## Front matter

title: "Лабораторная 1" subtitle: "Работа с git" author: "Александра Болотина"

## Generic otions

lang: ru-RU toc-title: "Содержание"

## Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

## Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt

## I18n polyglossia

polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english

## I18n babel

babel-lang: russian babel-otherlangs: english

## Fonts

mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT Mono mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions: Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase,Scale=0.9

## Biblatex

biblatex: true biblio-style: "gost-numeric" biblatexoptions:

parentracker=true backend=biber hyperref=auto language=auto autolang=other\* citestyle=gost-numeric

## Pandoc-crossref LaTeX customization

figureTitle: "Рис." tableTitle: "Таблица" listingTitle: "Листинг" lofTitle: "Список иллюстраций" lotTitle: "Список таблиц" lolTitle: "Листинги"

## Misc options

indent: true header-includes:

\usepackage{indentfirst}

\usepackage{float} # keep figures where there are in the text

\floatplacement{figure}{H} # keep figures where there are in the text

# Цель работы

Научиться работать с git.

# Выполнение лабораторной работы

### Установка имени и электронной почты

Выполните следующие команды, чтобы git узнал ваше имя и электронную почту.



git config --global user.name "Your Name"

git config --global user.email "[your\_email@whatever.com](mailto:your_email@whatever.com)"

### Параметры установки окончаний строк

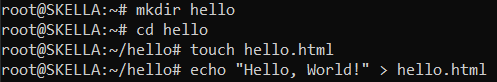


**Установка отображения unicode**

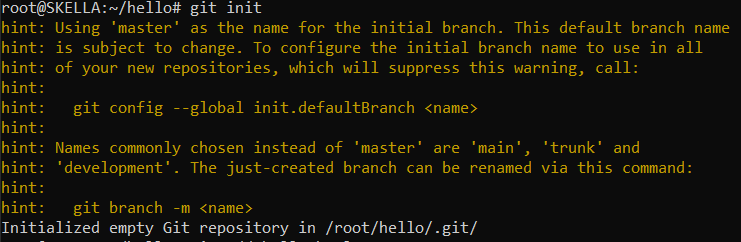


**Создание проекта.**

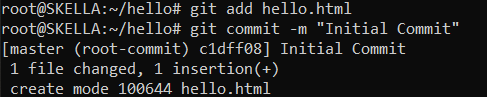
### Создайте страницу «Hello, World»



**Создание репозитория**



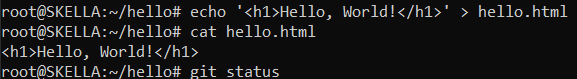
### Добавление файла в репозиторий



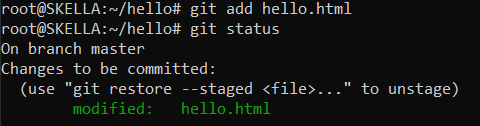
**Проверка состояние репозитория**



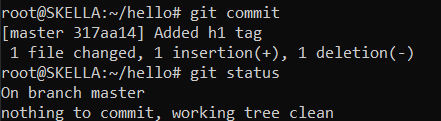
### Измените страницу «Hello, World»



**Индексация изменений**

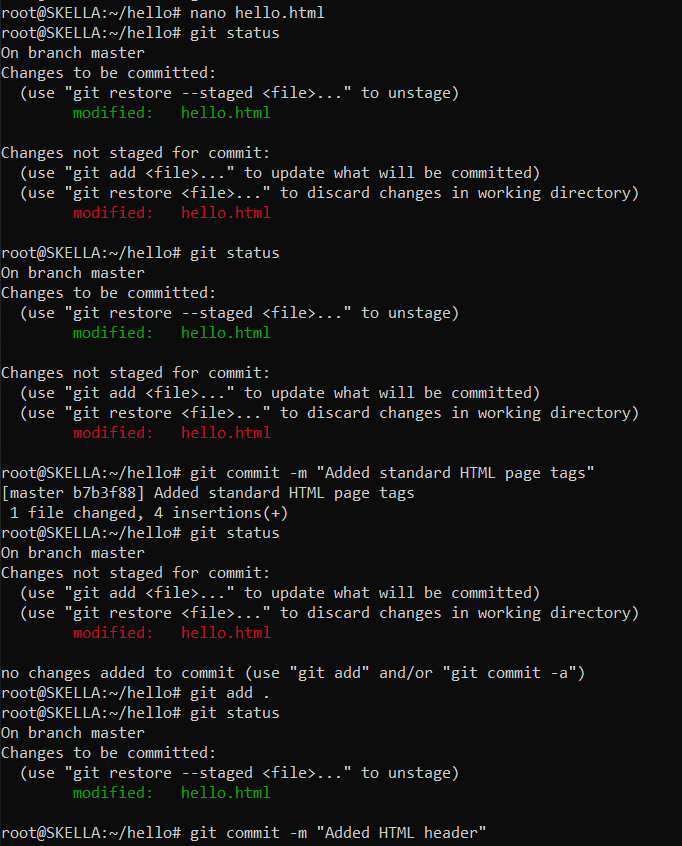


### Коммит изменений



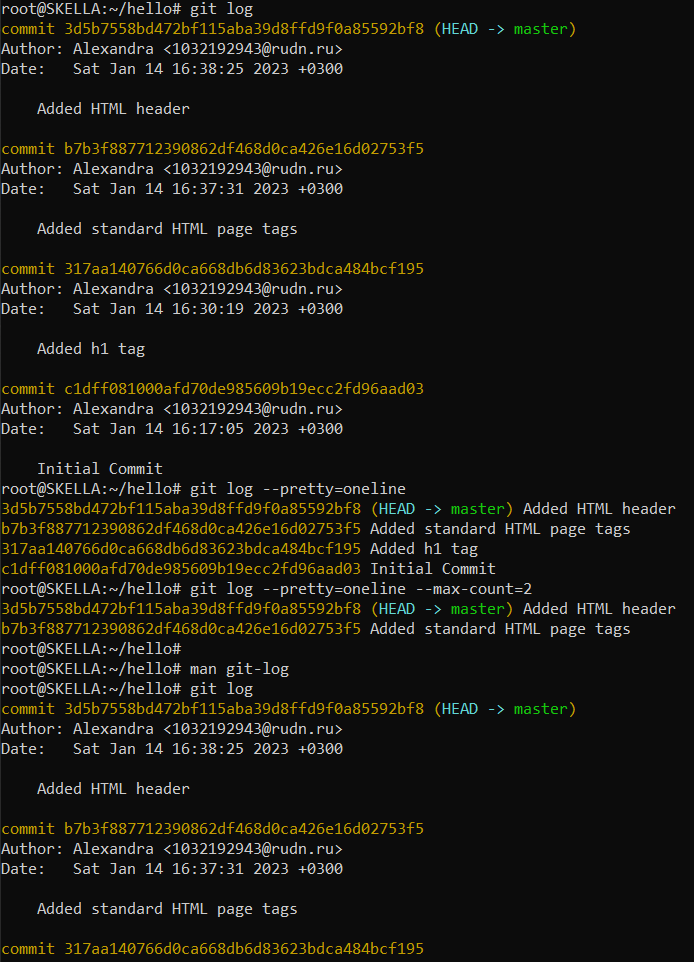
Добавьте стандартные теги страницы



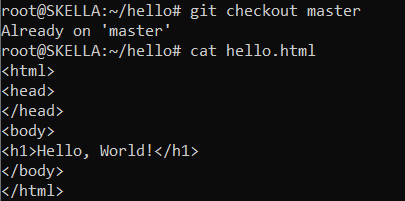


### История

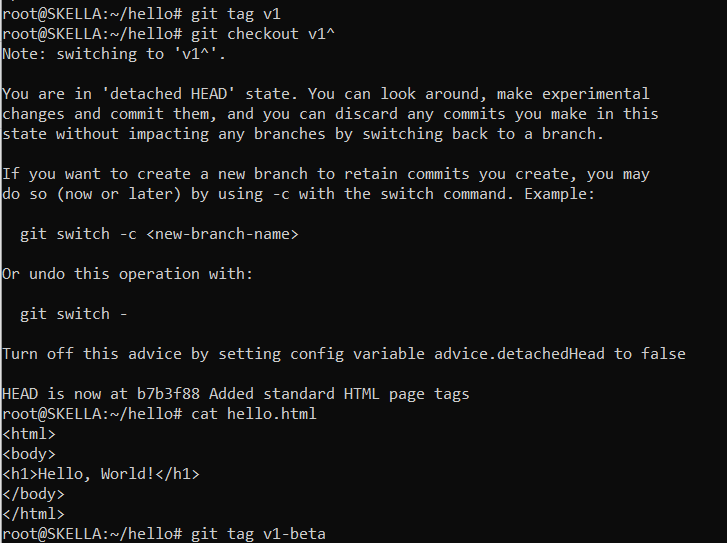
Получим список произведенных изменений:



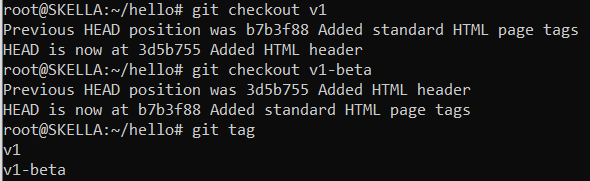
### Получение старых версий



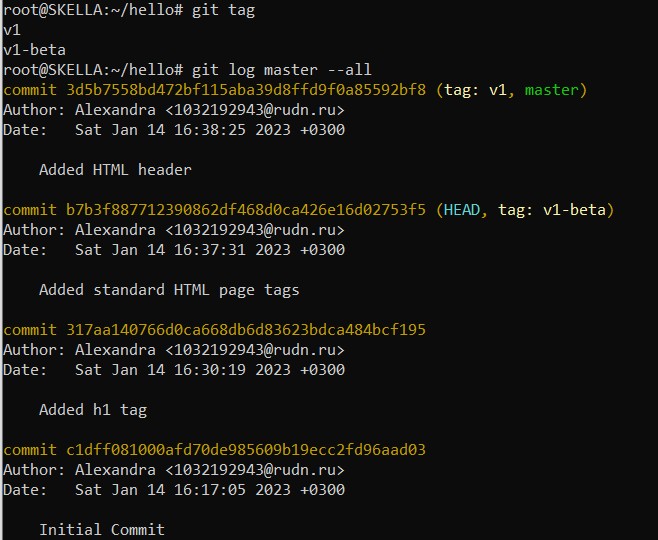
**Создание тегов версий**



### Переключение по имени тега



**Просмотр тегов с помощью команды tag**



## Отмена локальных изменений (до индексации)

### Переключитесь на ветку master

Убедитесь, что вы находитесь на последнем коммите ветки master, прежде чем продолжить работу.



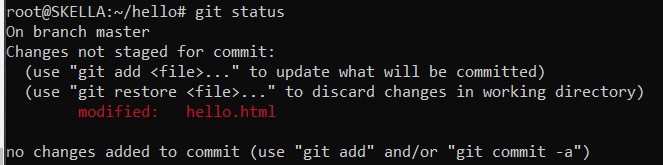
### Измените hello.html

Иногда случается, что вы изменили файл в рабочем каталоге, и хотите отменить последние коммиты. С этим справится команда git checkout. Внесите изменение в файл hello.html в

виде нежелательного комментария.

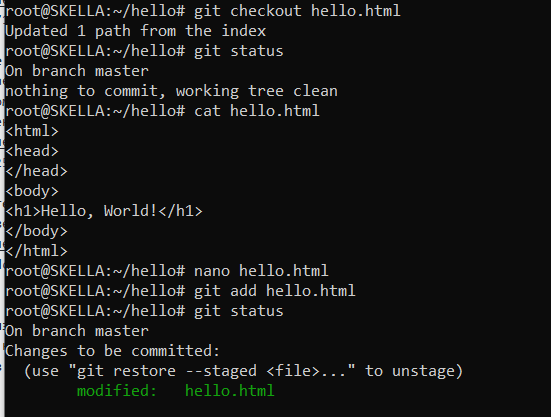
### Проверьте состояние

Сначала проверьте состояние рабочего каталога.



### Отмена изменений в рабочем каталоге

Используйте команду git checkout для переключения версии файла hello.html в репозитории.



## Отмена проиндексированных изменений (перед коммитом)

### Измените файл и проиндексируйте изменения

Внесите изменение в файл hello.html в виде нежелательного комментария

### Проверьте состояние

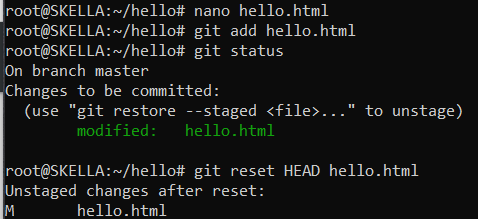
Проверьте состояние нежелательного изменения. Состояние показывает, что изменение было проиндексировано и готово к коммиту.

### Выполните сброс буферной зоны

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.

### Переключитесь на версию коммита

Наш рабочий каталог опять чист



## Отмена коммитов

### Измените файл и сделайте коммит

Измените файл hello.html Выполните:

git add hello.html

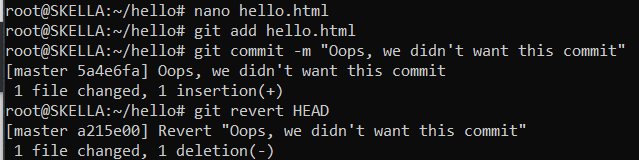
git commit -m "Oops, we didn't want this commit"

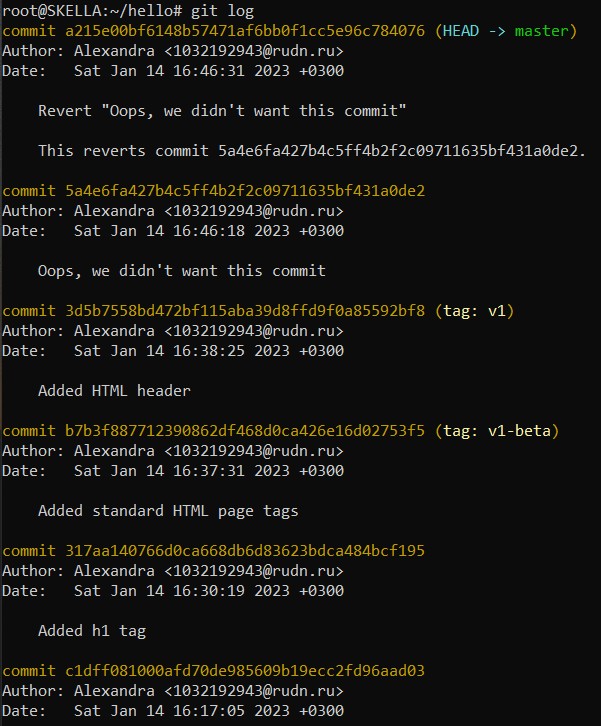
### Сделайте коммит с новыми изменениями, отменяющими предыдущие

Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом.

### Проверьте лог

Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий.





## Удаление коммиттов из ветки

### Проверьте нашу историю

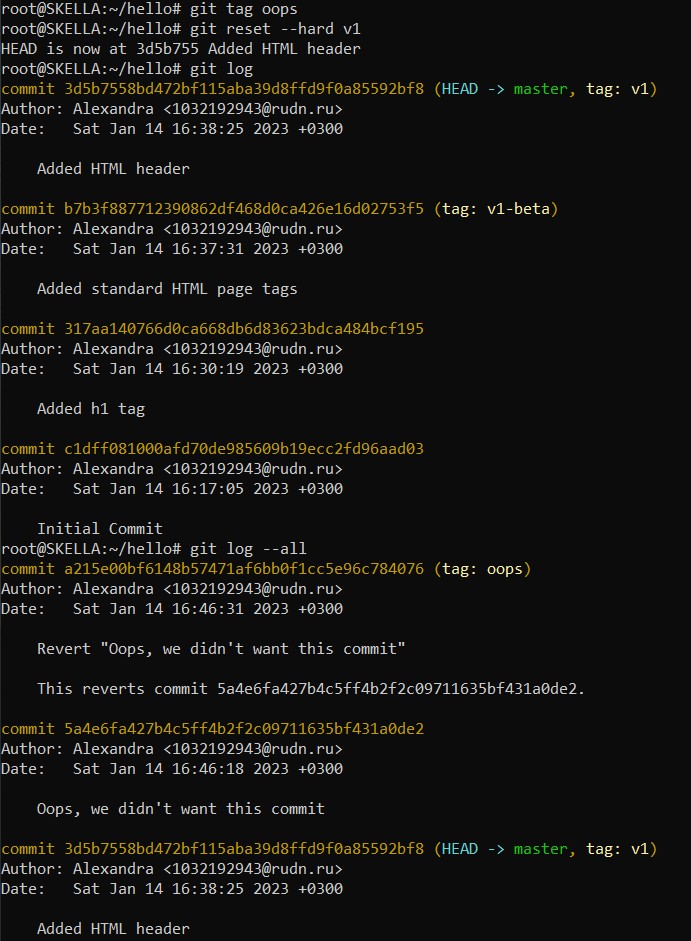
сделаем быструю проверку нашей истории коммитов

### Для начала отметьте эту ветку

Но прежде чем удалить коммиты, давайте отметим последний коммит тегом, чтобы потом можно было его найти.

### Сброс коммитов к предшествующим коммиту Oops

Глядя на историю лога, мы 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).



### Ничего никогда не теряется

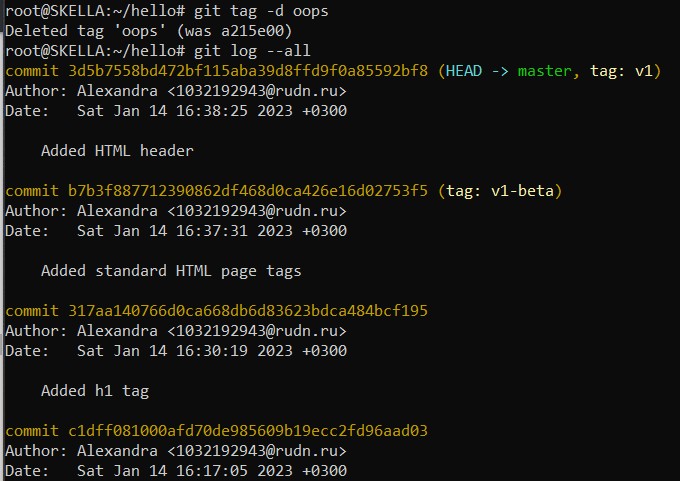
Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории



## Удаление тега oops

### Удаление тега oops

Тег oops свою функцию выполнил. Давайте удалим его и коммиты, на которые он ссылался, сборщиком мусора. Тег «oops» больше не будет отображаться в репозитории.



## Внесение изменений в коммиты

### Измените страницу, а затем сделайте коммит

Добавьте в страницу комментарий автора

### Необходим email

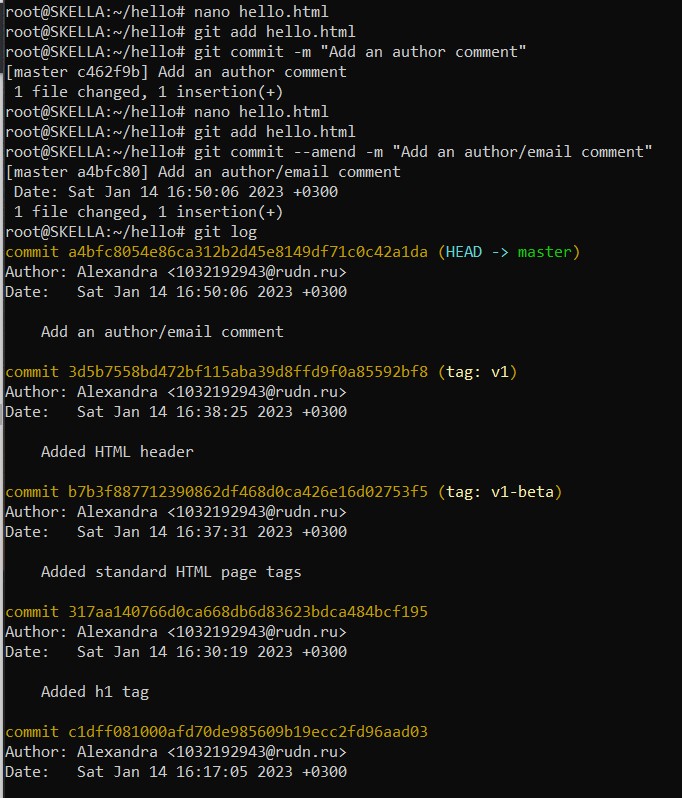
Обновите страницу hello, включив в нее email.

### Измените предыдущий коммит

Мы действительно не хотим создавать отдельный коммит только ради электронной почты. Давайте изменим предыдущий коммит, включив в него адрес электронной почты.

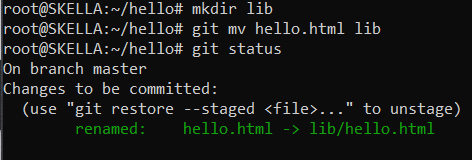
### Просмотр истории

Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email»



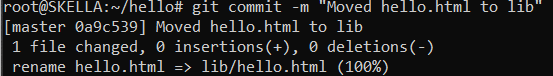
### Перемещение файлов

Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесем страницу в каталог lib.



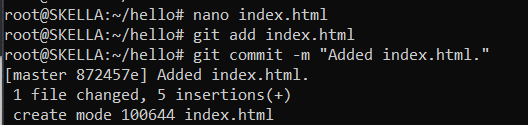
### Коммит в новый каталог

Давайте сделаем коммит этого перемещения:



## Подробнее о структуре

**Добавление index.html**



## Git внутри: Каталог .git

### Каталог .git

Это каталог, в котором хранится вся информация git.

### База данных объектов

Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося в git.

### Углубляемся в базу данных объектов

Смотрим в один из каталогов с именем из 2 букв. Вы увидите файлы с именами из 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Они сжаты и закодированы, поэтому просмотр их содержимого нам мало чем поможет.

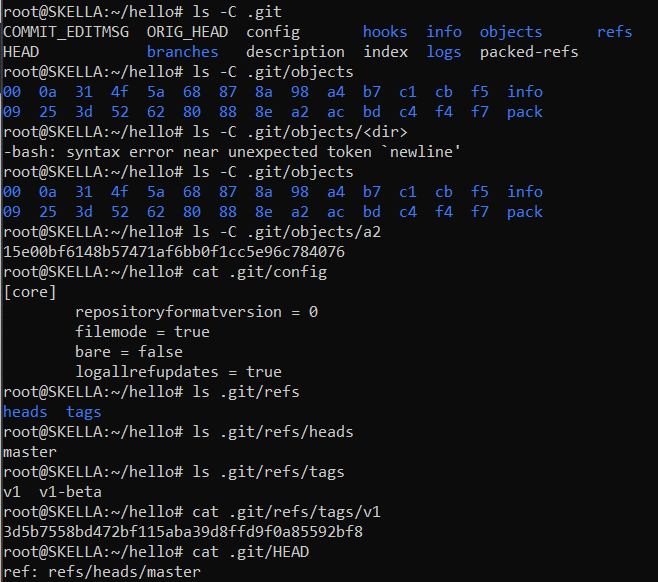
### Config File

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.

### Ветки и теги

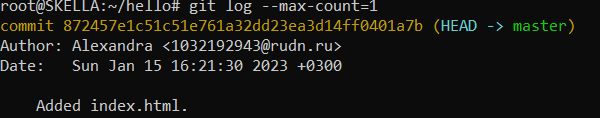
Вы должны узнавать файлы в подкаталоге тегов. Каждый файл соответствует тегу, ранее созданному с помощью команды git tag. Его содержание — это всего лишь хэш коммита, привязанный к тегу. Каталог heads практически аналогичен, но используется для веток, а не тегов. На данный момент у нас есть только одна ветка, так что все, что вы увидите в этом каталоге – это ветка master.

### Файл HEAD

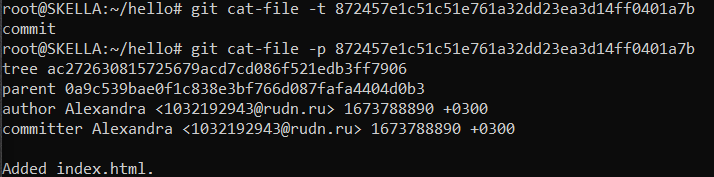
Файл HEAD содержит ссылку на текущую ветку, в данный момент это должна быть ветка master.

## Работа непосредственно с объектами git

### Поиск последнего коммита



**Вывод последнего коммита с помощью SHA1 хэша**



### Поиск дерева

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

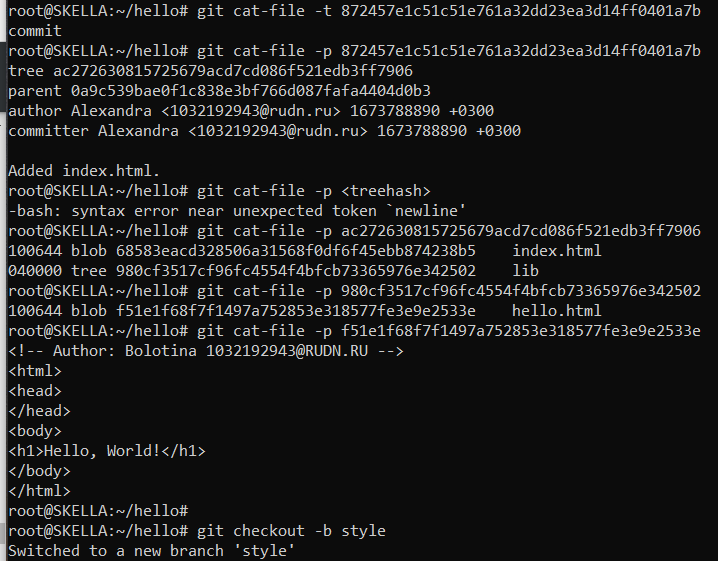
Выполните:

git cat-file -p <libhash>

### Вывод файла hello.html

Выполните:

git cat-file -p <hellohash>



## Создание ветки

### Создайте ветку

Назовем нашу новую ветку «style».

### Добавьте файл стилей style.css

Выполните:

touch lib/style.css Файл lib/style.css: h1 {

color: red;

}

Выполните:

git add lib/style.css

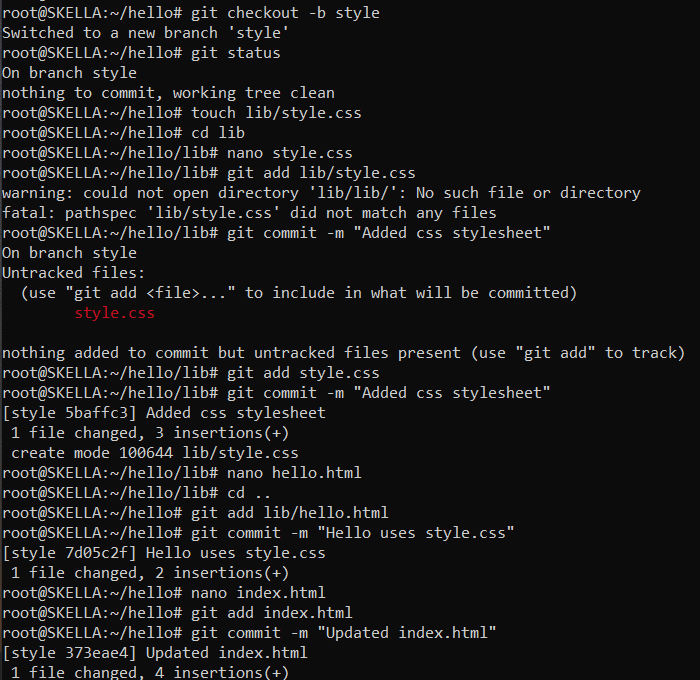
git commit -m "Added css stylesheet"

### Измените основную страницу

Измените основную страницу

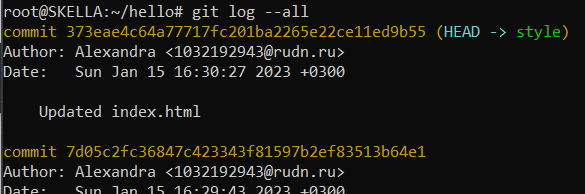
### Измените index.html

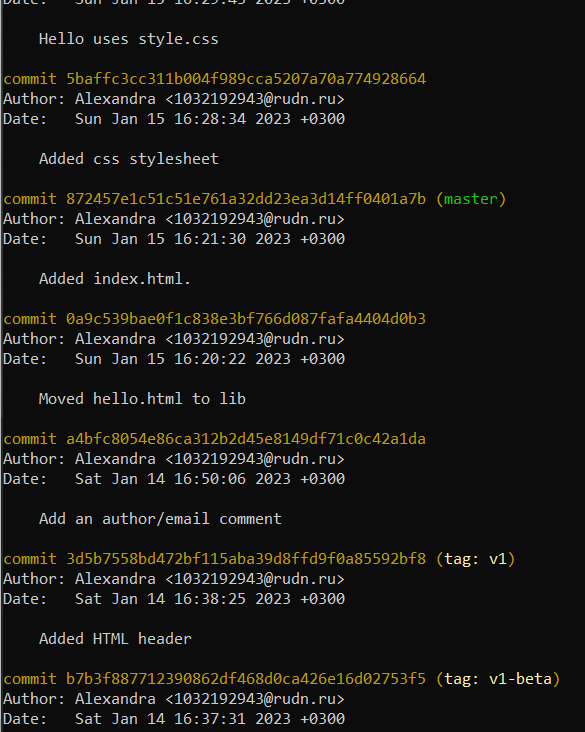
Обновите файл index.html, чтобы он тоже использовал style.css



## Навигация по веткам

Теперь в проекте есть две ветки



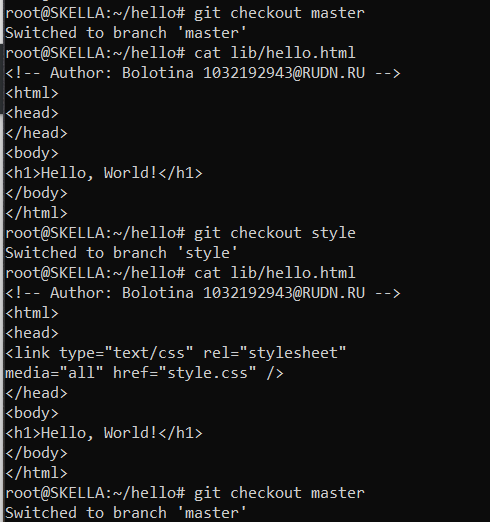


## Переключение на ветку master

Используйте команду git checkout для переключения между ветками

## Вернемся к ветке style

Содержимое lib/hello.html подтверждает, что мы вернулись на ветку style



**Изменения в ветке master**

## Создайте файл README в ветке master

Выполните: git checkout master Создайте файл README.md echo "This is the Hello World





example from the git tutorial." > README.md

## Сделайте коммит изменений README.md в ветку master

Выполните:

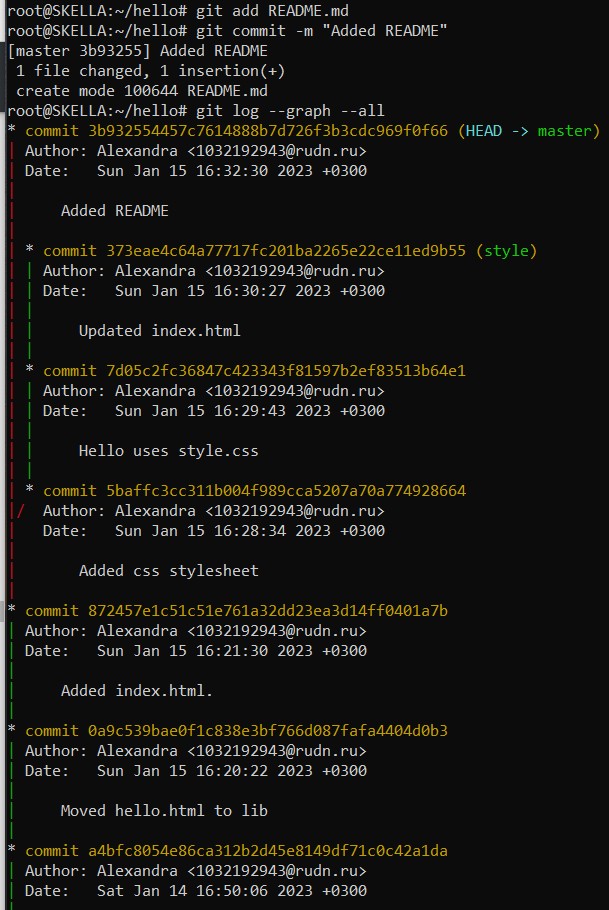
git add README.md

git commit -m "Added README"

## Просмотрите текущие ветки

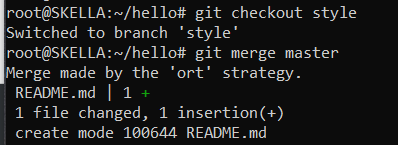
Добавление опции --graph в git log вызывает построение дерева коммитов с помощью простых ASCII символов. Мы видим обе ветки (style и master), и то, что ветка master является текущей HEAD. Общим предшественником обеих веток является коммит «Added index.html». Опция --all гарантированно означает, что мы видим все ветки. По умолчанию

показывается только текущая ветка.



## Слияние

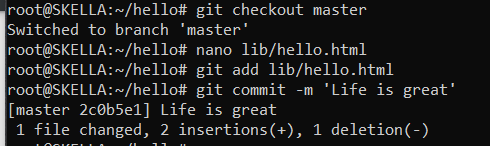
Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и



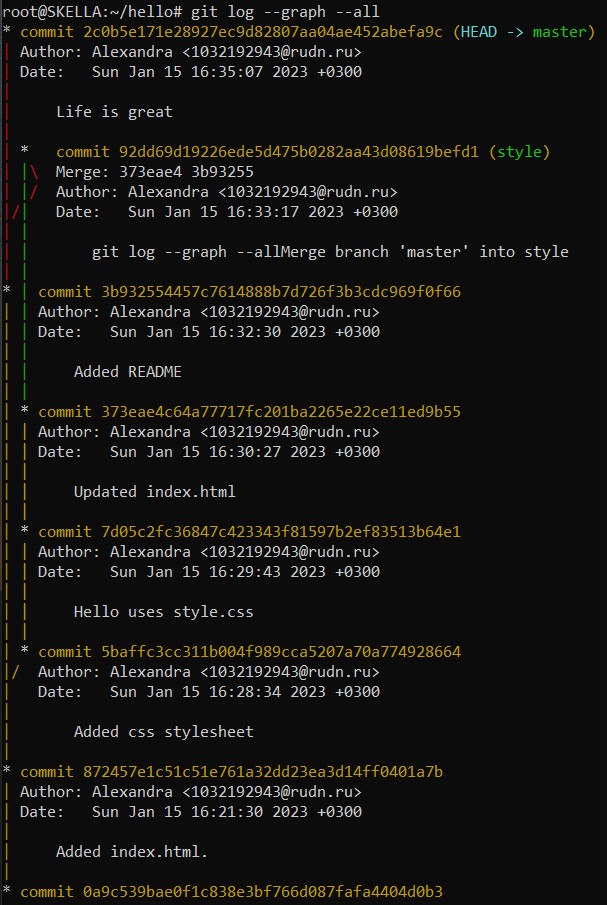
сольем master с style.

## Создание конфликта

### Вернитесь в master и создайте конфликт

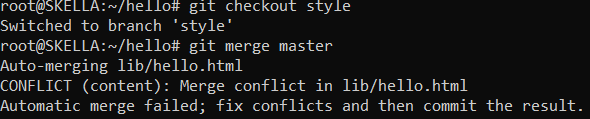


**Просмотр веток**



## Разрешение конфликтов

**Слияние master с веткой style**

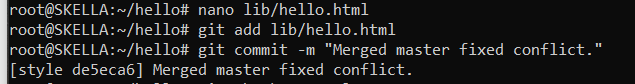


## Решение конфликта

необходимо вручную разрешить конфликт

## Сделайте коммит решения конфликта

Выполните:



git add lib/hello.html

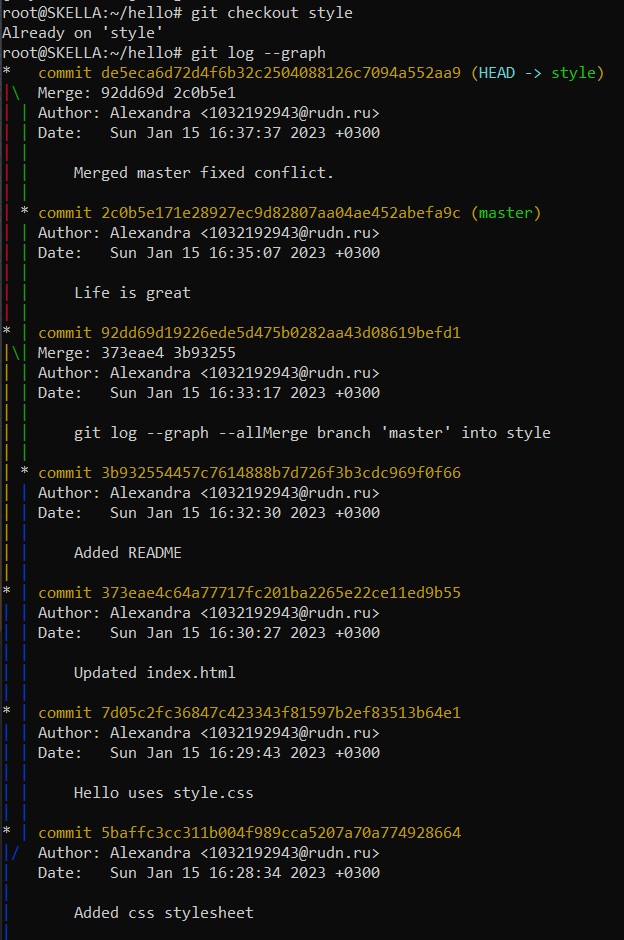
git commit -m "Merged master fixed conflict."

## Сброс ветки style

### Сброс ветки style

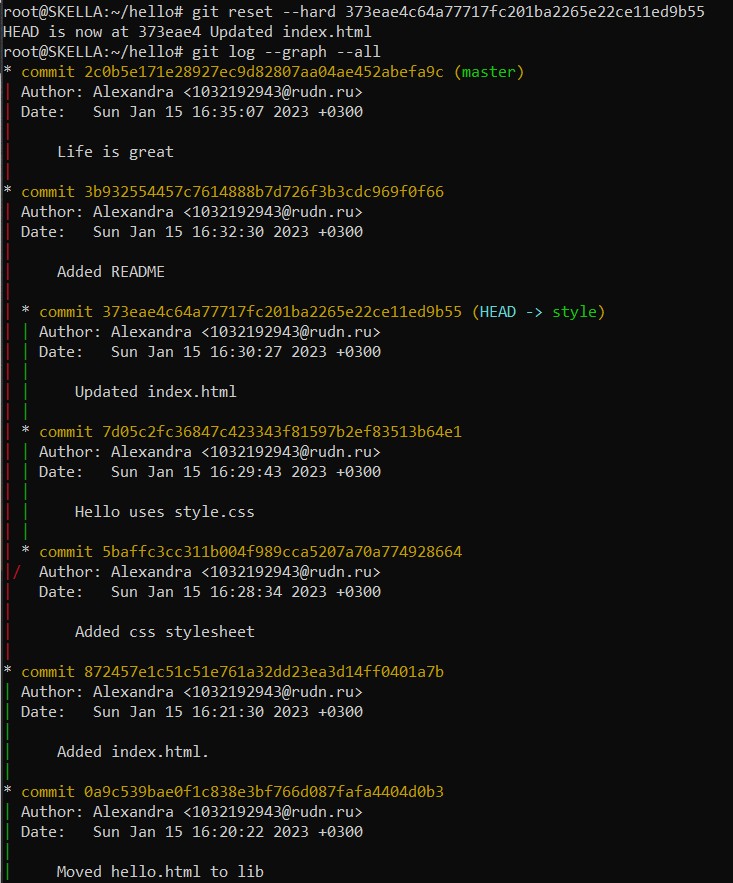
Вернемся на ветке style к точке перед тем, как мы слили ее с веткой master. Мы можем сбросить ветку к любому коммиту. По сути, это изменение указателя ветки на любую точку дерева коммитов. В этом случае мы хотим вернуться в ветке style в точку перед слиянием

с master. Нам необходимо найти последний коммит перед слиянием.

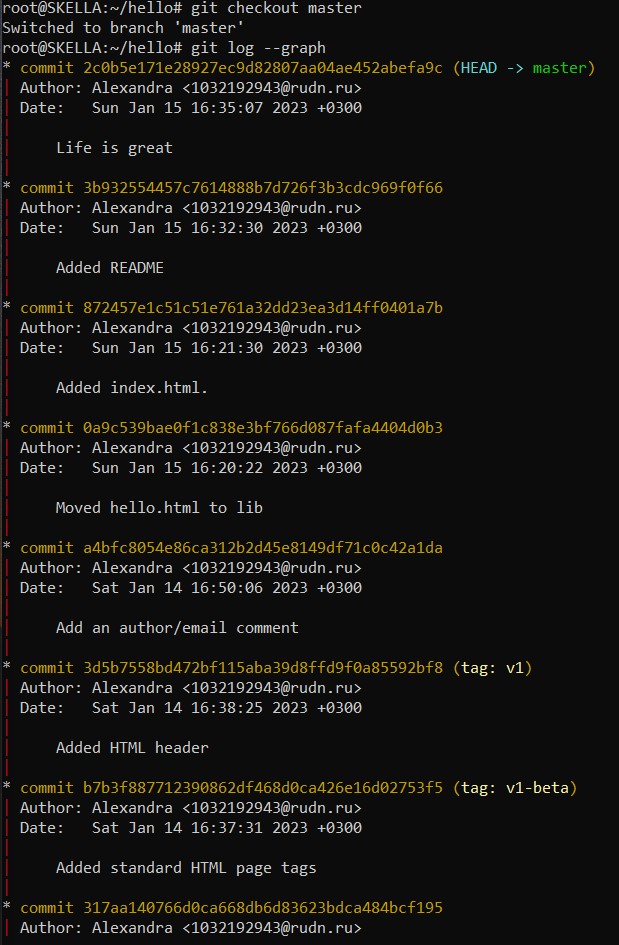


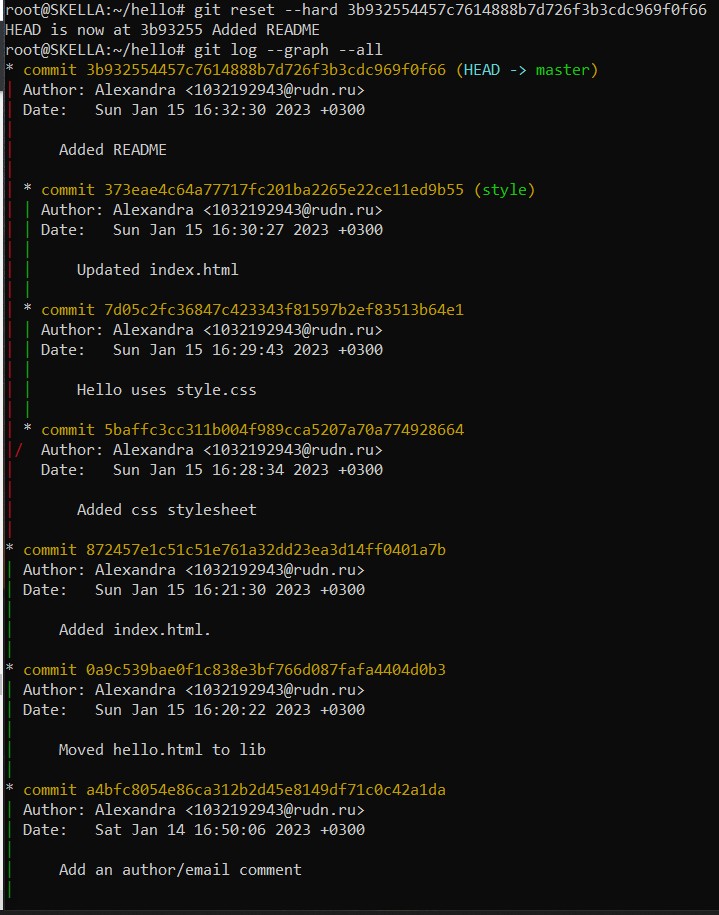
### Проверьте ветку

Поищите лог ветки style. У нас в истории больше нет коммитов слияний. Выполните:

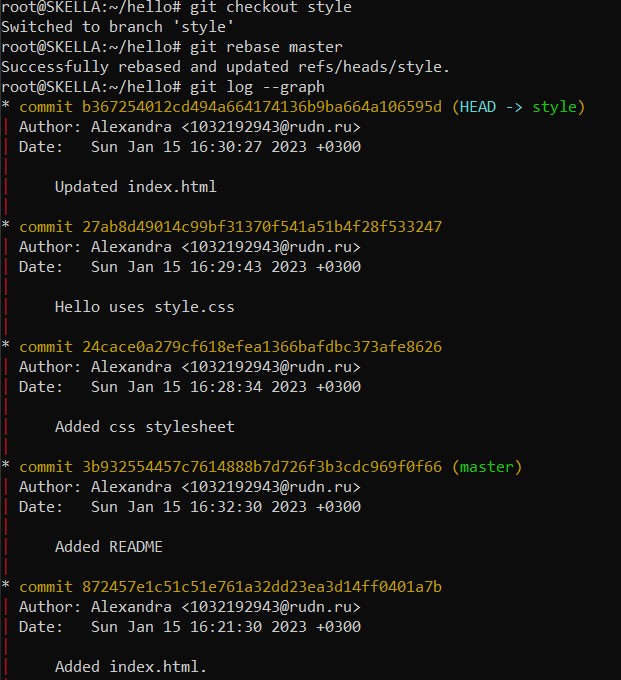


## Сброс ветки master





**Перебазирование**

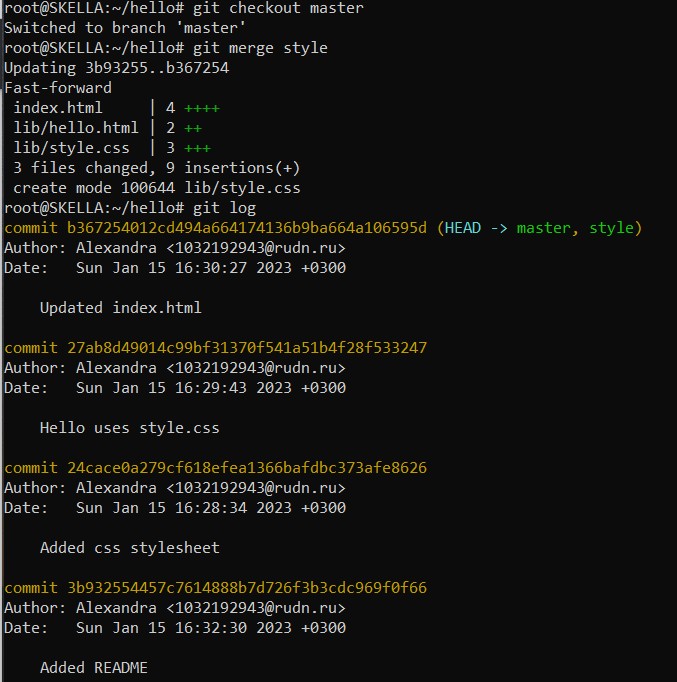


## Слияние в ветку master

### Слияние style в master

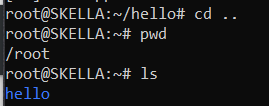
Поскольку последний коммит ветки master прямо предшествует последнему коммиту ветки style, git может выполнить ускоренное слияние-перемотку. При быстрой перемотке вперед git просто передвигает указатель вперед, таким образом указывая на тот же коммит, что и ветка style. При быстрой перемотке конфликтов быть не может.

**Просмотрите логи**

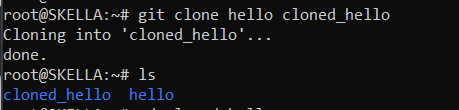


## Клонирование репозиториев

### Перейдите в рабочий каталог



**Создайте клон репозитория hello**

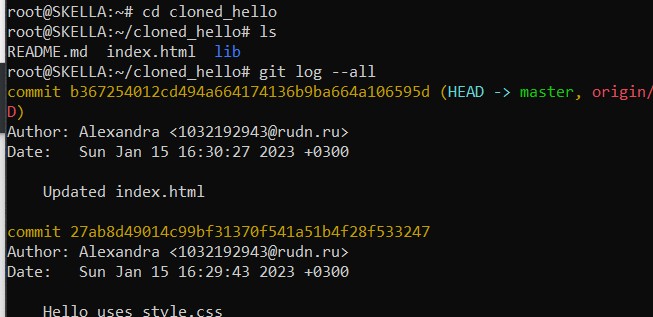


## Просмотр клонированного репозитория

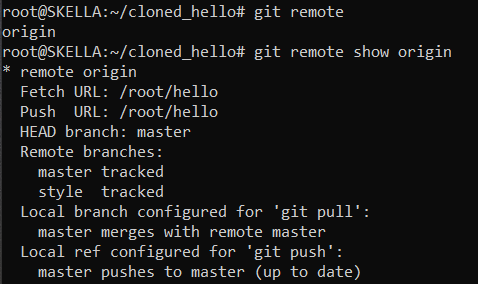
### Давайте взглянем на клонированный репозиторий.

Cписок всех файлов на верхнем уровне оригинального репозитория README.md, index.html и lib.

**Просмотрите историю репозитория**



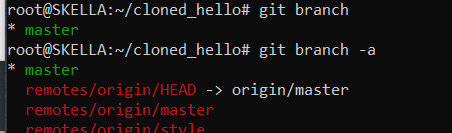
## Что такое origin?



**Удаленные ветки**

Как мы видим, в списке только ветка master.

### Список удаленных веток

Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные

## Изменение оригинального репозитория

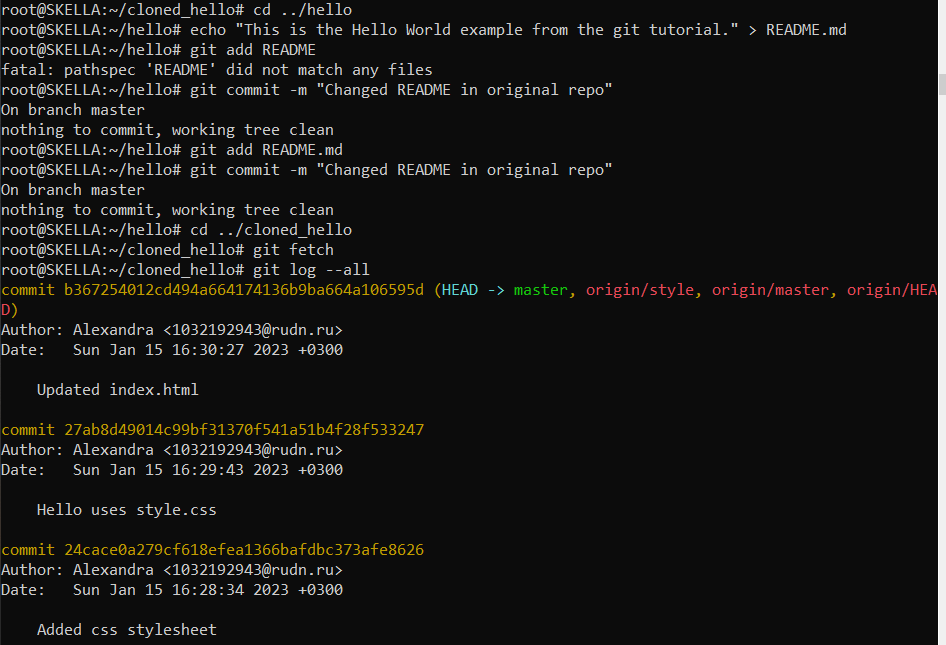
### Внесите изменения в оригинальный репозиторий hello

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

### Извлечение изменений

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. В истории выше найдите коммит «Changed README in original repo». Обратите внимание, что коммит включает в себя коммиты «origin/master» и

«origin/HEAD». Теперь давайте посмотрим на коммит «Updated index.html». Вы увидите, что локальная ветка master указывает на этот коммит, а не на новый коммит, который мы

только что извлекли. 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.

### Проверьте README.md

Мы можем продемонстрировать, что клонированный файл README.md не изменился.



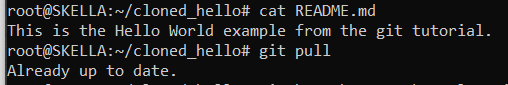
## Слияние извлеченных изменений

### Слейте извлеченные изменения в локальную ветку master

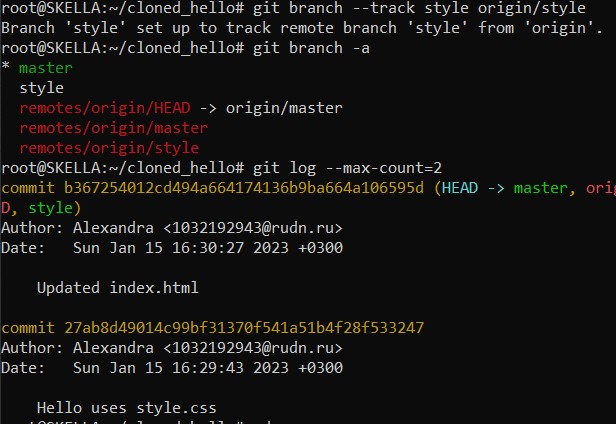
Выполните:

git merge origin/master

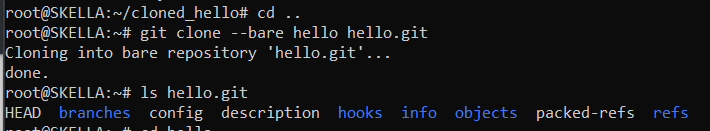
## Еще раз проверьте файл README.md



**Добавление ветки наблюдения**



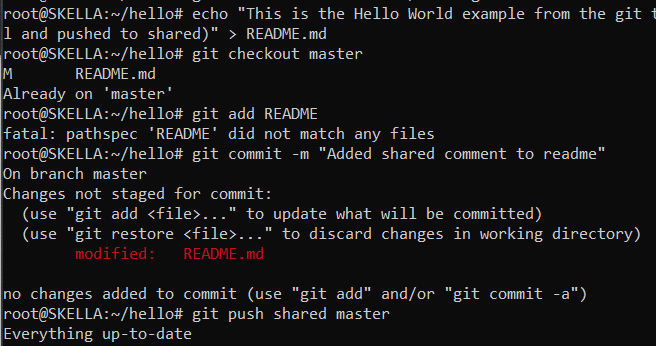
## Создайте чистый репозиторий

Как правило, репозитории, оканчивающиеся на .git являются чистыми репозиториями. Мы видим, что в репозитории hello.git нет рабочего каталога. По сути, это есть не что иное, как каталог .git нечистого репозитория.

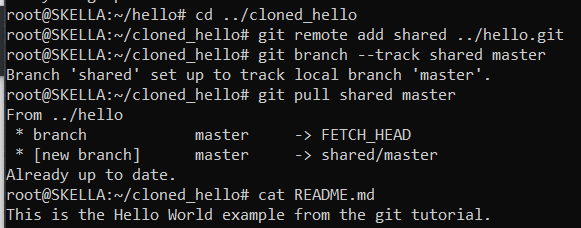
## Добавление удаленного репозитория



**Отправка изменений**



**Извлечение общих изменений**



# Выводы

Я научилась работать с git.

# Список литературы{.unnumbered}

::: {#refs} :::