Лабораторная работа №1

Работа с git

Кудряшов Артём Николаевич

Содержание

# 1 Цель работы

Приобрести практические навыки работы с системой управления версиями Git.

# 2 Теоретическое введение

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года; координатор — Дзюн Хамано.

# 3 Выполнение лабораторной работы

## 3.1 Подготовка

Сначала настроим core.autocrlf с параметрами true и input, чтобы сделать все переводы строк текстовых файлов в главном репозитории одинаковыми, а затем настроим отображение unicode

## 3.2 Создание проекта

Создадим пустой каталог hello, а в нём файл с именем hello.html. Затем создадим git репозиторий из этого каталога, выполнив команду git init. Добавим файл в репозиторий и проверим статус, который сообщает, что коммитить нечего(рис. [1](#fig:001)).

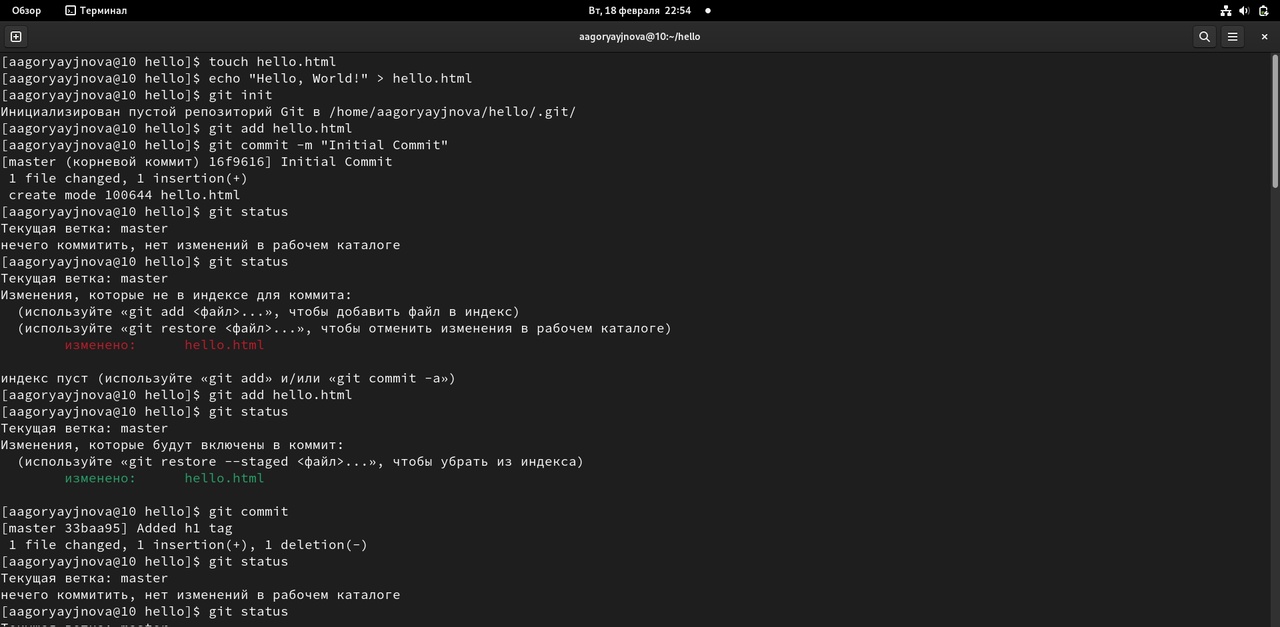


Figure 1: Создание репозитория

## 3.3 Внесение изменений

Изменим содержимое файла hello.html на:

<h1>Hello, World!</h1>

Проверив состояние рабочего каталога увидим, что git знает, что файл hello.html был изменен, но при этом эти изменения еще не зафиксированы в репозитории. Теперь проиндексируем изменения и снова посмотрим статус, в нём указано, что изменения пока не записаны в резапозиторий. И наконец закоммитим изменения, внеся их в репозиторий и снова посмотрим статус, который теперь показывает, что все изменения внесены в репозиторий.

Изменим страницу «Hello, World», чтобы она содержала стандартные теги

и

.

<html>  
 <body>  
 <h1>Hello, World!</h1>  
 </body>  
</html>

Теперь добавим это изменение в индекс git и добавим заголовки HTML (секцию

) к странице «Hello, World» (рис. [2](#fig:002)). Проверив текущий статус увидим, что hello.html указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) является непроиндексированным. Произведем коммит проиндексированного изменения, затем проиндексируем оставшееся изменение, посмотрим статус и прокоммитим его.

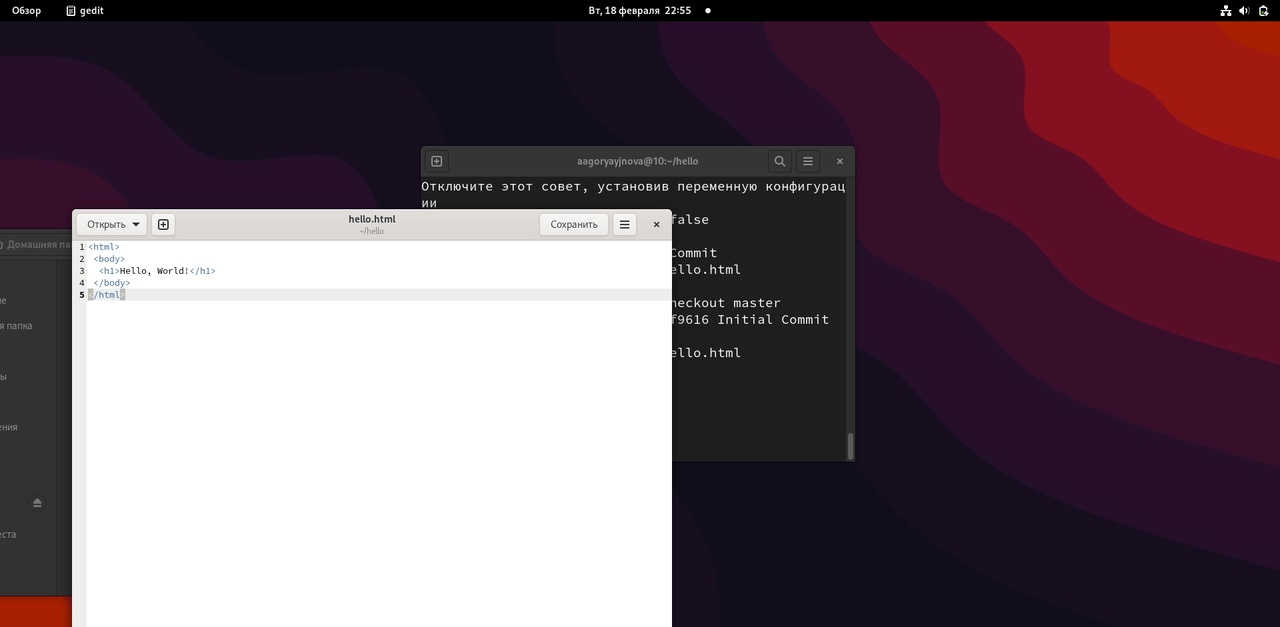


Figure 2: Внесение нескольких изменений в файл

## 3.4 История

Получим список произведённыз изменений в станлартном виде, затем в однострочном, а также с указанием времени и количества.

## 3.5 Получение старых версий

Изучим данные лога и найдем там хэш первого коммита, используя его вернемся к первой верссии и просмотрим файл hello.html, действительно, увидим первую версию. Затем вернемся к последней версии в ветке master и вновь посмотрим на файл(рис. [3](#fig:003)).

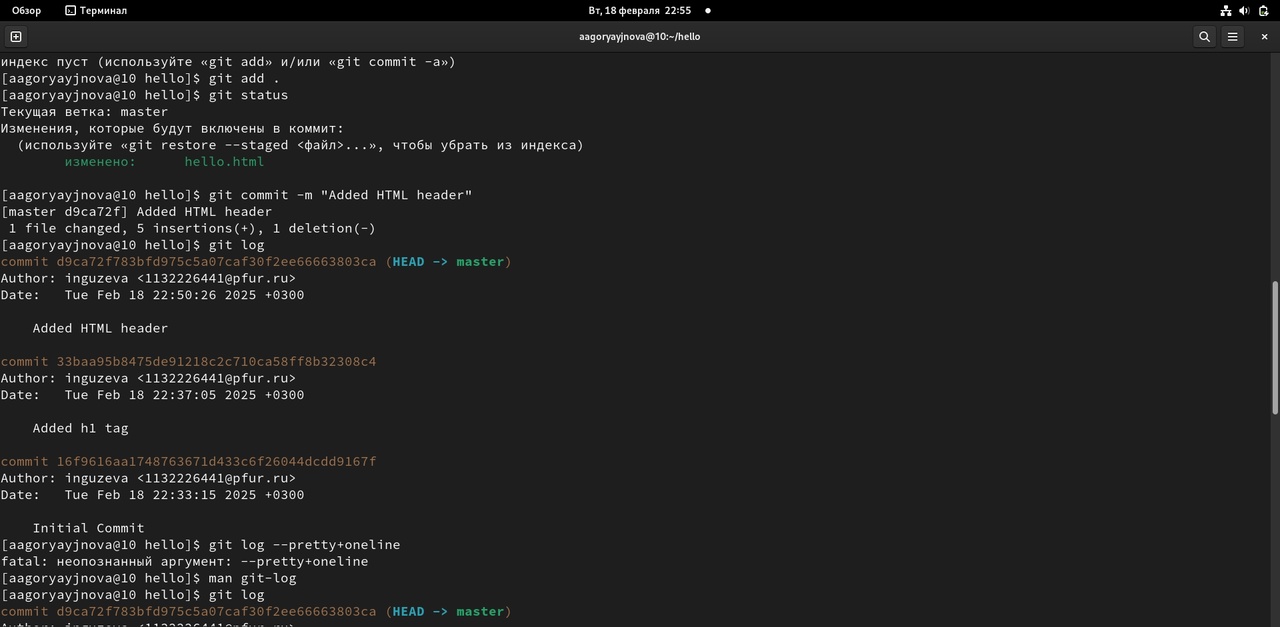


Figure 3: Просмотр разных версий репозитория

## 3.6 Создание тегов версий

Назовем текущую версию страницы hello первой (v1). Создадим тег первой версии и используем его для того чтобы венуться к предыдущей, которой также присвоим тег.

Переключимся по тегам между двумя отмеченными версиями. Просмотрим все доступные теги(их два) и посмотрим теги в логе(рис. [4](#fig:004)).

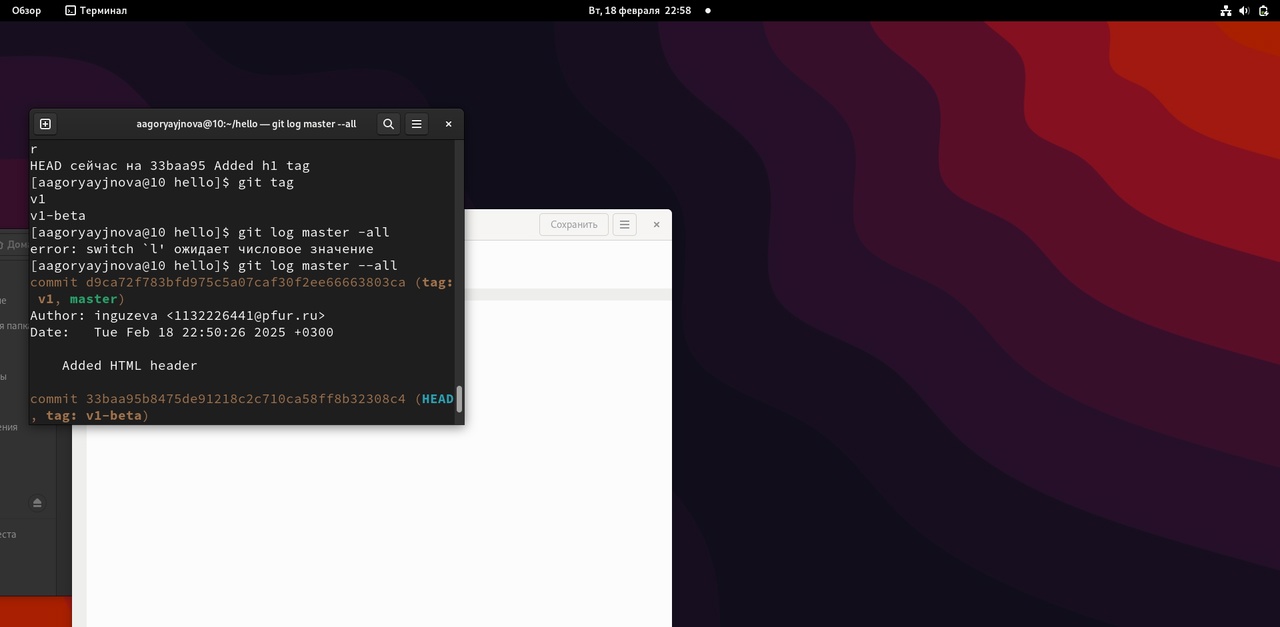


Figure 4: Переключение по имени тега и просмотр доступных тегов

## 3.7 Отмена локальных изменений (до индексации)

Убдеимся, что мы находимся на последнем коммите ветки master и внесем изменение в файл hello.html в виде нежелательного комментария (рис. [5](#fig:005)). Затем проверим статус, увидим, что изменения ещё не проиндексированы. Используем команду git checkout для переключения версии файла hello.html в репозитории.

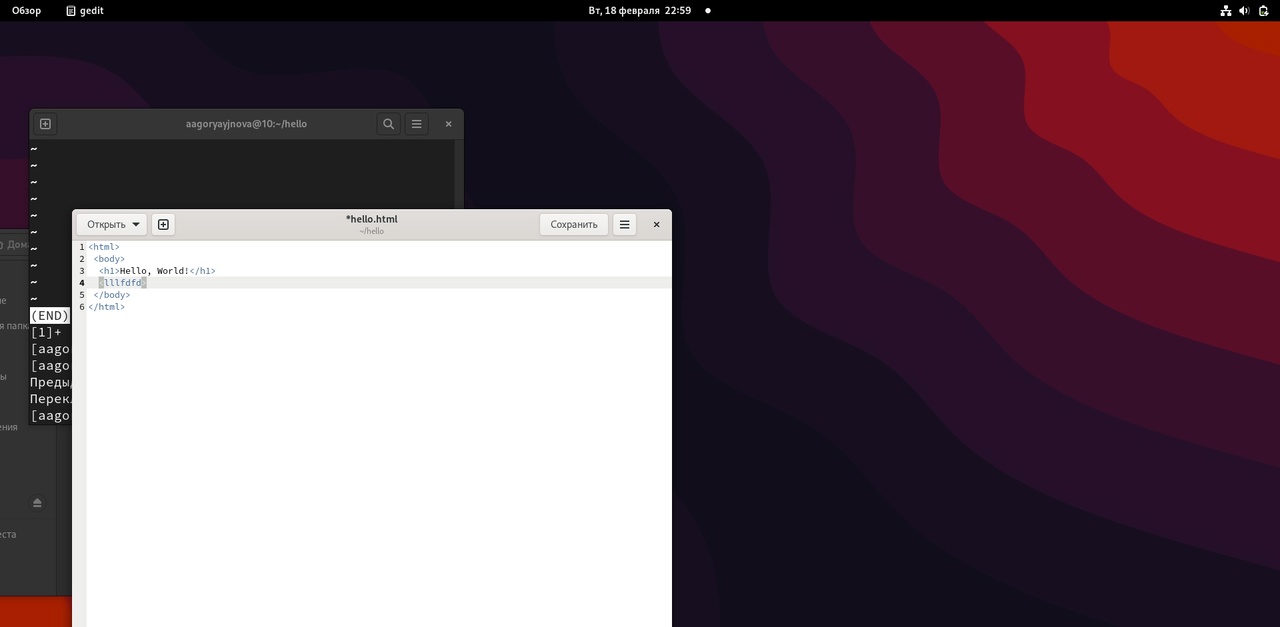


Figure 5: Добавления нежелательного комментария

## 3.8 Отмена проиндексированных изменений (перед коммитом)

Внесем изменение в файл hello.html в виде нежелательного комментария

<html>  
 <head>  
 <!-- This is an unwanted but staged comment -->  
 </head>  
 <body>  
 <h1>Hello, World!</h1>  
 </body>  
</html>

Проиндексируем это изменение и проверим состояние. Состояние показывает, что изменение было проиндексировано и готово к коммиту. Используем команду git reset, чтобы сбросить буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали. И переключимся на последнюю версию коммита, посмотрев статус увидим, что наш каталог опять чист.

## 3.9 Отмена коммитов

Изменим файл hello.html на следующий.

<html>  
 <head>  
 </head>  
 <body>  
 <h1>Hello, World!</h1>  
 <!-- This is an unwanted but committed change -->  
 </body>  
</html>

Проиндексируем изменения файла и прокоммитим их. Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом. Перейдем в редактор, где изменим нежелательный коммит. Проверим лог. Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий(рис. [6](#fig:006)).

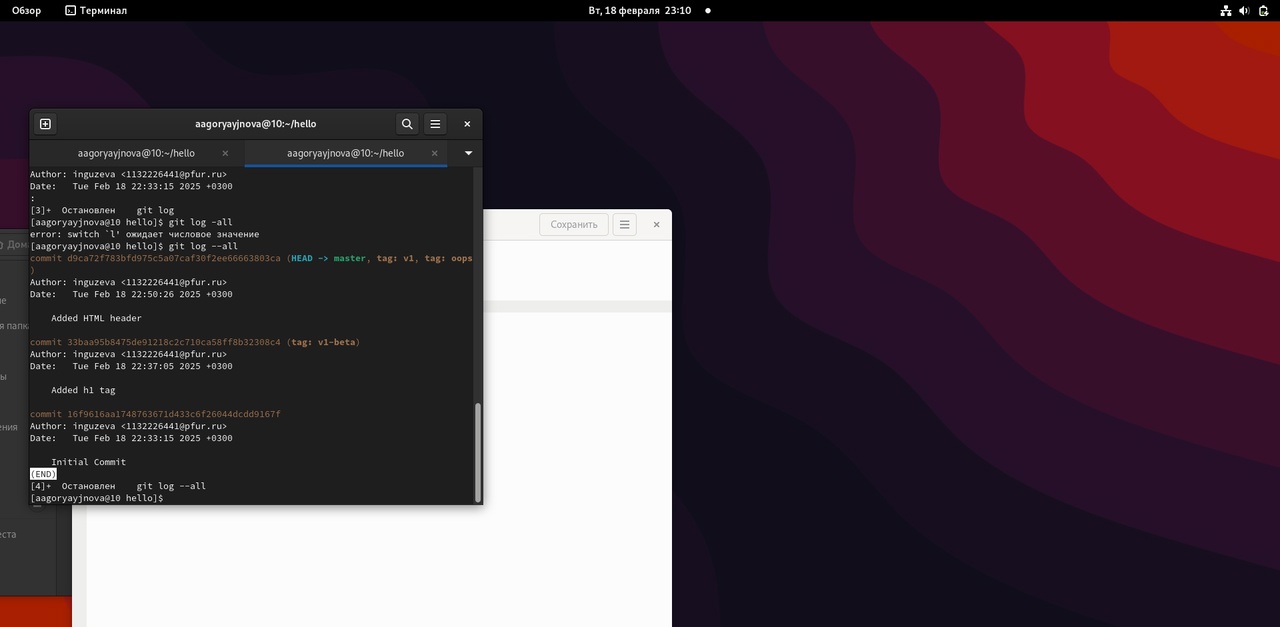


Figure 6: Отмена коммитов

## 3.10 Удаление коммиттов из ветки

Удалим последние два коммита с помощью сброса, сначала отметим последний коммит тегом, чтобы его можно было потом найти. Используем команду git reset, чтобы вернуться к версии до этих коммитов. Теперь в логе их нет, но если посмотреть логи с опцией –all можно всё ещё их увидеть, но метка HEAD находится на нужной нам версии.

## 3.11 Удаление тега oops

Удалим тег oops и коммиты, на которые он ссылался, сборщиком мусора. Теперь этот тег не отображается в репозитории.

## 3.12 Изменение предыдущего коммита

Добавим в страницу комментарий автора.

Затем добавим их в репозиторий. Теперь мы хотим добавить в комментарий автора почту, обноим страницу hello, включив в неё почту. Чтобы у нас остался один коммит, а не два, изменим последний с помощьб опции –amend, теперь в логах отображается последняя версия коммита(рис. [7](#fig:007)).

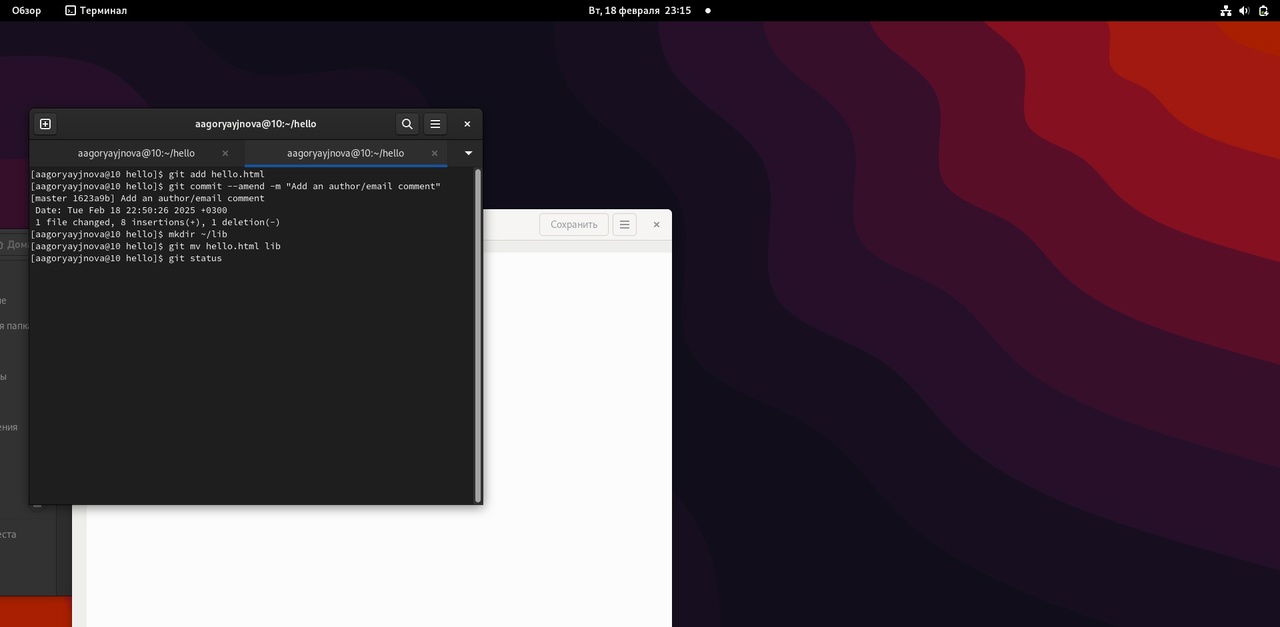


Figure 7: Изменение предыдущего коммита

## 3.13 Перемещение файлов

Переместим наш файл в каталог lib. Для этого создадим его и используем команду git mv, сделаем коммит этого пермещения.

## 3.14 Подробнее о структуре

Добавим файл index.html в наш репозиторий

<html>  
 <body>  
 <iframe src="lib/hello.html" width="200" height="200" />  
 </body>  
</html>

Добавим файл и сделаем коммит. (рис.@fig:008).

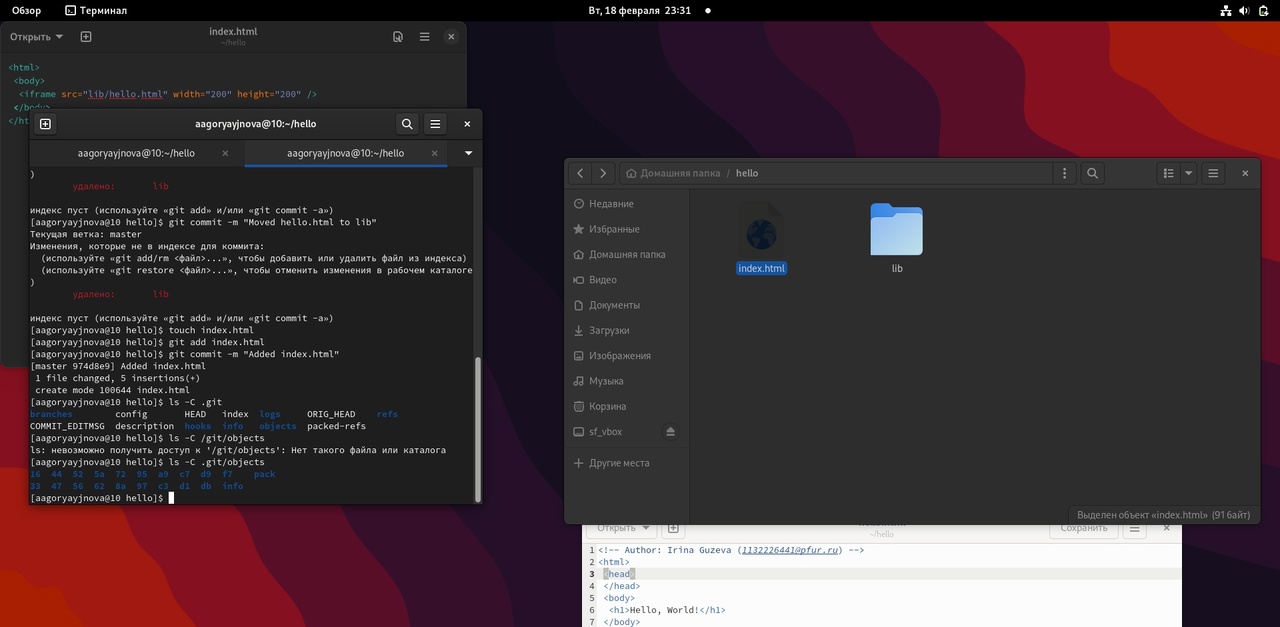


Figure 8: index.html

Теперь при открытии index.html, увидим кусок страницы hello в маленьком окошке

## 3.15 Git внутри: Каталог .git

Просмотрим каталог, в котором хранится вся информация git. Затем посмотрим набор каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git. Посмотрим в один из каталогов с именем из 2 букв. Увидим файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Посмотрим файл конфигурации, создающийся для каждого конкретного проекта. Затем посмотрим подкаталоги .git/refs/heads и .git/refs/tags, а также содержимое файла v1, в нём хранится хэш коммита, привязанный к тегу. Также посмотрим содержимоей файла HEAD, который содержит ссылку на текущую ветку, в данный момент это ветка master((рис. [9](#fig:009)).

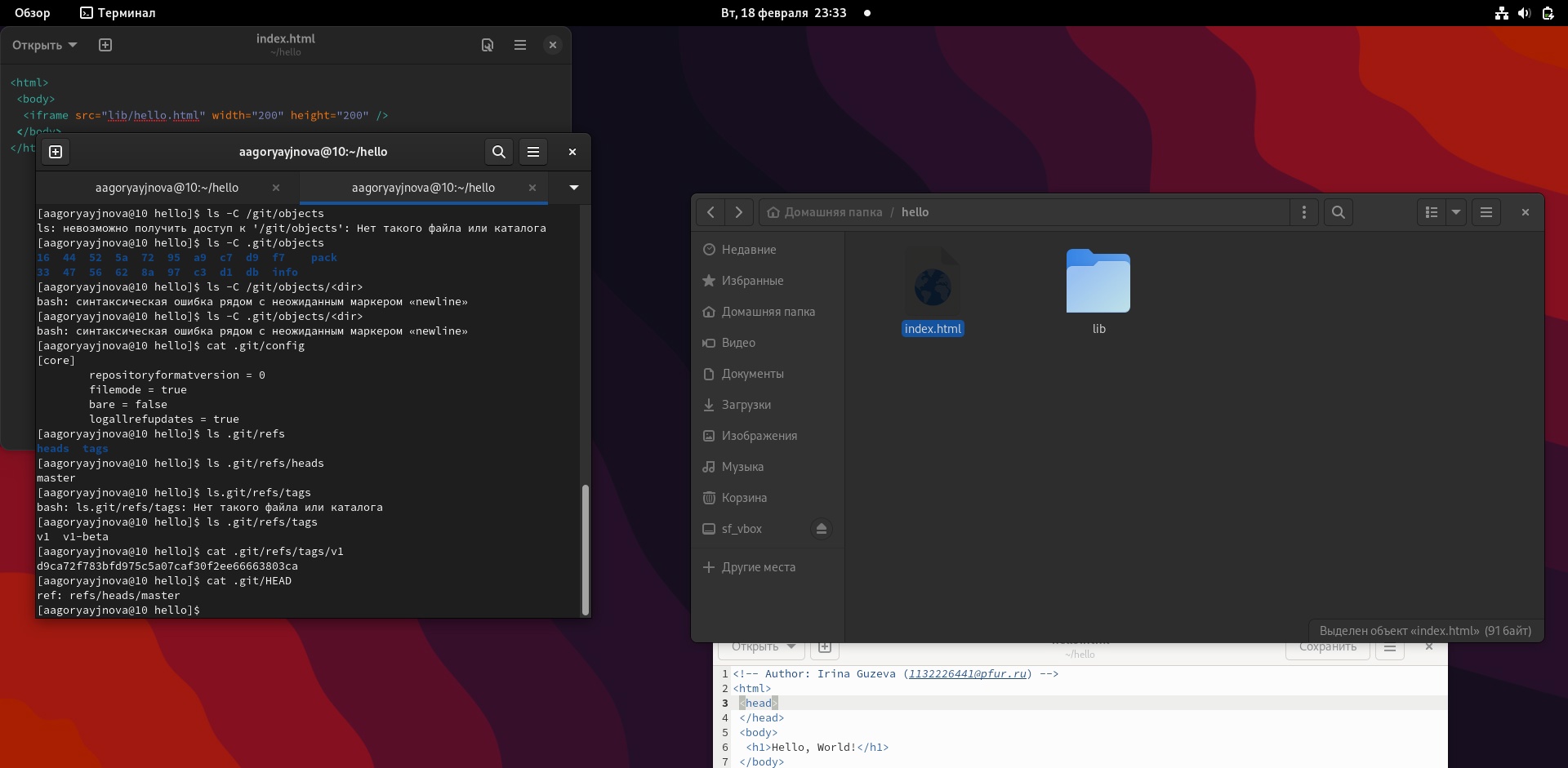


Figure 9: Каталог .git

## 3.16 Работа непосредственно с объектами git

Найдем последний коммит и выедем его с помощью SHA1 хэша. Затем посмотрим дерево каталогов, ссылка на который идёт в последнем коммите, вывдем каталог lib и файл hello.html(рис. [10](#fig:010)).

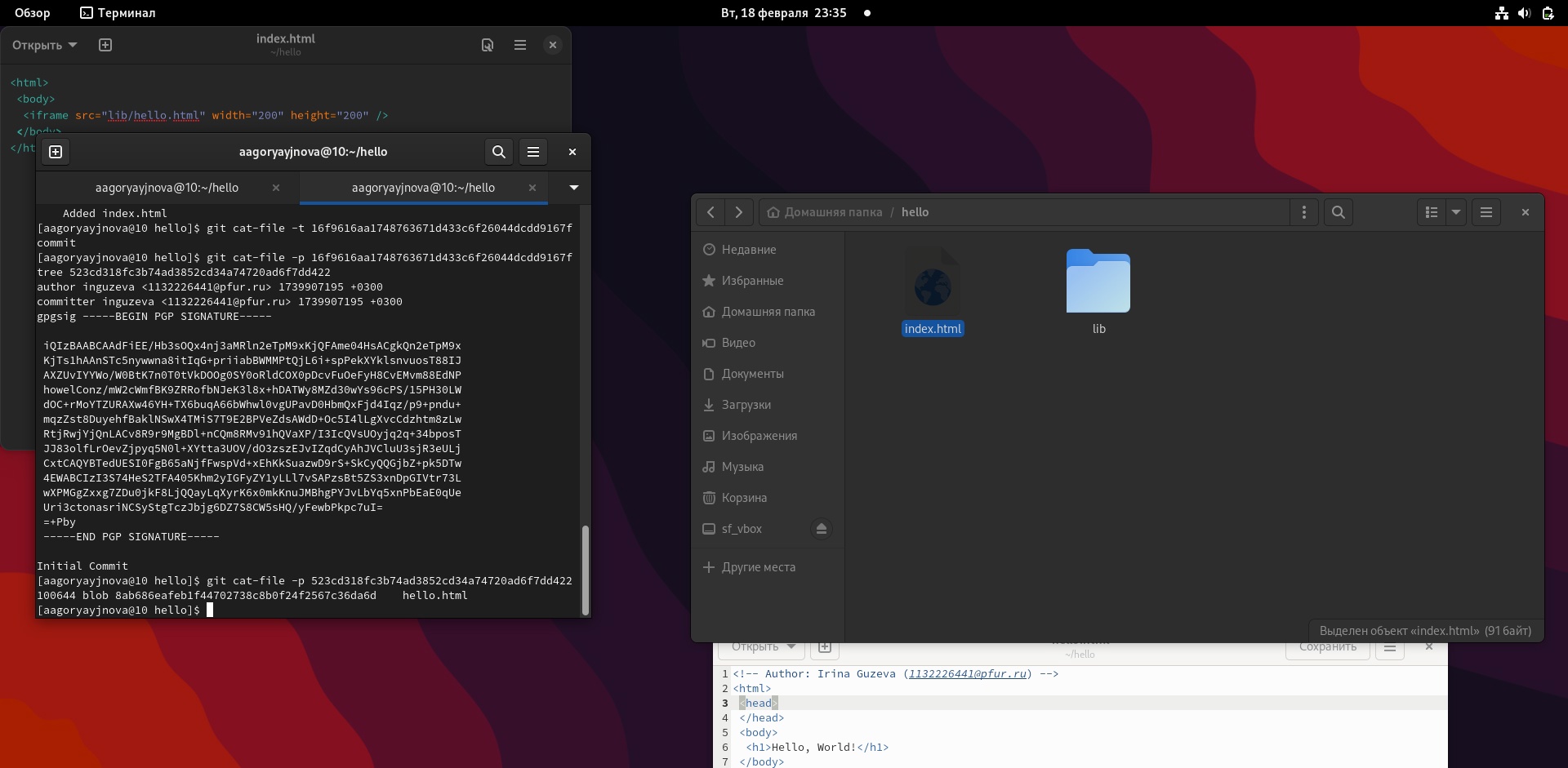


Figure 10: Работа непосредственно с объектами git

Исследуем git репозиторий вручную самостоятельно. Используя хэш родительского коммита последовательно дойдем до первой версии файла hello.html и посмотрим его.

## 3.17 Создание ветки

Создадим новую ветку «style» и перейдем в неё. Добавим туда файл стилей style.css и добавим его в репозиторий. Обновим файл hello.html, чтобы использовать стили style.css и index.html, также обновим их в репозиторий(рис. [11](#fig:011)).

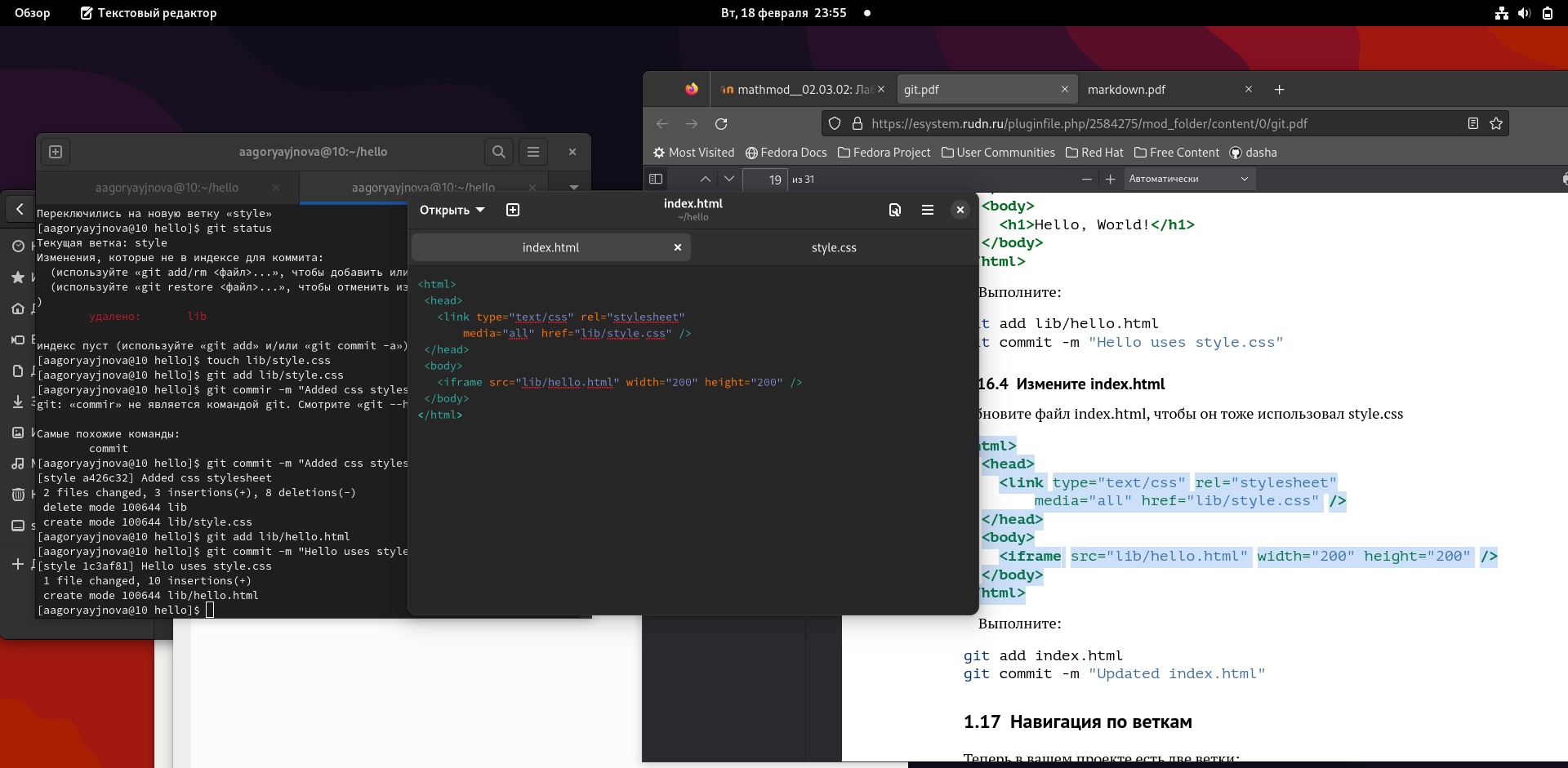


Figure 11: Редактирование файла

## 3.18 Навигация по веткам

Посмотрим все логи. Переключимся обратно на основную ветку и просмотрим содержимое файла ib/hello.html, заметим, что он не использует стили, также просмотрим содержимое этого файла в новой ветке.

## 3.19 Изменения в ветке master

Вернемся в основную ветку и добавим файл README.md. Просмотрим ветки и их различия.

## 3.20 Слияние

Слияние переносит изменения из двух веток в одну. Вернемся к ветке style и сольем master с style.

## 3.21 Создание конфликта

Вернемся в ветку master и создадим конфликт, внеся изменения в файл hello.html. Просмотрим ветки. После коммита «Added README» ветка master была объединена с веткой style, но в настоящее время в master есть дополнительный коммит, который не был слит с style. Последнее изменение в master конфликтует с некоторыми изменениями в style.

## 3.22 Разрешение конфликтов

Вернемся к ветке style и попытаемся объединить ее с новой веткой master. В файле lib/hello.html можно увидеть записи с обеих версий этого файла. Первый раздел — версия текущей ветки (style). Второй раздел — версия ветки master. Внесем изменения в lib/hello.html, оставив только необходимую нам запись и добавим этот файл в репозиторий, чтобы вручную разрешить конфликт.

## 3.23 Сброс ветки style

Вернемся на ветке style к точке перед тем, как мы слили ее с веткой master. Мы хотим вернуться в ветке style в точку перед слиянием с master. Нам необходимо найти последний коммит перед слиянием.

Мы видим, что коммит «Updated index.html» был последним на ветке style перед слиянием. Сбросим ветку style к этому коммиту.

Поищим лог ветки style. Увидим, что у нас в истории больше нет коммитов слияний.

## 3.24 Сброс ветки master

Добавив интерактивный режим в ветку master, мы внесли изменения, конфликтующие с изменениями в ветке style. Давайте вернемся в ветке master в точку перед внесением конфликтующих изменений. Это позволяет нам продемонстрировать работу команды git rebase, не беспокоясь о конфликтах. Просмотрим коммиты ветки master.

Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «Added README».

## 3.25 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.

## 3.26 Слияние в ветку master

Вернемся в ветку master и сольем ветку style в неё с помощью команды git merge.

## 3.27 Клонирование репозиториев

Перейдем в наш рабочий каталог и сделаем клон репозитория hello, затем создадим клон репозитория. Просмотрев его увидим список всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib. Затем просмотрим историю репозитория и увидим список всех коммитов в новый репозиторий, и он совпадает с историей коммитов в оригинальном репозитории. Единствен в названиях веток.

## 3.28 Что такое origin?

Клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Посмотрим, подробную информацию об имени по умолчанию.Для того, чтобы увидеть все ветки используем опцию -a(рис. [12](#fig:012)).

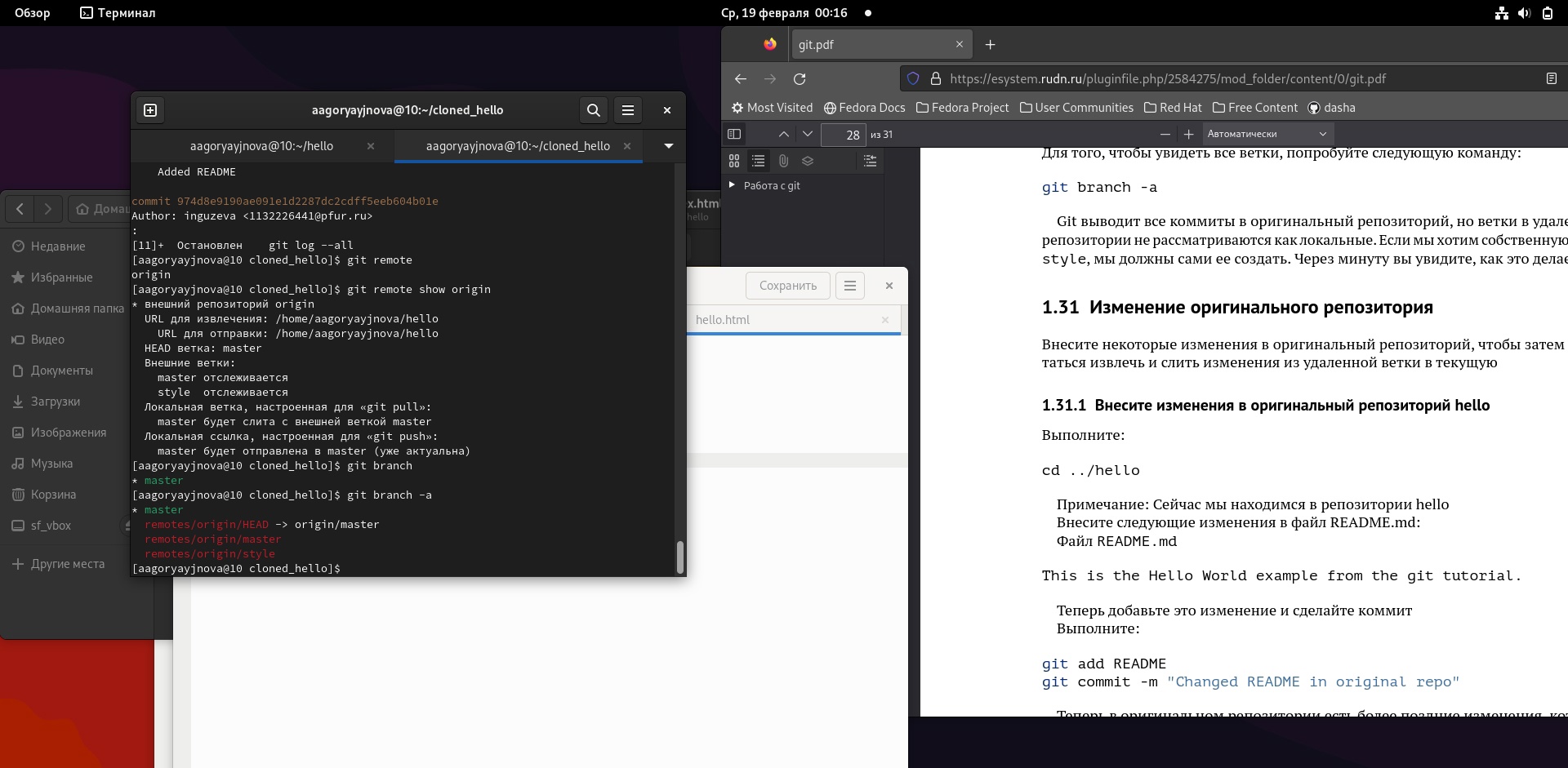


Figure 12: Просмотр имени по умолчанию удаленного репозитория

## 3.29 Удаленные ветки

Посмотрим на ветки, доступные в нашем клонированном репозитории. Можно увидеть, что в списке только ветка master.

## 3.30 Изменение оригинального репозитория

Перейдем в репозиторий hello. Внесем изменения в файл README.md. Затем добавим их в репозиторий. Перейдём в клон репозитория и используем команду git fetch, которая будет извлекать новые коммиты из удаленного репозитория, но не будет сливать их с наработками в локальных ветках.

## 3.31 Слияние извлеченных изменений

Сольем внесённые изменения в главную ветку. Также можно было бы использовать команду git pull, которая является объединением fetch и merge в одну команду.

## 3.32 Добавление ветки наблюдения

Добавим локальную ветку, которая отслеживает удаленную ветку, теперь мы можем видеть ветку style в списке веток и логе.

## 3.33 Создание чистого репозитория

Как правило, репозитории, оканчивающиеся на .git являются чистыми репозиториями. Создадим такой в рабочем каталоге. Затем добавим репозиторий hello.git к нашему оригинальному репозиторию(рис. [13](#fig:013)).

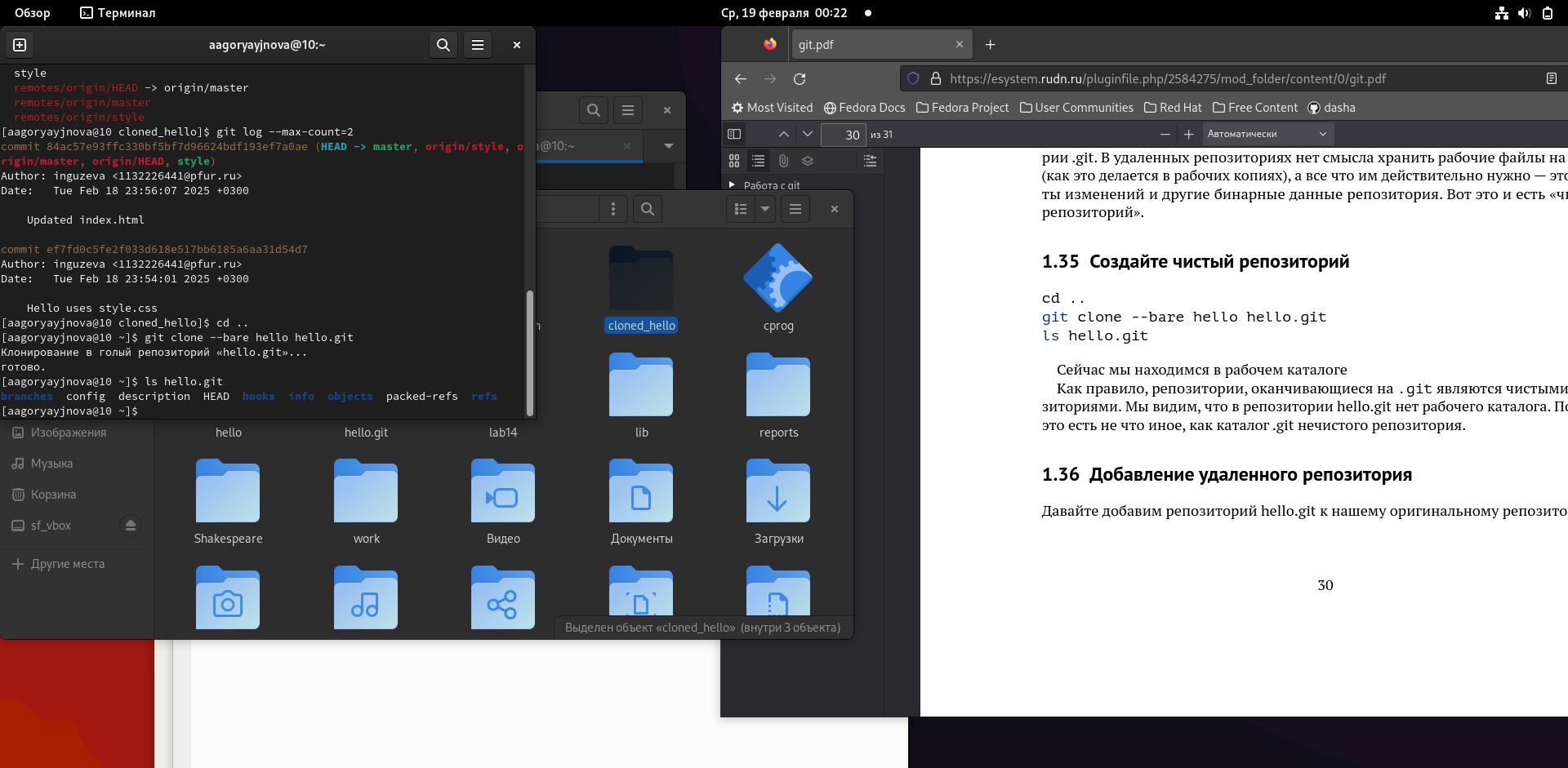


Figure 13: Создание чистого репозитория

## 3.34 Отправка и извлечение изменений

Так как чистые репозитории, как правило, расшариваются на каком-нибудь сетевом сервере, нам необходимо отправить наши изменения в другие репозитории. Начнем с создания изменения для отправки. Отредактируем файл README.md и сделаем коммит, затем отправим изменения в общий репозиторий. Затем извлечем изменения из общего репозитория(рис. [14](#fig:014)).

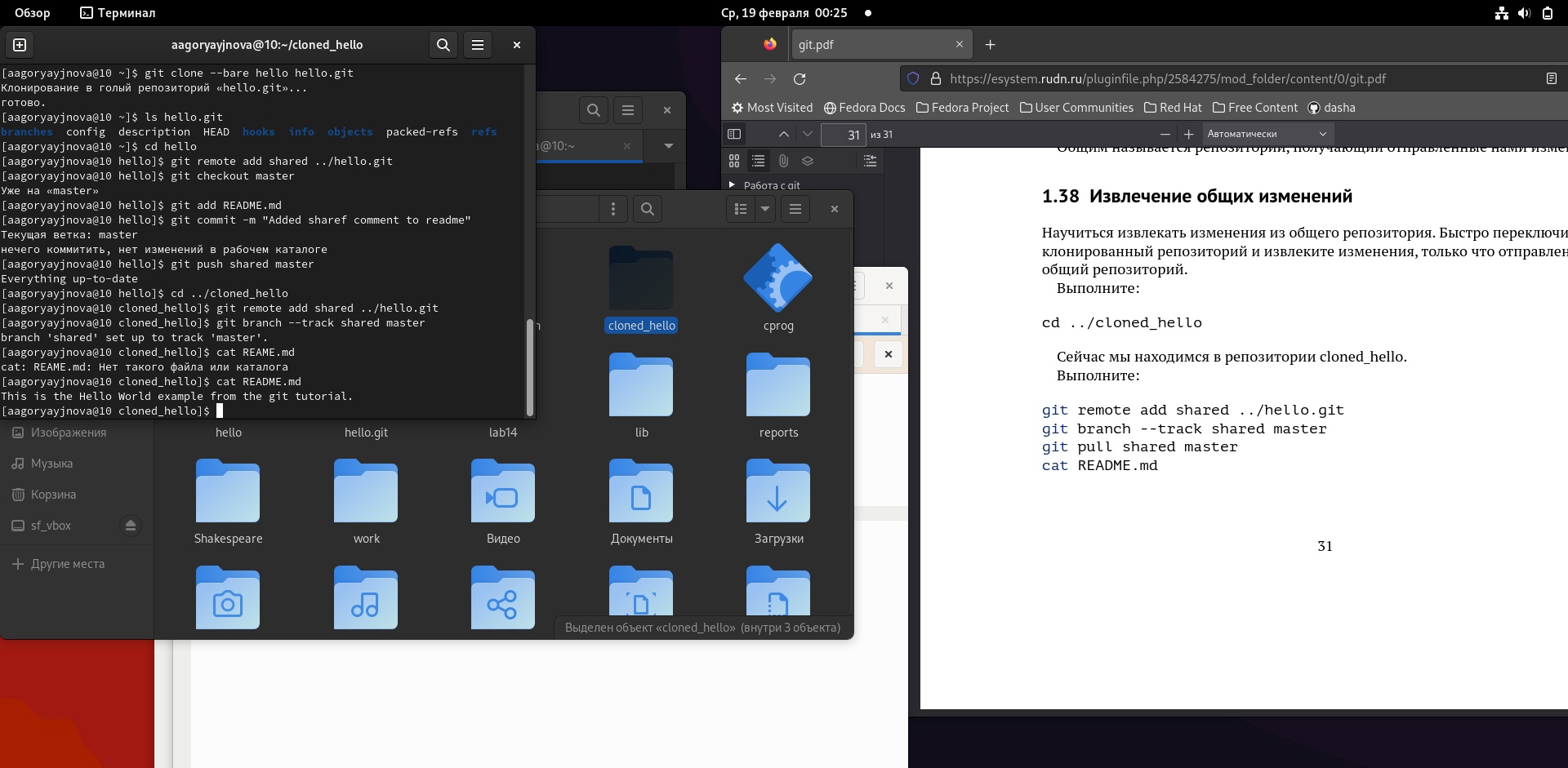


Figure 14: Извлечение изменений

# 4 Выводы

В процессе выполнения данной лабораторной работы я приобрел практические навыки работы с Git.

# Список литературы