Отчёт по лабораторной работе №1

Дисциплина: Математическое моделирование

Зиязетдинов Алмаз Радикович

Содержание

Список иллюстраций

Список таблиц

# 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 [**wiki?**].

# 3 Выполнение лабораторной работы

## 3.1 Подготовка

Ранее я уже использовала git, поэтому мне не пришлось выполнять установку. Для начала настроим core.autocrlf с параметрами true и input, чтобы сделать все переводы строк текстовых файлов в главном репозитории одинаковыми, а затем настроим отображение unicode (рис. fig. 1).

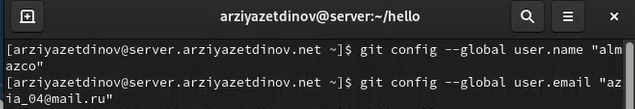


Рис. 1: Настройка git

## 3.2 Создание проекта

Лабораторную работу выполняю в каталоге со своим логином aamishina.

Начнем работу в пустом рабочем каталоге с создания пустого каталога с именем hello, затем войдем в него и создадим там файл с именем hello.html. Запишем в него фразу “Hello world”. Создадим git репозиторий из этого каталога, выполнив команду git init. Добавим файл в репозиторий: git add hello.html и git commit -m “Initial Commit”. Используем команду git status, чтобы проверить текущее состояние репозитория. Команда проверки состояния сообщает, что коммитить нечего. Это означает, что в репозитории хранится текущее состояние рабочего каталога, и нет никаких изменений, ожидающих записи (рис. fig. 2).

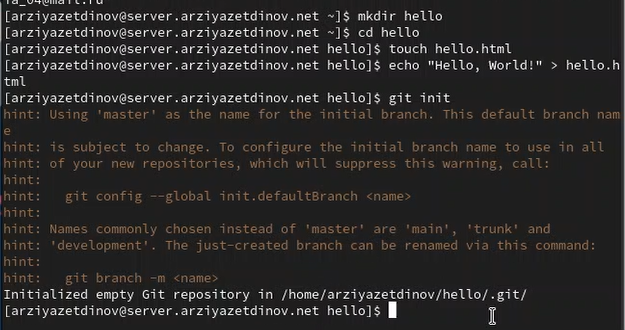


Рис. 2: Создание файла и репозитория. Добавление файла в репозиторий

## 3.3 Внесение изменений

Добавим кое-какие HTML-теги к нашему приветствию. Изменим содержимое файла hello.html на:

Hello, World!

(рис. fig. 3).

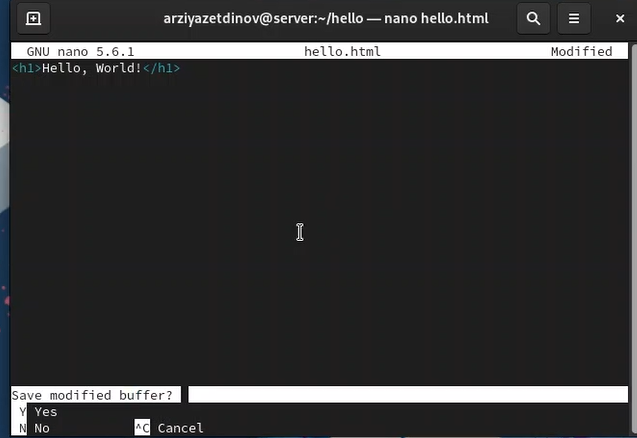


Рис. 3: Файл hello.html

Проверим состояние рабочего каталога: git status. git знает, что файл hello.html был изменен, но при этом эти изменения еще не зафиксированы в репозитории. Также обратим внимание на то, что сообщение о состоянии дает подсказку о том, что нужно делать дальше.

## 3.4 Индексация изменений

Теперь выполним команду git, чтобы проиндексировать изменения и проверим состояние: git add hello.html и git status (рис. fig. 4). Изменения файла hello.html были проиндексированы. Это означает, что git теперь знает об изменении, но изменение пока не записано в репозиторий. Следующий коммит будет включать в себя проиндексированные изменения. Сделаем коммит: git commit. Откроется редактор: в первой строке введем комментарий: «Added h1 tag» (рис. fig. 5). Теперь еще раз проверим состояние: git status. Рабочий каталог чистый, можно продолжить работу.

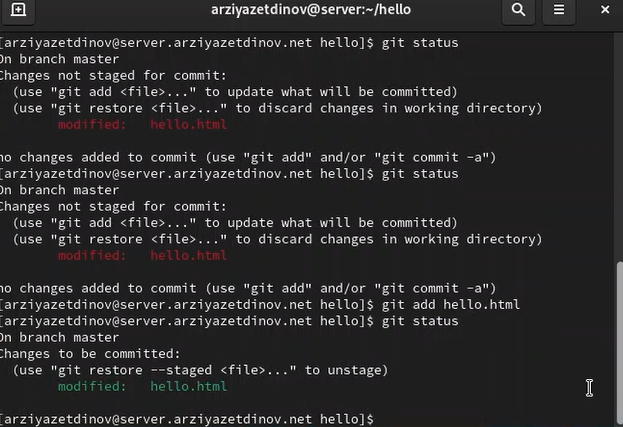


Рис. 4: Индексация и коммит изменений

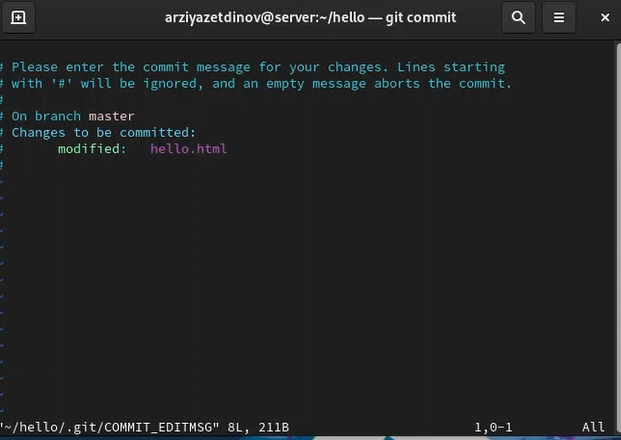


Рис. 5: Редактор

Добавим стандартные теги страницы (рис. fig. 6).

Рис. 6: Файл hello.html

Рис. 6: Файл hello.html

Теперь добавим это изменение в индекс git: git add hello.html (рис. fig. 7). Добавим заголовки HTML (секцию

) к странице «Hello, World» (рис. fig. 8). Проверим текущий статус: git status. Обратим внимание на то, что hello.html указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) является непроиндексированным. Если бы вы делали коммит сейчас, заголовки не были бы сохранены в репозиторий. Произведем коммит проиндексированного изменения, а затем еще раз проверим состояние: git commit -m “Added standard HTML page tags” и git status. Состояние команды говорит о том, что hello.html имеет незафиксированные изменения, но уже не в буферной зоне. Теперь добавим второе изменение в индекс, а затем проверим состояние с помощью команды git status: git add . и git status. Второе изменение было проиндексировано и готово к коммиту. Сделаем коммит второго изменения: git commit -m “Added HTML header”.

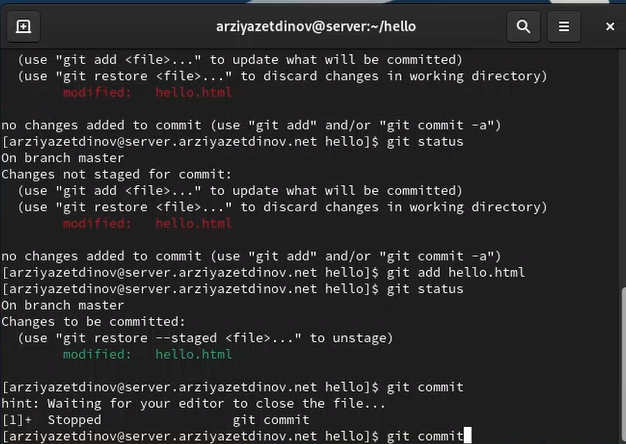


Рис. 7: Индексация и коммит изменений



Рис. 8: Файл hello.html

## 3.5 История

Получим список произведенных изменений: git log. Посмотрим однострочный формат истории: git log –pretty=oneline (рис. fig. 9).

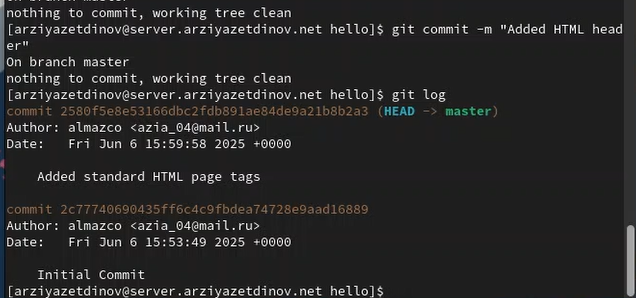


Рис. 9: История

Возвращаться назад в историю очень просто. Команда checkout скопирует любой снимок из репозитория в рабочий каталог. Получим хэши предыдущих версий: git log. Изучим данные лога и найдем хэш для первого коммита. Используем этот хэш-код (достаточно первых 7 знаков) в команде: git checkout . Затем проверим содержимое файла hello.html: cat hello.html. Веремся к последней версии в ветке main: git checkout main и cat hello.html (рис. fig. 10).

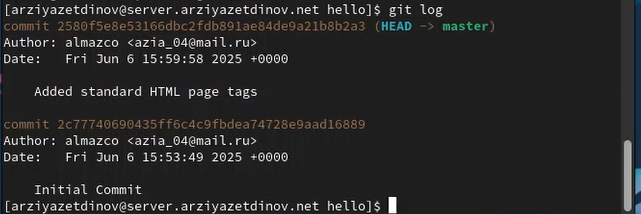


Рис. 10: Просмотр разных версий репозитория

## 3.6 Создание тегов версий

Давайте назовем текущую версию страницы hello первой (v1). Создаем тег первой версии: git tag v1. Давайте создадим тег для версии, которая идет перед текущей версией и назовем его v1-beta. Переключаемся на предыдущую версию и просмотрим содержимое файла hello.html: git checkout v1^ и cat hello.html. Это версия c тегами

и

, но еще пока без

. Давайте сделаем ее версией v1-beta: git tag v1-beta. Теперь попереключаемся между двумя отмеченными версиями: git checkout v1 и git checkout v1-beta (рис. fig. 11).

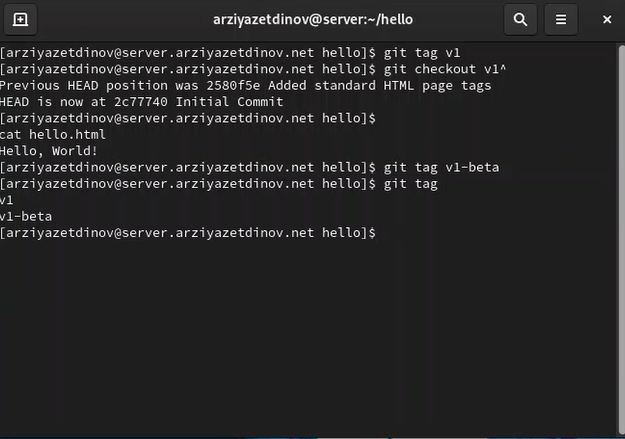


Рис. 11: Создание тегов версий, переключение между ними

Посмотрим, какие теги доступны, используя команду git tag: git tag. Также посмотрим теги в логе: git log main –all. Можем видеть теги (v1 и v1-beta) в логе вместе с именем ветки (main). Кроме того HEAD показывает коммит, на который вы переключились (на данный момент это v1-beta) (рис. fig. 12).

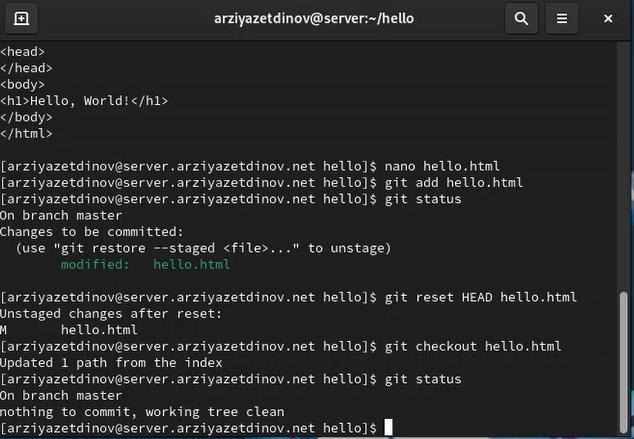


Рис. 12: Доступные теги

## 3.7 Отмена локальных изменений (до индексации)

Перейдем на последний коммит ветки main: git checkout main (рис. fig. 13). Внесем изменение в файл hello.html в виде нежелательного комментария (рис. fig. 14). Проверим состояние рабочего каталога: git status. Мы видим, что файл hello.html был изменен, но еще не проиндексирован. Используем команду git checkout для переключения версии файла hello.html в репозитории: git checkout hello.html, git status и cat hello.html. Команда git status показывает нам, что не было произведено никаких изменений, не зафиксированных в рабочем каталоге.

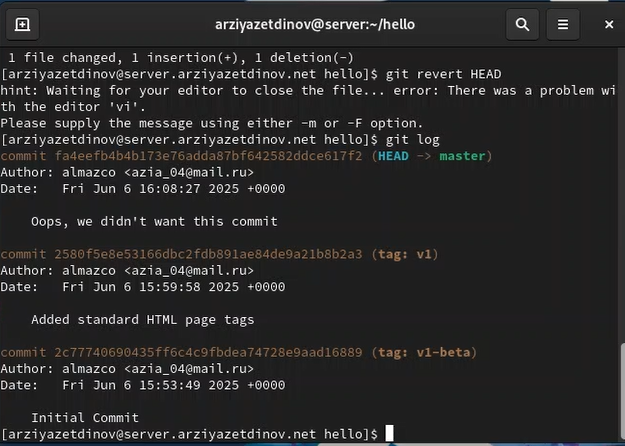


Рис. 13: Отмена локальных изменений (до индексации)

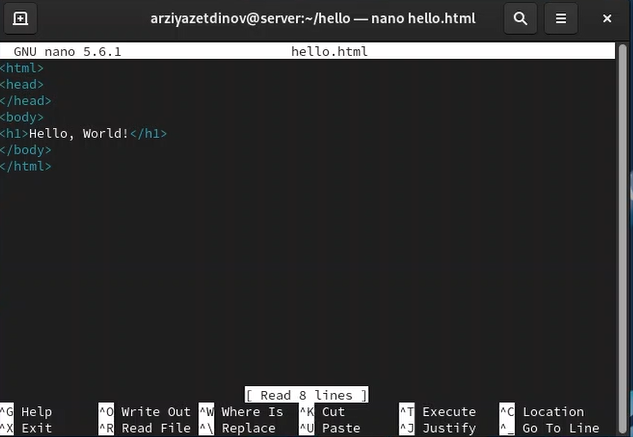


Рис. 14: Изменения в файле hello.html

## 3.8 Отмена проиндексированных изменений (перед коммитом)

Изменим файл (рис. fig. 15).

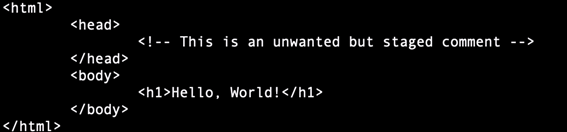


Рис. 15: Изменения в файле hello.html

Проиндексируем изменения: git add hello.html. Проверим состояние нежелательного изменения: git status. Состояние показывает, что изменение было проиндексировано и готово к коммиту. Выполним сброс буферной зоны: git reset HEAD hello.html. Команда git reset сбрасывает буферную зону к HEAD. Это очищает буферную зону от изменений, которые мы только что проиндексировали. Команда git reset (по умолчанию) не изменяет рабочий каталог. Поэтому рабочий каталог все еще содержит нежелательный комментарий. Мы можем использовать команду git checkout, чтобы удалить нежелательные изменения в рабочем каталоге. Переключимся на версию коммита: git checkout hello.html и git status. Наш рабочий каталог опять чист (рис. fig. 16).

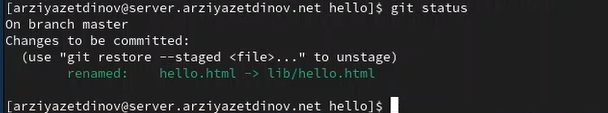


Рис. 16: Отмена локальных изменений (перед коммитом)

## 3.9 Отмена коммитов

Отменим коммит путем создания нового коммита, отменяющего нежелательные изменения. Изменим файл hello.html:

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

Выполним: git add hello.html и git commit -m “Oops, we didn’t want this commit” (рис. fig. **¿fig:017?**). Сделаем коммит с новыми изменениями, отменяющими предыдущие: git revert HEAD. В редакторе отредактируем коммит-сообщение . Сохраним и закроем файл. Проверим лог: git log.

## 3.10 Удаление коммиттов из ветки

Прежде чем удалить коммиты, давайте отметим последний коммит тегом, чтобы потом можно было его найти: git tag oops. Глядя на историю лога, видим, что коммит с тегом «v1» является коммитом, предшествующим ошибочному коммиту. Давайте сбросим ветку до этой точки. Поскольку ветка имеет тег, мы можем использовать имя тега в команде сброса (если она не имеет тега, мы можем использовать хэш-значение): git reset –hard v1 и git log. Наша ветка main теперь указывает на коммит v1, а коммитов Oops и Revert Oops в ветке уже нет .

Давайте посмотрим на все коммиты: git log –all (рис. fig. **¿fig:020?**). Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории.

## 3.11 Удаление тега oops

Давайте удалим тег oops его и коммиты, на которые он ссылался, сборщиком мусора: git tag -d oops и git log –all. Тег «oops» больше не будет отображаться в репозитории .

## 3.12 Внесение изменений в коммиты

Изменим страницу, а затем сделаем коммит.

Обновим страницу hello, включив в нее email

Изменим предыдущий коммит: git add hello.html и git commit –amend -m “Add an author/email comment”. Просмотрим историю: git log. Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email».

## 3.13 Перемещение файлов

Переместим файл hello.html в каталог lib: mkdir lib, git mv hello.html lib и git status. Сделаем коммит этого перемещения: git commit -m “Moved hello.html to lib” (рис. fig. **¿fig:025?**).

Перемещение файлов ## 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: git add index.html и git commit -m “Added index.html” (рис. fig. **¿fig:026?**).Теперь при открытии index.html, видим кусок страницы hello в маленьком окошке

## 3.14 Git внутри: Каталог .git

Выполним: ls -C .git. Это каталог, в котором хранится вся информация git. Выполним: ls -C .git/objects. Видим набор каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git. Выполним: ls -C .git/objects/01. Смотрим в один из каталогов с именем из 2 букв. Видим файл с именем из 38 символов. Выполним: cat .git/config. Это файл конфигурации, создающийся для каждого конкретного проекта. Записи в этом файле будут перезаписывать записи в файле .gitconfig вашего главного каталога, по крайней мере в рамках этого проекта. Выполним: ls .git/refs, ls .git/refs/heads, ls .git/refs/tags и cat .git/refs/tags/v1. Выполним: cat .git/HEAD. Файл HEAD содержит ссылку на текущую ветку, в данный момент это ветка main (рис. fig. 17).

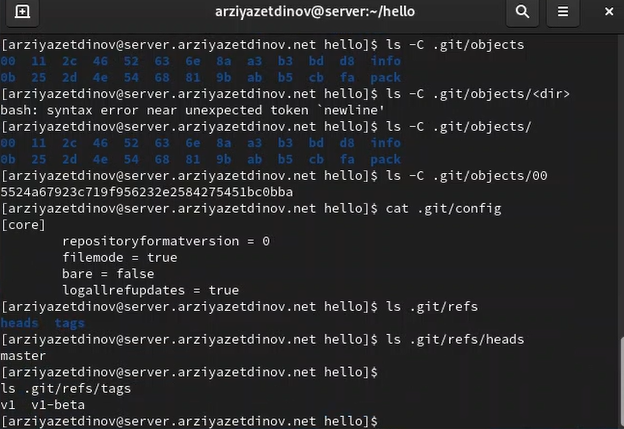


Рис. 17: Git внутри: Каталог .git

## 3.15 Работа непосредственно с объектами git

Поиск последнего коммита: git log –max-count=1. Вывод последнего коммита с помощью SHA1 хэша: git cat-file -t и git cat-file -p . Поиск дерева, используем SHA1 хэш из строки «дерева», из списка выше: git cat-file -p . Вывод каталога lib: git cat-file -p . Вывод файла hello.html: git cat-file -p (рис. fig. 18).

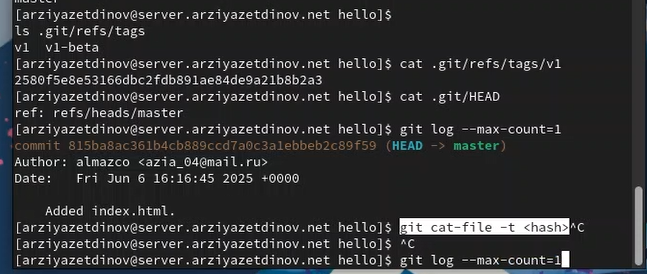


Рис. 18: Работа с объектами git

## 3.16 Создание ветки

Создадим ветку «style»: git checkout -b style и git status. Добавим файл стилей style.css: touch lib/style.css, git add lib/style.css и git commit -m “Added css stylesheet” (рис. fig. 19).

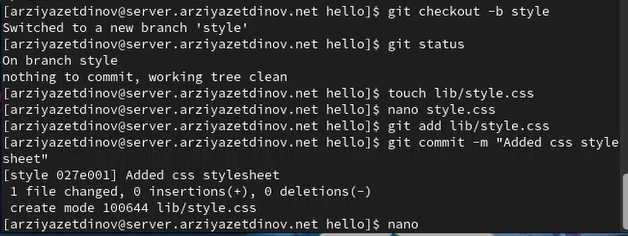


Рис. 19: Создание ветки style и файла style.css

Обновим файл hello.html, чтобы использовать стили style.css (рис. fig. 20).

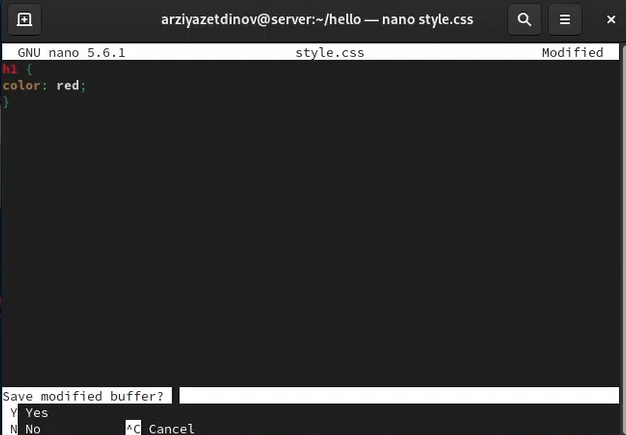


Рис. 20: Файл hello.html

Выполним: git add lib/hello.html и git commit -m “Hello uses style.css”. Изменим index.html, чтобы он тоже использовал style.css. Выполним: git add index.html и git commit -m “Updated index.html” (рис. fig. 21).

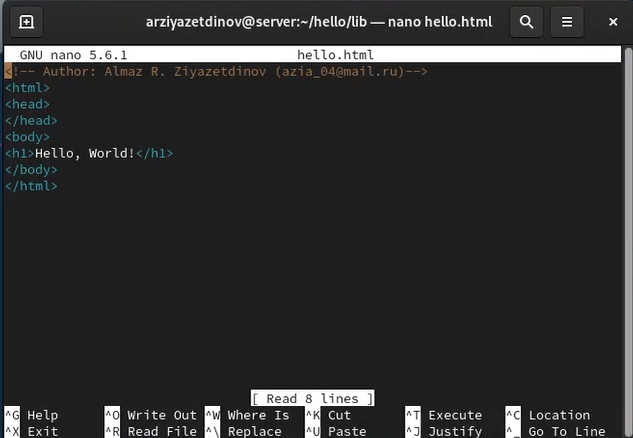


Рис. 21: Файлы hello.html и index.html

## 3.17 Навигация по веткам

Теперь в вашем проекте есть две ветки: git log –all (рис. fig. 22).

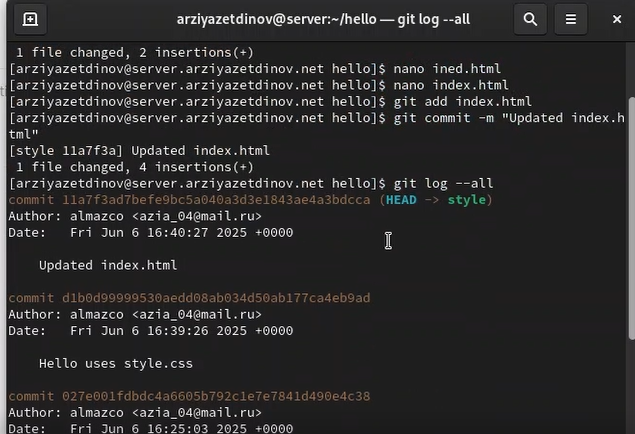


Рис. 22: Логи

Переключение на ветку main: git checkout main и cat lib/hello.html. Сейчас мы находимся на ветке main. Это заметно по тому, что файл hello.html не использует стили style.css. Вернемся к ветке style: git checkout style и cat lib/hello.html. Содержимое lib/hello.html подтверждает, что мы вернулись на ветку style (рис. fig. 23).

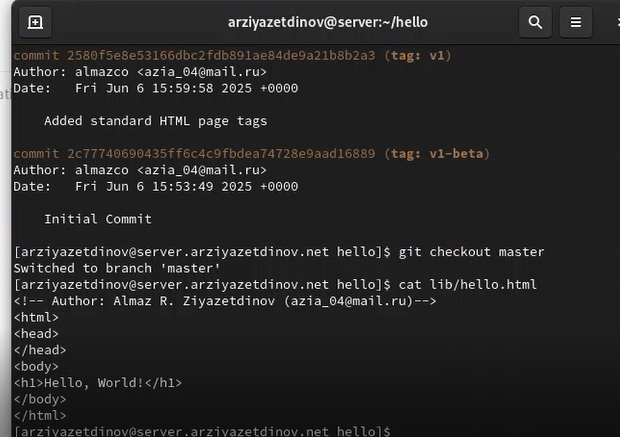


Рис. 23: Навигация по веткам

## 3.18 Изменения в ветке main

Пока вы меняли ветку style, кто-то решил обновить ветку main. Они добавили файл README.md. Создадим файл README в ветке main: git checkout main. Создадим файл README.md: echo “This is the Hello World example from the git tutorial.” > README.md. Сделаем коммит изменений README.md в ветку main: git add README.md и git commit -m “Added README”. Используем следующую лог-команду для просмотра веток и их отличий: git log –graph –all (рис. fig. 24).

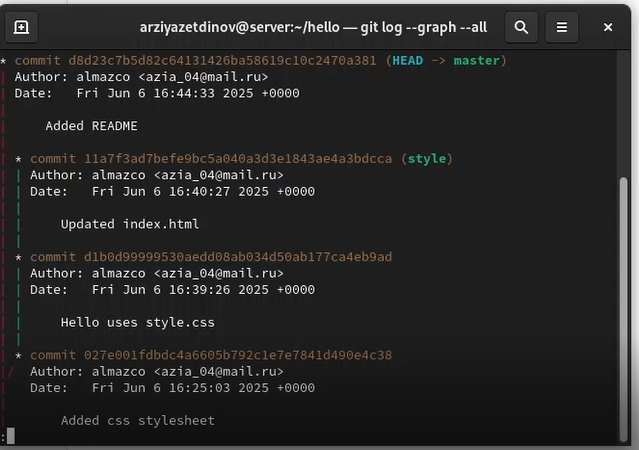


Рис. 24: Изменения в ветке main

## 3.19 Слияние

Давайте вернемся к ветке style и сольем main с style: git checkout style, git merge main, git log –graph –all (рис. fig. 25).

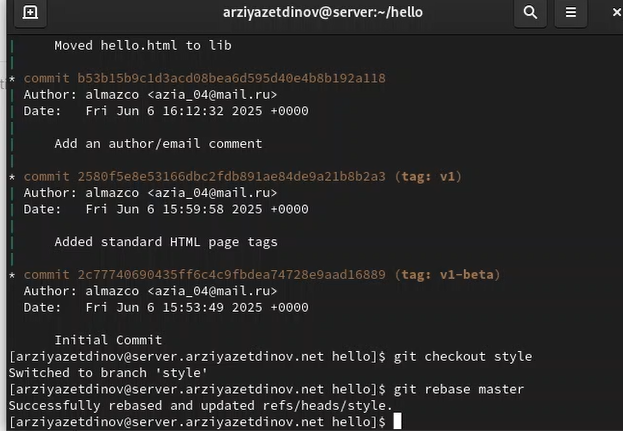


Рис. 25: Слияние main с style

## 3.20 Создание конфликта

Вернемся в main и создадим конфликт. Файл lib/hello.html

Выполним: git add lib/hello.html и git commit -m ‘Life is great’. Просмотр веток: git log –graph –all. После коммита «Added README» ветка main была объединена с веткой style, но в настоящее время в main есть дополнительный коммит, который не был слит с style. Последнее изменение в main конфликтует с некоторыми изменениями в style

## 3.21 Разрешение конфликтов

Слияние main с веткой style. Теперь вернемся к ветке style и попытаемся объединить ее с новой веткой main: git checkout style, git merge main

Откроем lib/hello.html. Первый раздел — версия текущей ветки (style). Второй раздел — версия ветки main.

Решение конфликта. Внесем изменения в lib/hello.html

Сделаем коммит решения конфликта: git add lib/hello.html и git commit -m “Merged main fixed conflict”.

## 3.22 Сброс ветки style

Вернемся на ветке style к точке перед тем, как мы слили ее с веткой main: git checkout style и git log –graph

Мы видим, что коммит «Updated index.html» был последним на ветке style перед слиянием. Давайте сбросим ветку style к этому коммиту: git reset –hard . Поищем лог ветки style. У нас в истории больше нет коммитов слияний: git log –graph –all .

## 3.23 Сброс ветки main

Сброс ветки main: git checkout main, git log –graph (рис. fig. 26).

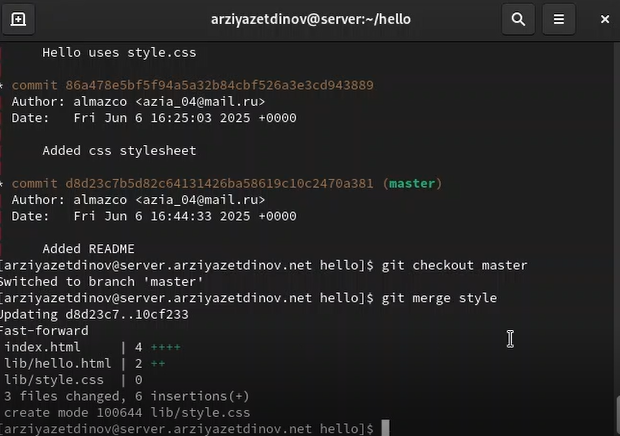


Рис. 26: Ветка main

Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку main к коммиту «Added README»: git reset –hard и git log –graph –all

Мы вернулись в точку до первого слияния и хотим перенести изменения из ветки main в нашу ветку style: git checkout style, git rebase main и git log –graph

## 3.24 Слияние в ветку main

Мы поддерживали соответствие ветки style с веткой main (с помощью rebase), теперь давайте сольем изменения style в ветку main: git checkout main и git merge style . Посмотрим логи. Теперь ветки style и main идентичны.

## 3.25 Клонирование репозиториев

Перейдем в рабочий каталог aamishina и сделаем клон репозитория hello: cd .. pwd ls, git clone hello cloned\_hello, ls. Давайте взглянем на клонированный репозиторий: cd cloned\_hello, ls. Видим список всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib. Просмотрим историю репозитория: git log –all. Видим список всех коммитов в новый репозиторий, и он совпадает с историей коммитов в оригинальном репозитории. Единственная разница в названиях веток

## 3.26 Что такое origin?

Выполним: git remote. Видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Посмотрим более подробную информацию: git remote show origin. Давайте посмотрим на ветки, доступные в нашем клонированном репозитории: git branch. Как видим, в списке только ветка main. Для того, чтобы увидеть все ветки, вводим команду: git branch -a (рис. fig. 27).

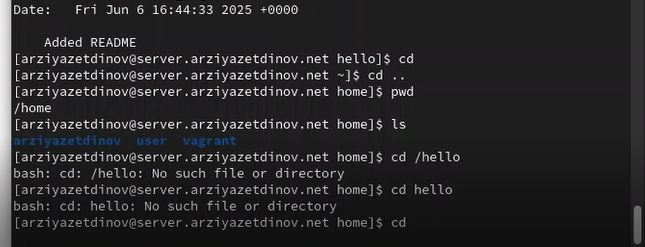


Рис. 27: Информация о репозитории и ветках

## 3.27 Изменение оригинального репозитория

Внесем изменения в оригинальный репозиторий hello: cd ../hello. Внесем изменения в файл README.md: This is the Hello World example from the git tutorial. Сделаем коммит: git add README и git commit -m “Changed README in original repo”

Теперь в оригинальном репозитории есть более поздние изменения, которых нет в клонированной версии.

Извлечение изменений: cd ../cloned\_hello, git fetch, git log –all

Сейчас мы находимся в репозитории cloned\_hello. На данный момент в репозитории есть все коммиты из оригинального репозитория, но они не интегрированы в локальные ветки клонированного репозитория. Выводом является то, что команда git fetch будет извлекать новые коммиты из удаленного репозитория, но не будет сливать их с вашими наработками в локальных ветках. Проверим README.md, он не изменился

## 3.28 Слияние извлеченных изменений

Сольем извлеченные изменения в локальную ветку main: git merge origin/main. Еще раз проверим файл README.md. Сейчас мы видим изменения (рис. fig. **¿fig:053?**).

## 3.29 Добавление ветки наблюдения

Добавим локальную ветку, которая отслеживает удаленную ветку: git branch –track style origin/style, git branch -a, git log –max-count=2. Теперь мы можем видеть ветку style в списке веток и логе

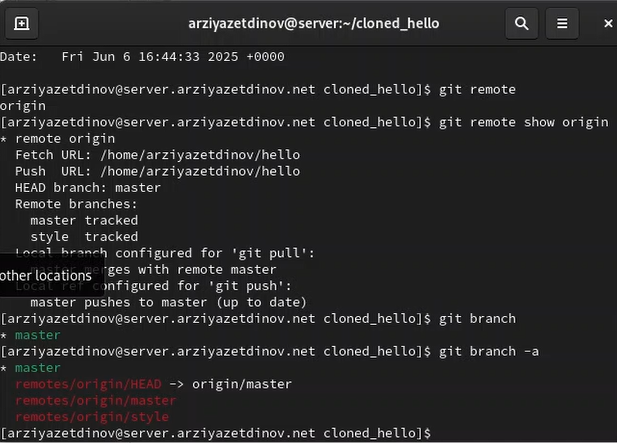


Рис. 28: Добавление ветки наблюдения

## 3.30 Создайте чистый репозиторий

Создадим чистый репозиторий: cd .., git clone –bare hello hello.git, ls hello.git. Сейчас мы находимся в рабочем каталоге. Мы видим, что в репозитории hello.git нет рабочего каталога

## 3.31 Добавление удаленного репозитория

Давайте добавим репозиторий hello.git к нашему оригинальному репозиторию: cd hello и git remote add shared ../hello.git . Создадим изменения для отправки: файл README.md, сделаем его коммит. Теперь отправим изменения в общий репозиторий: git push shared main. Общим называется репозиторий, получающий отправленные нами изменения.

## 3.32 Извлечение общих изменений

Быстро переключаемся в клонированный репозиторий и извлекаем изменения, только что отправленные в общий репозиторий: cd ../cloned\_hello. Сейчас мы находимся в репозитории cloned\_hello: git remote add shared ../hello.git, git branch –track shared main, git pull shared main, cat README.md (рис. fig. 29).

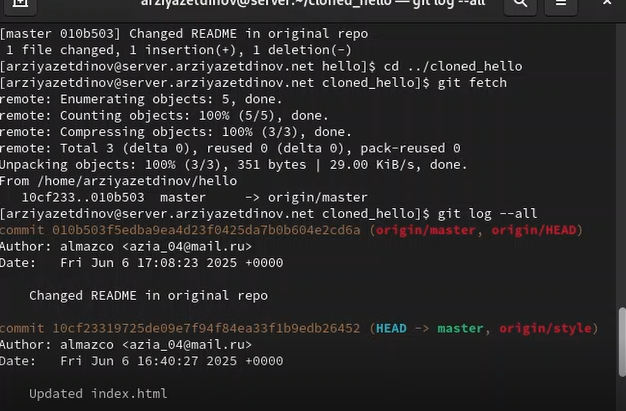


Рис. 29: Извлечение общих изменений

# 4 Выводы

В ходе выполнения данной лабораторной работы я приобрела практические навыки работы с системой управления версиями Git.

# Список литературы