

# **Лабораторная работа №1**

**Работа с git**

Гузева Ирина Николаевна

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>3</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
3.1	Подготовка . . . . .	7
3.2	Создание проекта . . . . .	7
3.3	Внесение изменений . . . . .	8
3.4	История . . . . .	9
3.5	Получение старых версий . . . . .	9
3.6	Создание тегов версий . . . . .	10
3.7	Отмена локальных изменений (до индексации) . . . . .	10
3.8	Отмена проиндексированных изменений (перед коммитом) . . . . .	11
3.9	Отмена коммитов . . . . .	12
3.10	Удаление коммитов из ветки . . . . .	13
3.11	Удаление тега oops . . . . .	13
3.12	Изменение предыдущего коммита . . . . .	13
3.13	Перемещение файлов . . . . .	14
3.14	Подробнее о структуре . . . . .	14
3.15	Git внутри: Каталог .git . . . . .	15
3.16	Работа непосредственно с объектами git . . . . .	15
3.17	Создание ветки . . . . .	16
3.18	Навигация по веткам . . . . .	17
3.19	Изменения в ветке master . . . . .	17
3.20	Слияние . . . . .	17
3.21	Создание конфликта . . . . .	17
3.22	Разрешение конфликтов . . . . .	17
3.23	Сброс ветки style . . . . .	18
3.24	Сброс ветки master . . . . .	18
3.25	Перебазирование . . . . .	18
3.26	Слияние в ветку master . . . . .	19
3.27	Клонирование репозитория . . . . .	19
3.28	Что такое origin? . . . . .	19
3.29	Удаленные ветки . . . . .	20
3.30	Изменение оригинального репозитория . . . . .	20
3.31	Слияние извлеченных изменений . . . . .	20
3.32	Добавление ветки наблюдения . . . . .	21

3.33 Создание чистого репозитория . . . . .	21
3.34 Отправка и извлечение изменений . . . . .	21
<b>4 Выводы</b>	<b>23</b>
<b>Список литературы</b>	<b>24</b>

## Список иллюстраций

3.1	Создание репозитория . . . . .	7
3.2	Внесение нескольких изменений в файл . . . . .	9
3.3	Просмотр разных версий репозитория . . . . .	9
3.4	Переключение по имени тега и просмотр доступных тегов . . . .	10
3.5	Добавления нежелательного комментария . . . . .	11
3.6	Отмена коммитов . . . . .	12
3.7	Изменение предыдущего коммита . . . . .	13
3.8	index.html . . . . .	14
3.9	Каталог .git . . . . .	15
3.10	Работа непосредственно с объектами git . . . . .	16
3.11	Редактирование файла . . . . .	16
3.12	Просмотр имени по умолчанию удаленного репозитория . . . . .	20
3.13	Создание чистого репозитория . . . . .	21
3.14	Извлечение изменений . . . . .	22

# 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:bash?](#)].

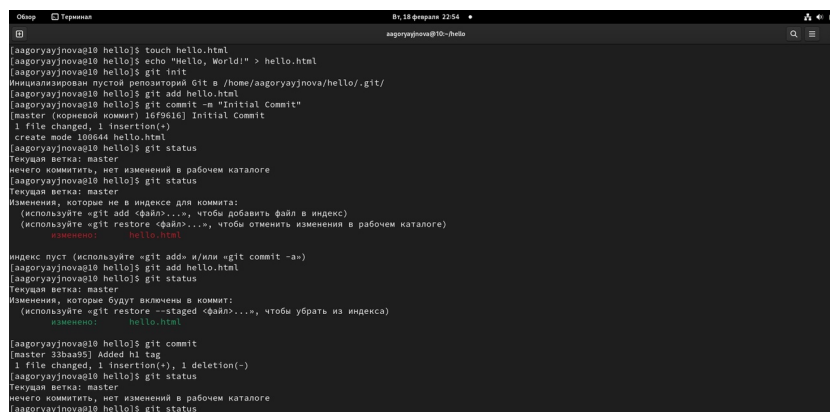
## 3 Выполнение лабораторной работы

### 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. fig. 3.1).



```
Обзор Терминал 01.10.Февраль 22:54
aagoryajnova@10:~$ touch hello.html
aagoryajnova@10:~$ echo "Hello, World!" > hello.html
aagoryajnova@10:~$ git init
Инициализирован пустой репозиторий Git в /home/aagoryajnova/hello/.git/
aagoryajnova@10:~$ git add hello.html
[аgoryajnova@10:~$ git commit -m "Initial Commit"
[master (корневой коммит) 16f9616] Initial Commit
1 file changed, 1 insertion(+)
create mode 100644 hello.html
aagoryajnova@10:~$ git status
текущая ветка: master
ничего коммитить, нет изменений в рабочем каталоге
aagoryajnova@10:~$ git status
текущая ветка: master
Изменения, которые не в индексе для коммита:
(используйте «git add <файл>...», чтобы добавить файл в индекс)
(используйте «git restore <файл>...», чтобы отменить изменения в рабочем каталоге)
изменено:   hello.html
индекс пуст (используйте «git add -A» или «git commit -a»)
aagoryajnova@10:~$ git add hello.html
aagoryajnova@10:~$ git status
текущая ветка: master
Изменения, которые будут включены в коммит:
(используйте «git restore --staged <файл>...», чтобы убрать из индекса)
индекс:      hello.html
aagoryajnova@10:~$ git commit
[master 13baa95] Added hi tag
1 file changed, 1 insertion(+), 1 deletion(-)
aagoryajnova@10:~$ git status
текущая ветка: master
ничего коммитить, нет изменений в рабочем каталоге
aagoryajnova@10:~$ git status
```

Рис. 3.1: 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 репозиторий. 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. fig. 3.2). 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.



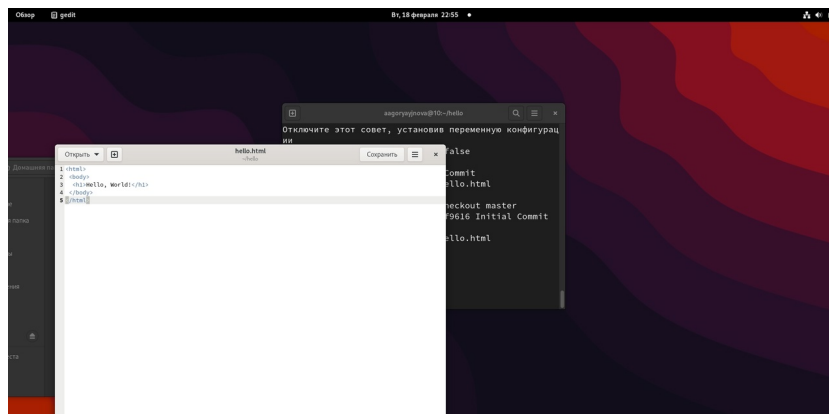


Рис. 3.2: Внесение нескольких изменений в файл

## 3.4 История

Получим список произведённых изменений в стандартном 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 версии и просмотрим файл `hello.html`, действительно, увидим первую версию. Затем вернемся к последней версии в ветке `master` и вновь посмотрим на файл (рис. fig. 3.3).

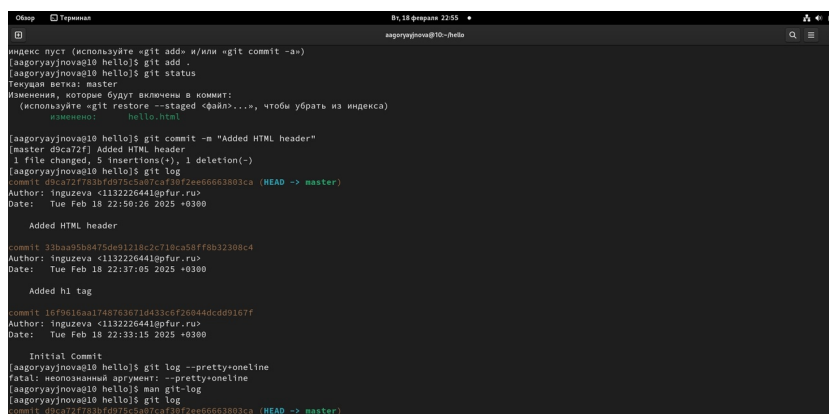


Рис. 3.3: Просмотр разных версий репозитория

## 3.6 Создание тегов версий

Назовем текущую версию страницы hello первой (v1). Создадим тег первой версии и используем его для того чтобы вернуться 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. fig. 3.4).

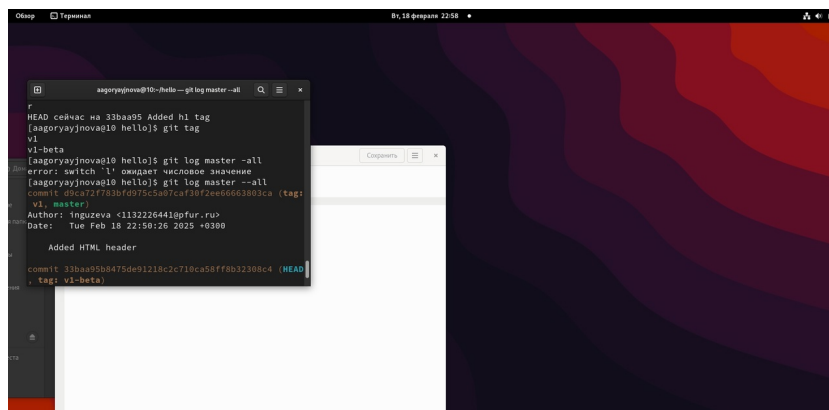


Рис. 3.4: Переключение по имени тега и просмотр доступных тегов

## 3.7 Отмена локальных изменений (до индексации)

Убедимся, что мы находимся на последнем коммите ветки master и внесем изменение в файл hello.html в виде нежелательного комментария (рис. fig. 3.5). Затем проверим статус, увидим, что изменения ещё не проиндексированы. Используем команду `git checkout` для переключения версии файла hello.html в репозитории.

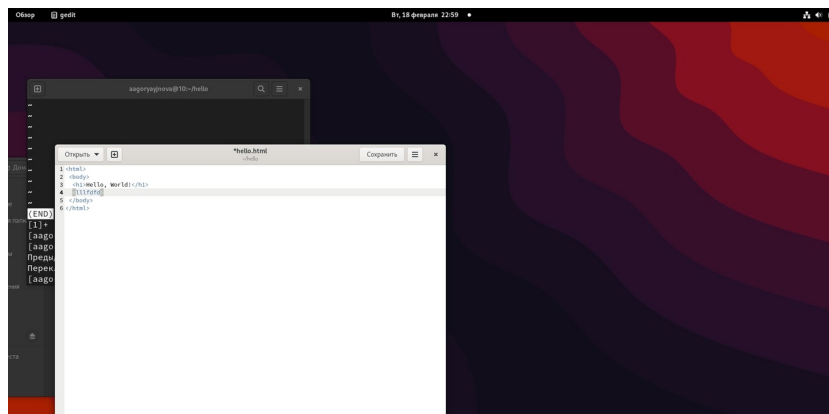


Рис. 3.5: Добавления нежелательного комментария

### 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 логa показывает нежелательные и отмененные коммиты в наш репозиторий(рис. fig. 3.6).

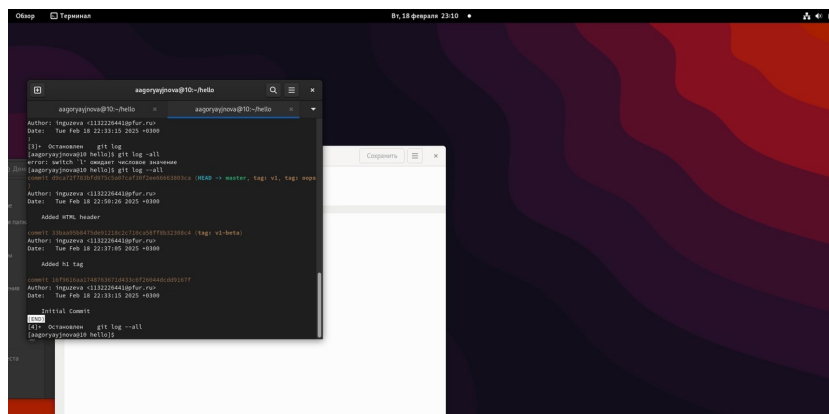


Рис. 3.6: Отмена коммитов

## 3.10 Удаление коммитов 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 оорс

Удалим тег `оорс` 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, обновиим страницу `hello`, включив в неё почту. Чтобы у нас остался один коммит, а не два, изменим последний с помощью опции `-amend`, теперь в логах отображается последняя версия коммита (рис. fig. 3.7).

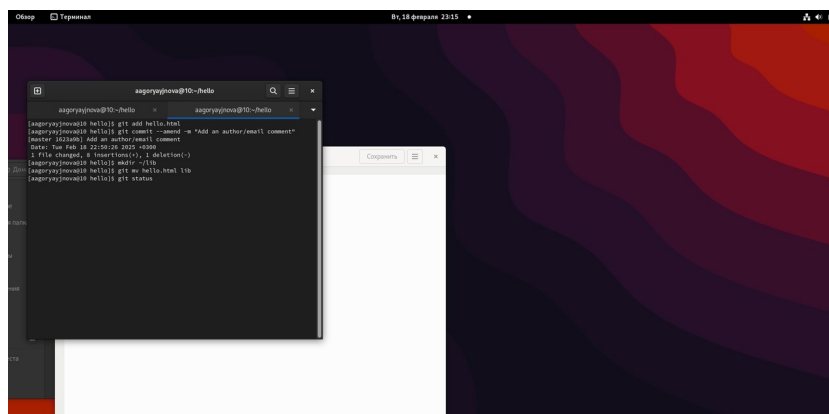


Рис. 3.7: Изменение предыдущего коммита

## 3.13 Перемещение файлов

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

## 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).

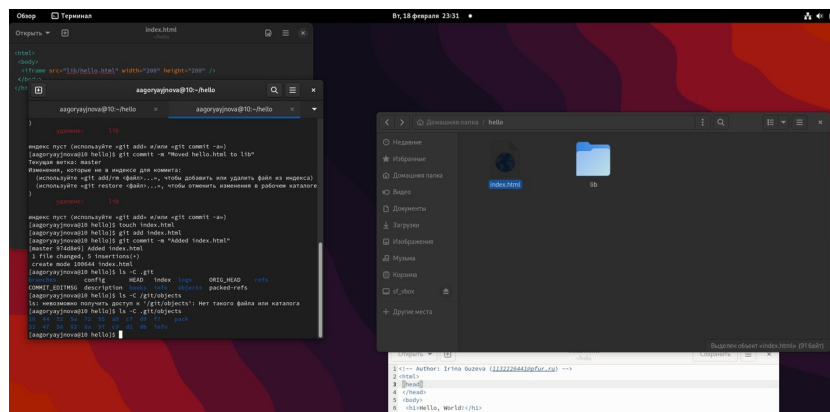


Рис. 3.8: `index.html`

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

## 3.15 Git внутри: Каталог .git

Посмотрим 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 содержимое файла HEAD, который содержит ссылку на текущую ветку, в данный момент это ветка master (рис. fig. 3.9).

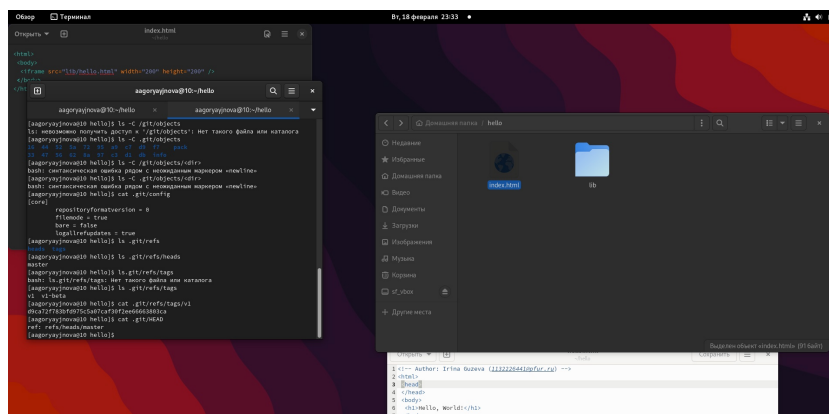


Рис. 3.9: Каталог .git

## 3.16 Работа непосредственно с объектами git

Найдем последний коммит и выведем его с помощью SHA1 хэша. Затем посмотрим дерево каталогов, ссылка на который идёт в последнем коммите, выведем каталог lib и файл hello.html (рис. fig. 3.10).





## 3.18 Навигация по веткам

Посмотрим все логи. Переключимся обратно на основную ветку и посмотрим содержимое файла `ib/hello.html`, заметим, что он не использует стили, также посмотрим 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, внося 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.

Поищем 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 репозитория

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 посмотрим 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. fig. 3.12).

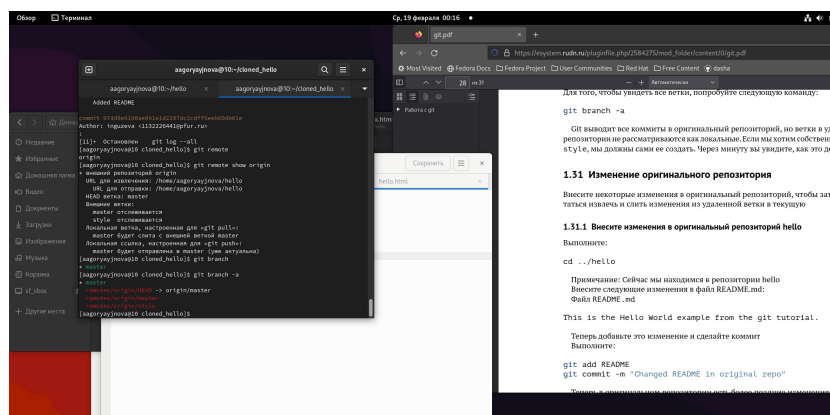


Рис. 3.12: Просмотр имени по умолчанию удаленного репозитория

## 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. fig. 3.13).

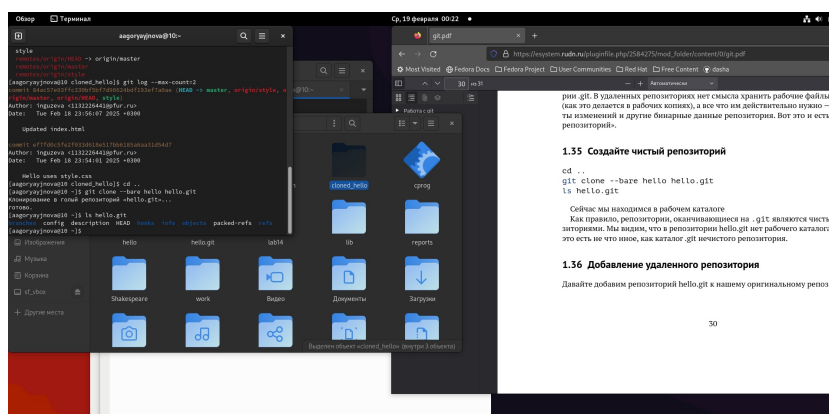


Рис. 3.13: Создание чистого репозитория

## 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. fig. 3.14).

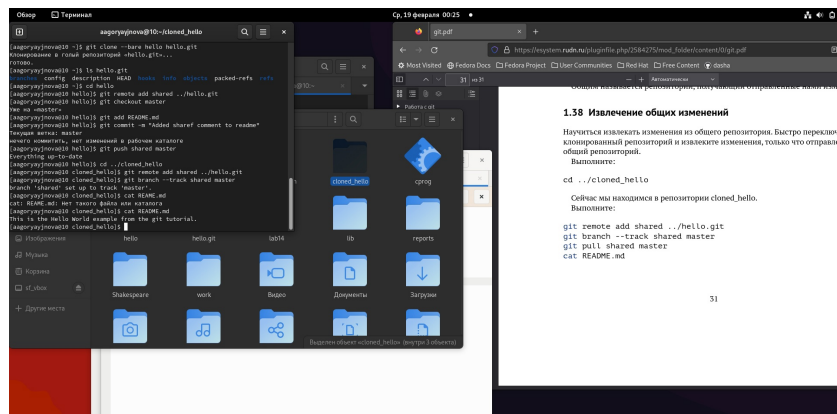


Рис. 3.14: Извлечение изменений

## 4 Выводы

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

## **Список литературы**