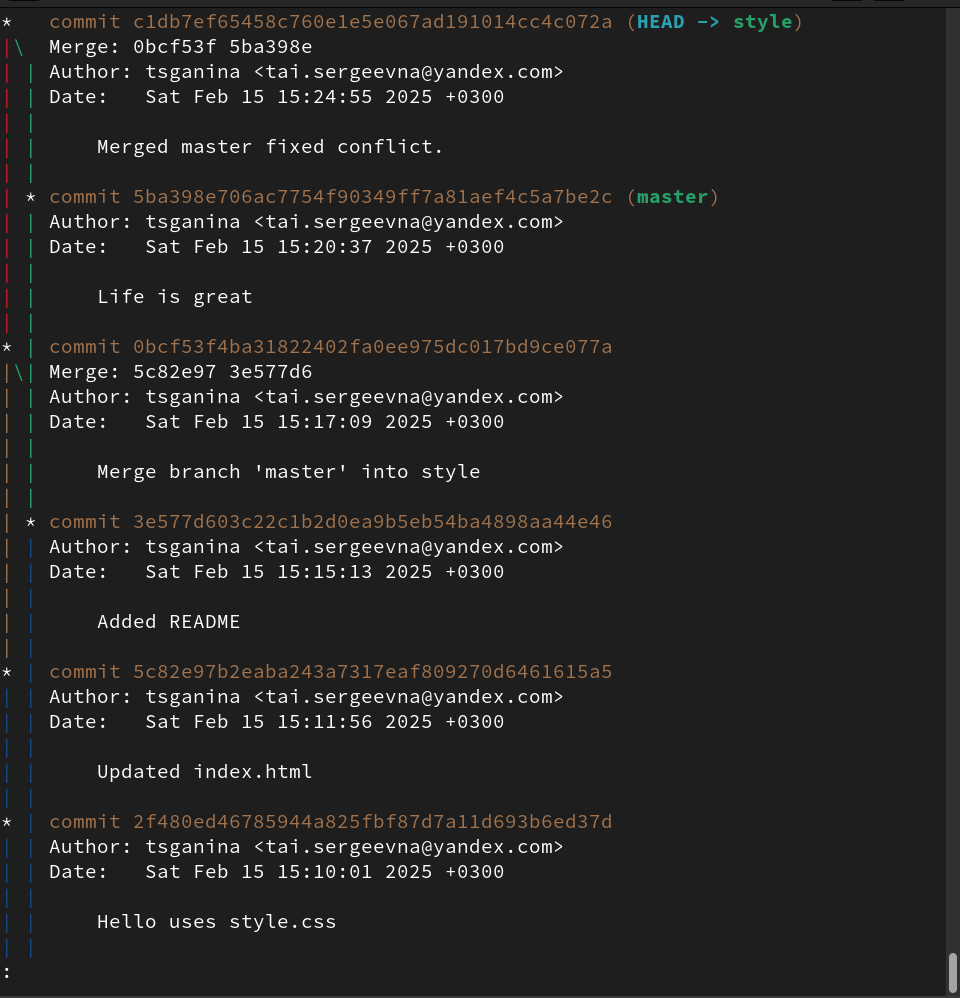


Figure 46: Внесём изменения в lib/hello.html

1. Сброс ветки style. Сброс ветки master.

Вернемся на ветке style к точке перед тем, как мы слили ее с веткой master. Мы можем сбросить ветку к любому коммиту. По сути, это изменение указателя ветки на любую точку дерева коммитов. В этом случае мы хотим вернуться в ветке style в точку перед слиянием с master. Нам необходимо найти последний коммит перед слиянием.(рис. [47](#fig:047), [48](#fig:048))

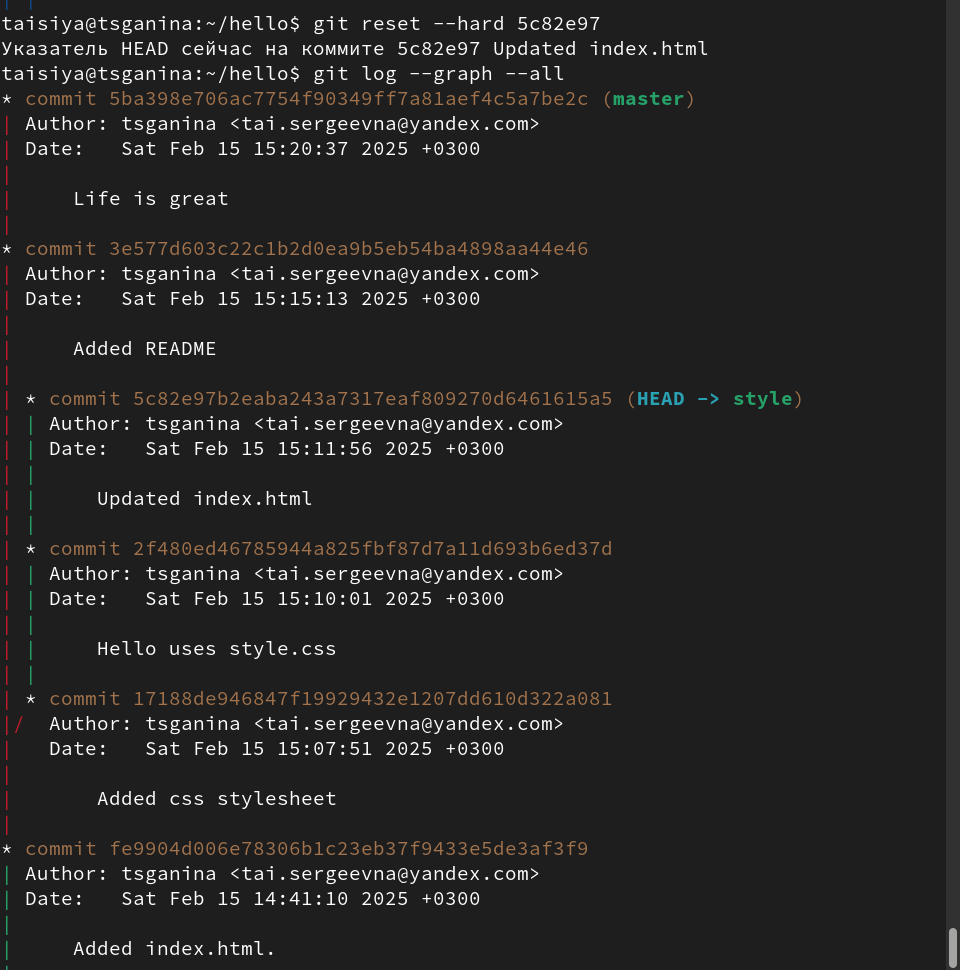


Figure 47: Сброс ветки style

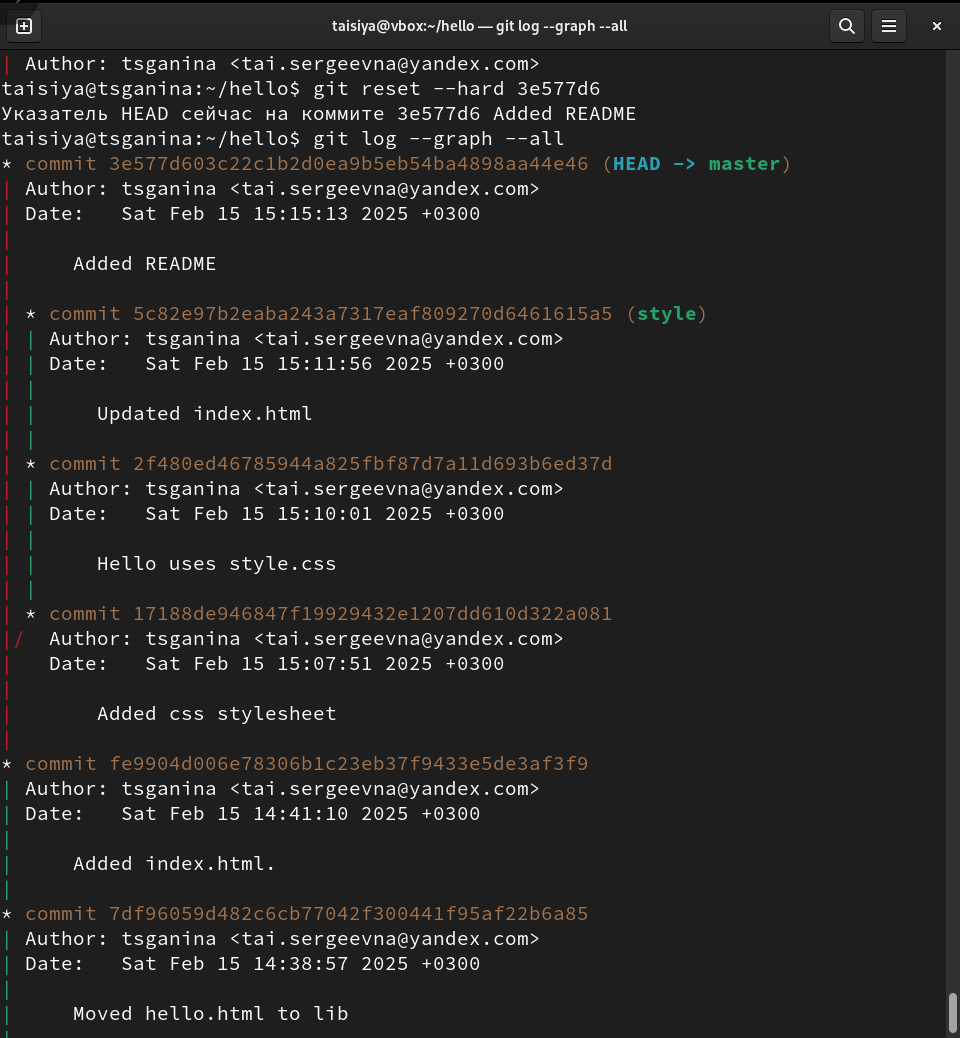


Figure 48: Сброс ветки master

1. Перебазирование.

Используем команду rebase вместо команды merge. Мы вернулись в точку до первого слияния и хотим перенести изменения из ветки master в нашу ветку style. На этот раз для переноса изменений из ветки master мы будем использовать команду git rebase вместо слияния. (рис. [49](#fig:049)).

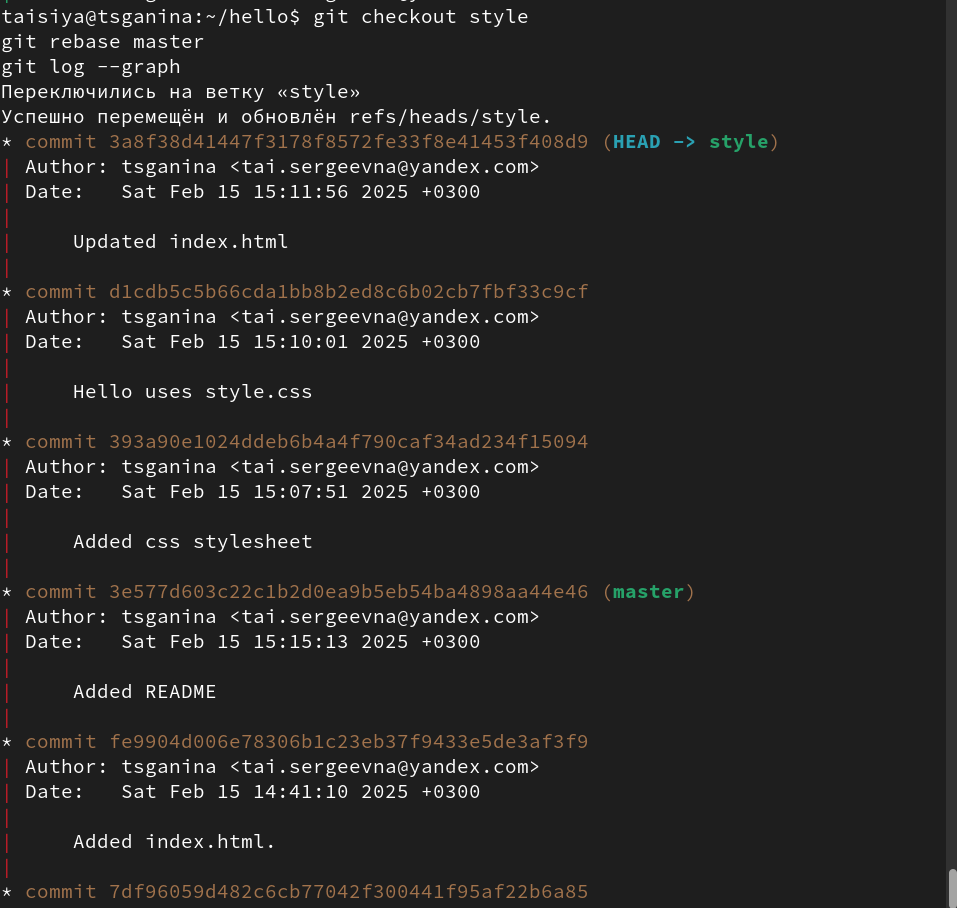


Figure 49: Сброс ветки master

Конечный результат перебазирования очень похож на результат слияния. Ветка style в настоящее время содержит все свои изменения, а также все изменения ветки master. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка master является частью истории коммитов. Это делает цепь коммитов линейной и гораздо более читабельной.

1. Слияние в ветку master.

Мы поддерживали соответствие ветки style с веткой master (с помощью rebase), теперь давайте сольем изменения style в ветку master. (рис. [50](#fig:050)).

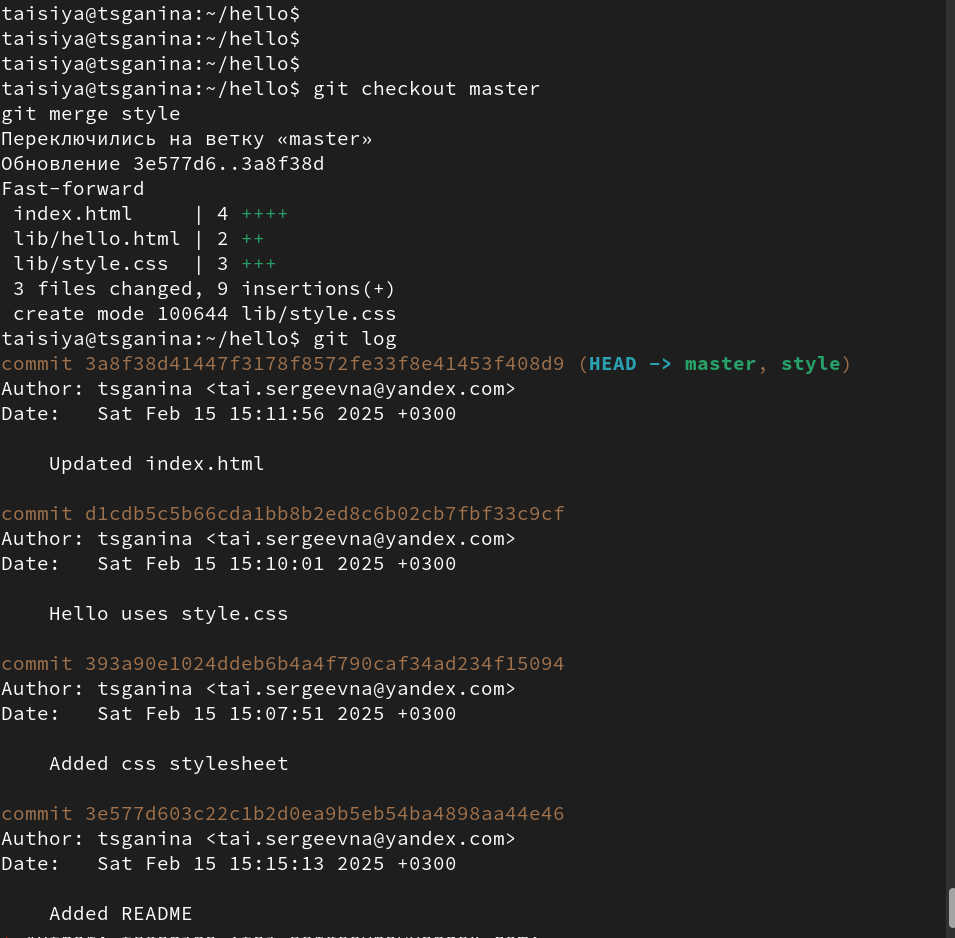


Figure 50: Слияние в ветку master

1. Клонирование репозиториев. Просмотр клонированного репозитория. (рис. [51](#fig:051), [52](#fig:052)).



Figure 51: Создадим клон репозитория hello



Figure 52: Просмотр клонированного репозитория

1. Что такое origin? Удаленные ветки. Изменение оригинального репозитория.

Мы видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию об имени по умолчанию. (рис. [53](#fig:053)).

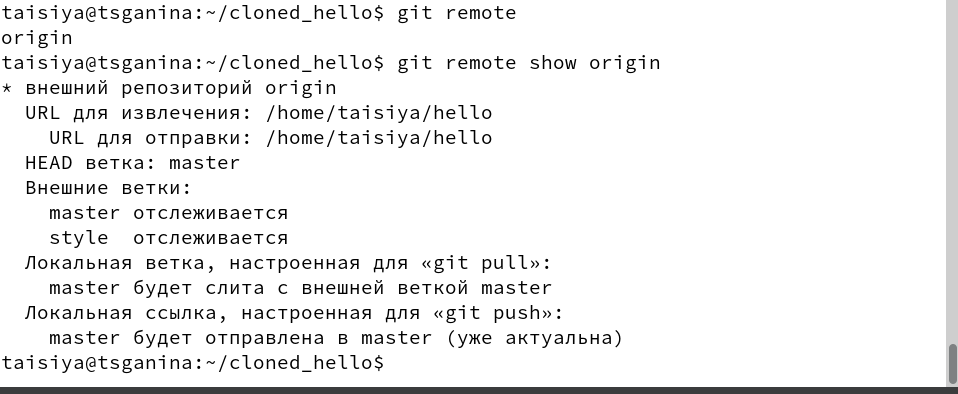


Figure 53: Более подробная информация об имени по умолчанию

1. Слияние извлеченных изменений. Научимся извлекать изменения из удаленного репозитория. (рис. [54](#fig:054), [55](#fig:055), [56](#fig:056), [57](#fig:057)).



Figure 54: Научиться извлекать изменения из удаленного репозитория

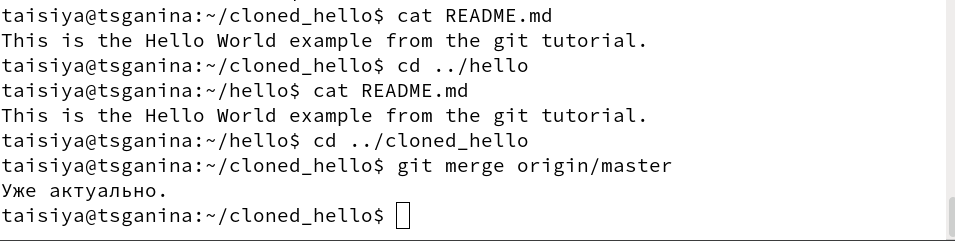


Figure 55: cat README.md



Figure 56: Добавим локальную ветку, которая отслеживает удаленную ветку

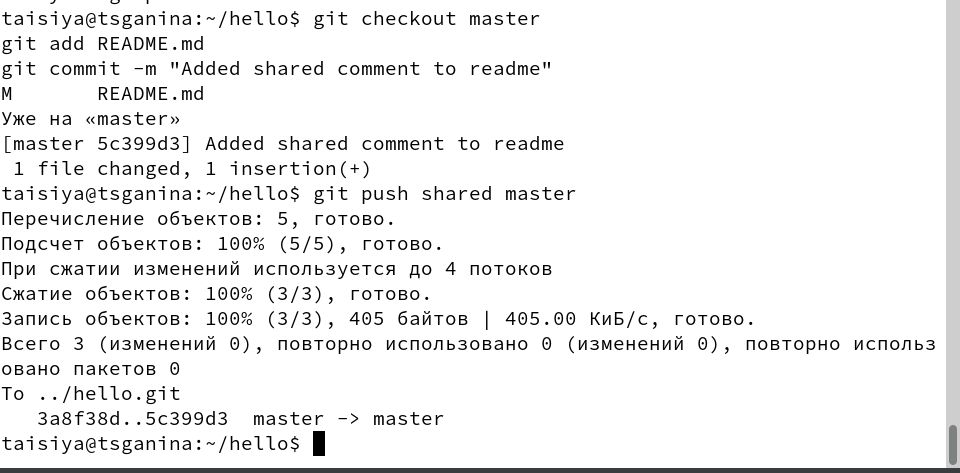


Figure 57: Отправка изменений

1. Извлечение общих изменений. (рис. [58](#fig:058))

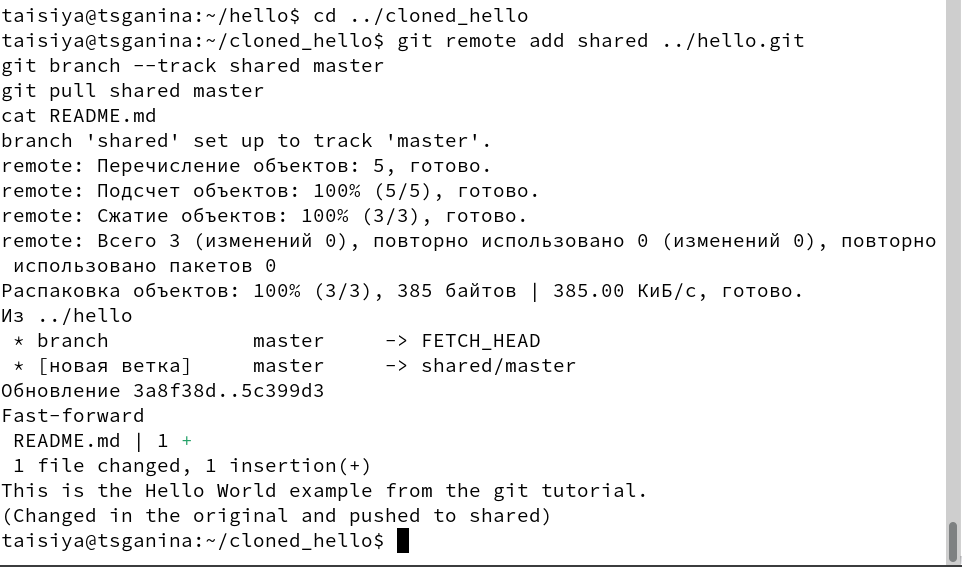


Figure 58: Извлечение общих изменений

1. Написание отчета по лабораторной работе в формате Markdown.

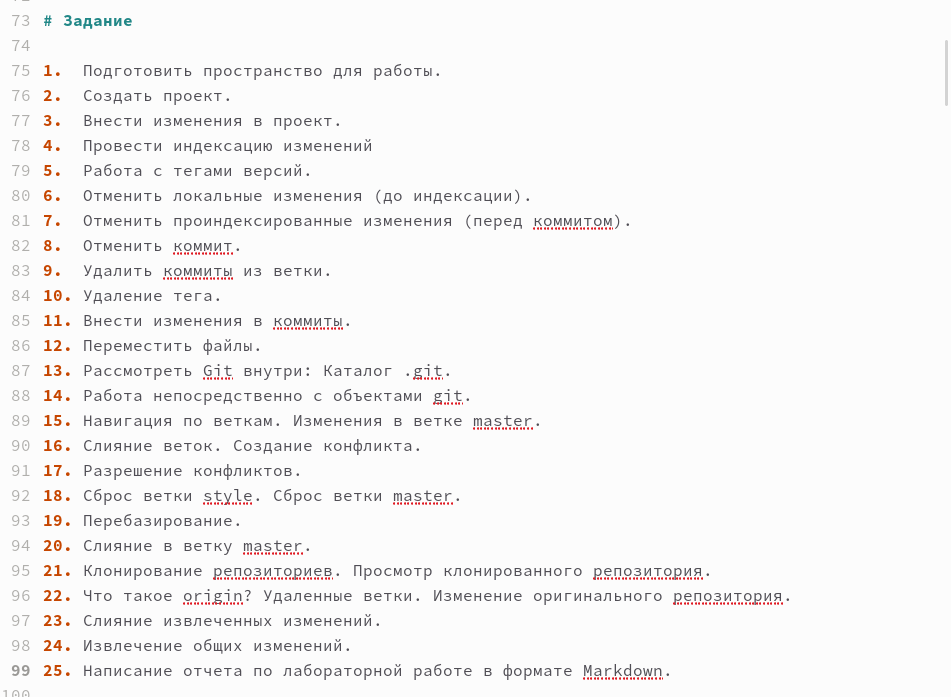


Figure 59: Заголовок и нумерованный список

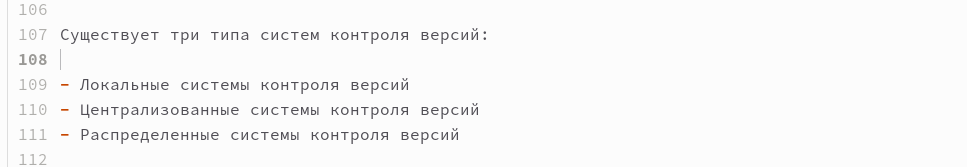


Figure 60: Маркированный список

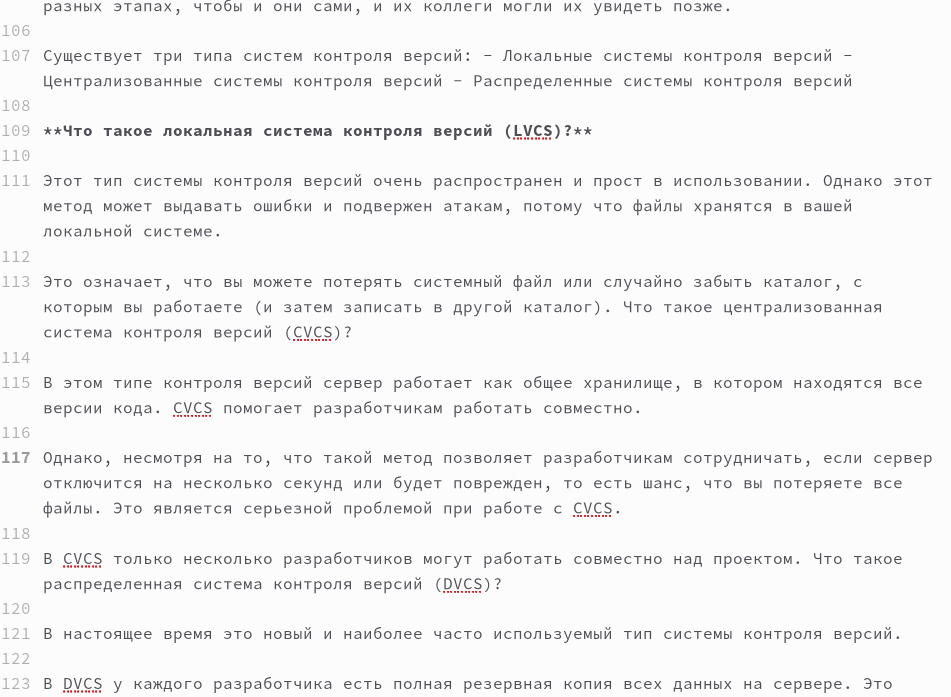


Figure 61: Жирный текст

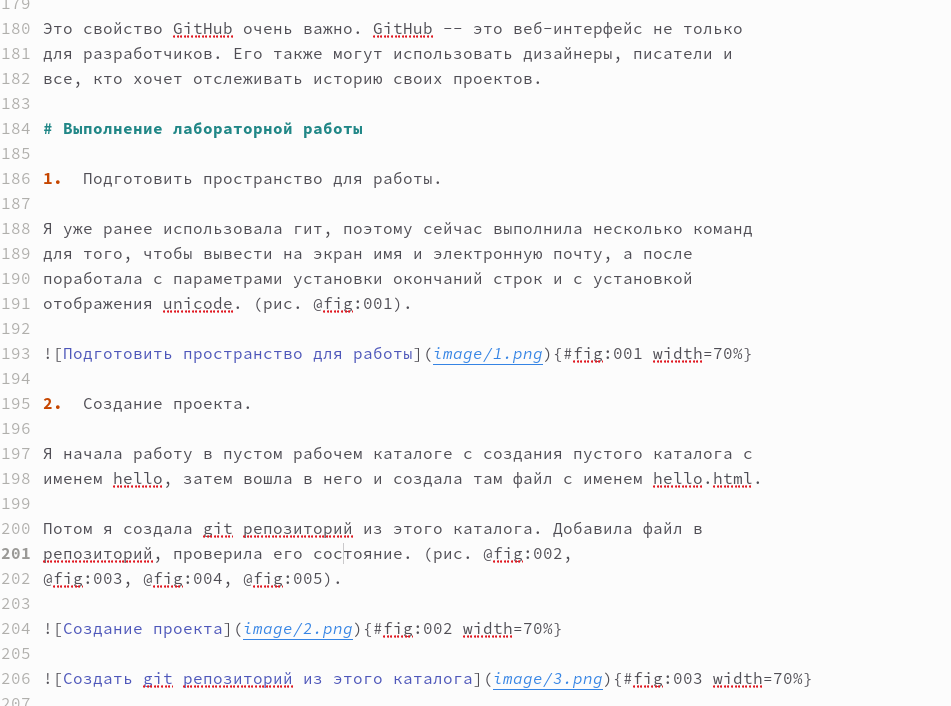


Figure 62: Работа с изображениями

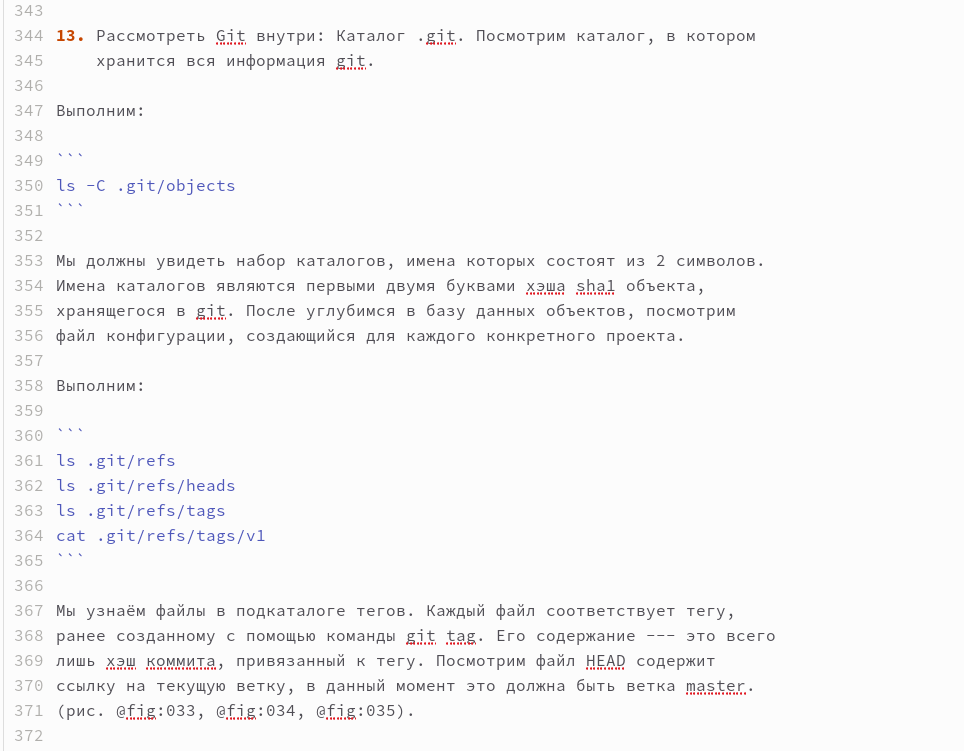


Figure 63: Вставка кода

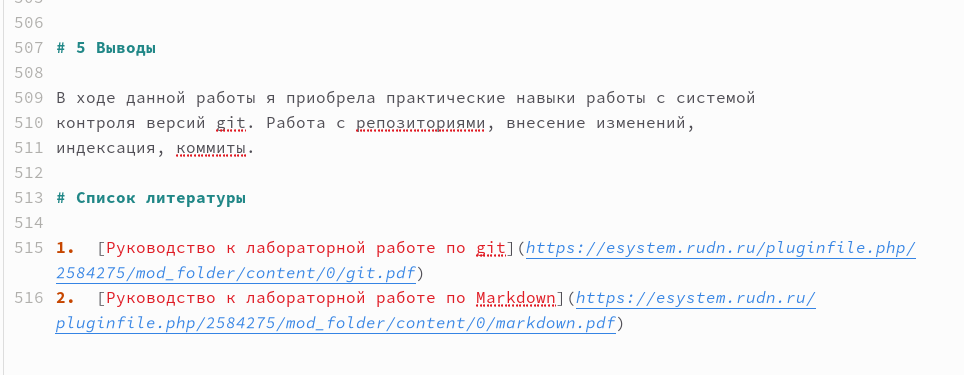


Figure 64: Ссылки

# 5 Выводы

В ходе данной работы я приобрела практические навыки работы с системой контроля версий git. Работа с репозиториями, внесение изменений, индексация, коммиты.

# 6 Список литературы

1. [Руководство к лабораторной работе по git](https://esystem.rudn.ru/pluginfile.php/2584275/mod_folder/content/0/git.pdf)
2. [Руководство к лабораторной работе по Markdown](https://esystem.rudn.ru/pluginfile.php/2584275/mod_folder/content/0/markdown.pdf)