#### МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра информационных технологий

**ОТЧЕТ по лабораторной работе 10**

ТЕМА **«Работа с git »**

**по дисциплине «Математическое моделирование»**

**Выполнил:**  
Студент группы НПИбд-02-21  
Студенческий билет № 1032205641  
Сатлихана Петрити

## Цель работы

Изучение работы с Git и знакомство с его утилитами.

## Последовательность выполнения работы

1. **Работа с git**

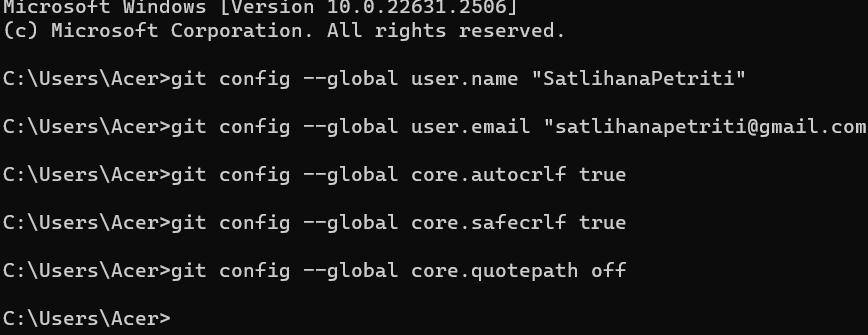
1.1 Подготовка

1.1.1 Установка имени и электронной почты

Если вы никогда ранее не использовали git, для начала вам необходимо осуществить установку. Выполните следующие команды, чтобы git узнал ваше имя и электронную почту. Если git уже установлен, можете переходить к разделу окончания строк.

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

git config --global user.email "your\_email@whatever.com"

  
 pиc.1- имя и электронную почту gita

1.1.2 Параметры установки окончаний строк

core.autocrlf configures newline character handling in Git:

true: Converts CRLF to LF on commit and vice versa on checkout (Windows)

input: Converts CRLF to LF on commit only (Mac/Linux).

core.safecrlf:

* true: Rejects irreversible LF <-> CRLF conversion.
* warn: Warns but accepts irreversible conversion.

Для пользователей Windows:

git config --global core.autocrlf true

git config --global core.safecrlf true

*(посмотрите рис 1 )*

1.1.3 Установка отображения unicode

По умолчанию, git будет печатать не-ASCII символов в именах файлов в виде восьмеричных последовательностей \nnn. Что бы избежать нечитаемых строк, установите соответствующий флаг.

git config --global core.quotepath off

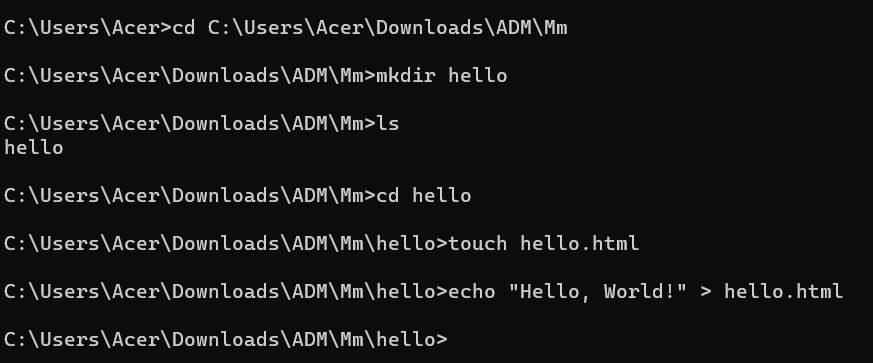
*(посмотрите рис 1 )*

1. **1.2 Создание проекта**

1.2.1 Создайте страницу «Hello, World»

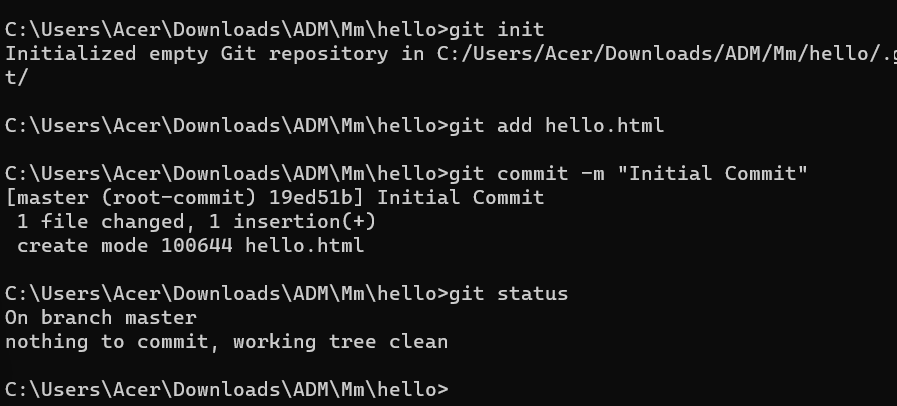
Начните работу в пустом рабочем каталоге с создания пустого каталога с именем hello, затем войдите в него и создайте там файл с именем hello.html

mkdir hello   
cd hello  
touch   
hello.html  
echo "Hello, World!" > hello.html

  
 pиc.2- создание каталог, html файл  
 1.2.2 Создание репозитория

Чтобы создать git репозиторий из этого каталога, выполните команду git init.

git init

  
 pиc.3- создать git репозиторий,добавление файла,проверка

1.2.3 Добавление файла в репозиторий  
Добавим файл в репозиторий.

git add hello.html  
git commit -m "Initial Commit"

*(посмотрите рис 3 )*

1.2.4 Проверка состояние репозитория. Используйте команду git status чтобы проверить текущее состояние репозитория.

git status

*(посмотрите рис 3 )*

1. **1.3 Внесение изменений**

* 1.3.1 Измените страницу «Hello, World»  
  Добавим кое-какие HTML-теги к нашему приветствию. Измените содержимое файла hello.html на:

Hello, World!

Проверьте состояние рабочего каталога.

git status

1. **1.4 Индексация изменений**

Теперь выполните команду git, чтобы проиндексировать изменения. Проверьте состояние.

git add hello.html  
git status

1.4.1 Коммит изменений

Сделайте коммит и проверьте состояние.

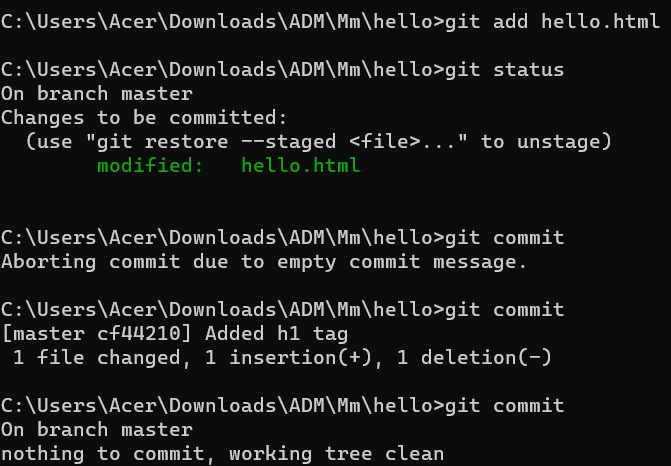
git commit

Откроется редактор. В первой строке введите комментарий: «Added h1 tag». Сохраните файл и выйдите из редактора (для этого в редакторе по-умолчанию (Vim) вам нужно нажать клавишу ESC, ввести :wq и нажать Enter).

Теперь еще раз проверим состояние.

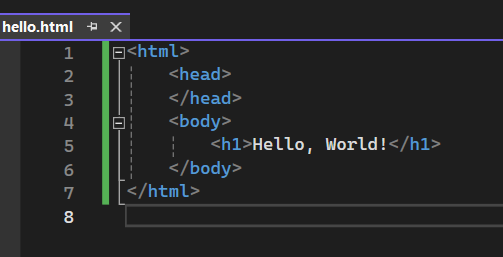
git status

Рабочий каталог чистый, можно продолжить работу.

  
 pиc.4- Индексация изменений

1.4.2 Добавьте стандартные теги страницы  
Измените страницу «Hello, World», чтобы она содержала стандартные теги и .

Hello, World!

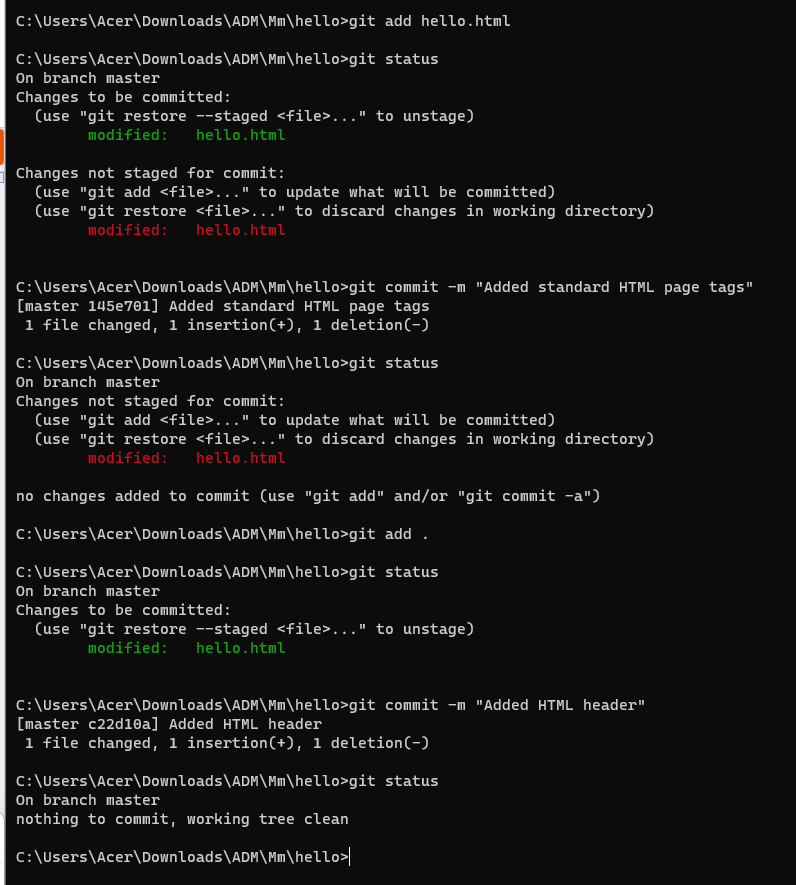
  
 pиc.5- добавим скрипт

Теперь добавьте это изменение в индекс git.

git add hello.html  
Теперь добавьте заголовки HTML (секцию ) к странице «Hello, World».

Hello, World!   
  
*(посмотрите рис 5 )*  
Проверьте текущий статус:

git status

  
pиc.6- добавьте изменение в git.Проверьте статус.

Произведите коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверьте состояние.

git commit -m "Added standard HTML page tags"

git status

*(посмотрите рис 6 )*

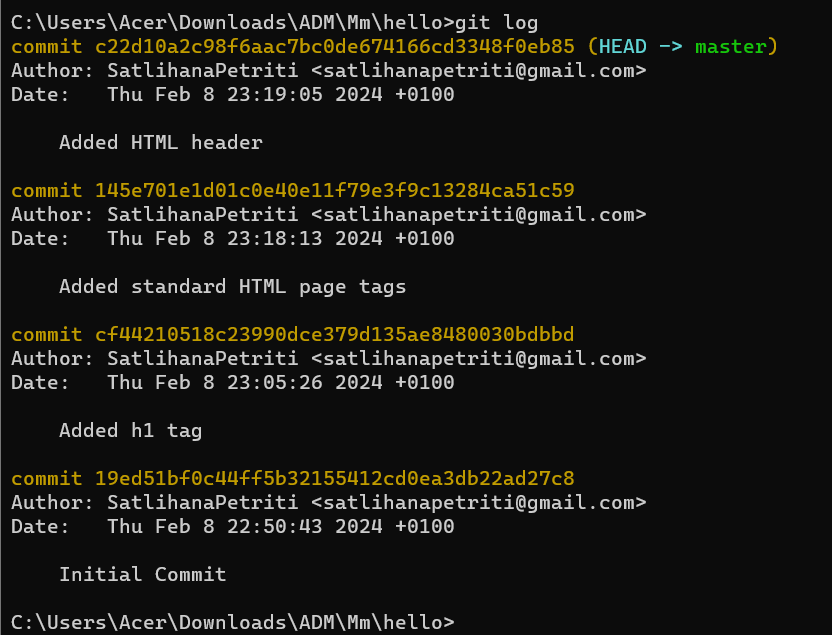
Теперь добавьте второе изменение в индекс, а затем проверьте состояние с помощью команды git status

git add .  
git status

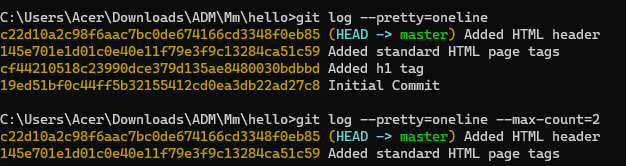
*(посмотрите рис 6 )*

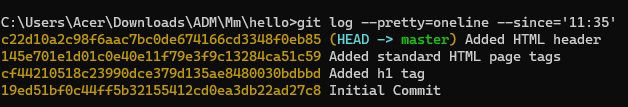
Второе изменение было проиндексировано и готово к коммиту. Сделайте коммит второго изменения  
git commit -m "Added HTML header"

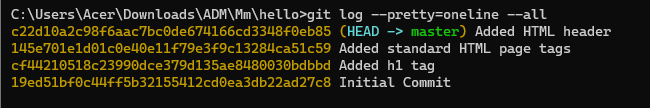
*(посмотрите рис 6 )*

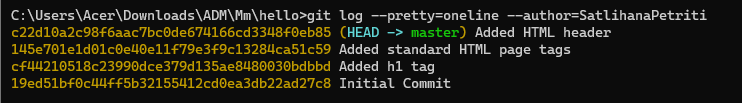
1.4.3 История  
Получим список произведенных изменений:  
git log  
  
 pиc.7- список произведенных изменений

Однострочный формат истории:  
git log --pretty=oneline  
Есть много вариантов отображения лога.  
git log --pretty=oneline --max-count=2  
git log --pretty=oneline --since='5 minutes ago'  
git log --pretty=oneline --until='5 minutes ago'  
git log --pretty=oneline --author=<your name>  
git log --pretty=oneline --all

  
 pиc.8- вариантов отображения лога(1)

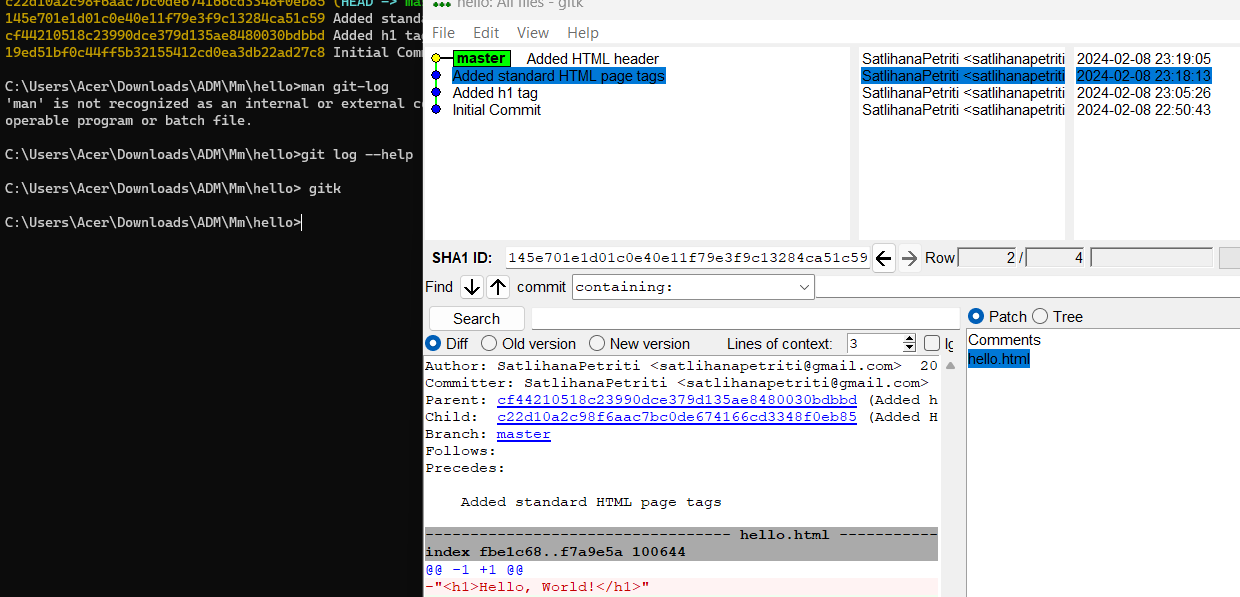
  
 pиc.9- вариантов отображения лога(2)

  
 pиc.10- вариантов отображения лога(3)

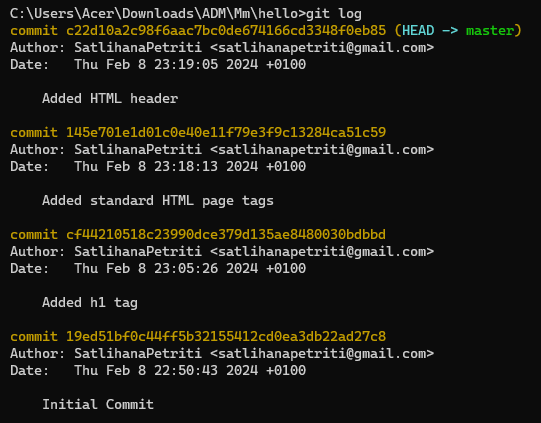
  
 pиc.11- вариантов отображения лога(4)

Справочная страница:  
man git-log

Инструмент gitk полезен в изучении истории изменений.

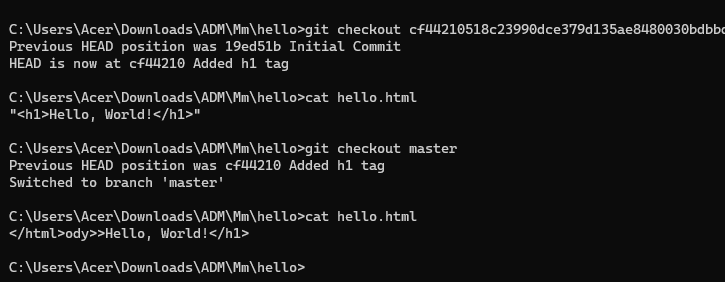
 pиc.12- Инструмент gitk

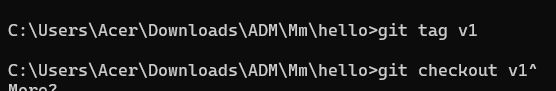
1.4.4 Получение старых версий  
Возвращаться назад в историю очень просто. Команда checkout скопирует любойснимок из репозитория в рабочий каталог.  
Получите хэши предыдущих версий  
git log

  
 pиc.13- хэши предыдущих версий

Затем проверьте содержимое файла hello.html.  
git checkout <hash>  
cat hello.html

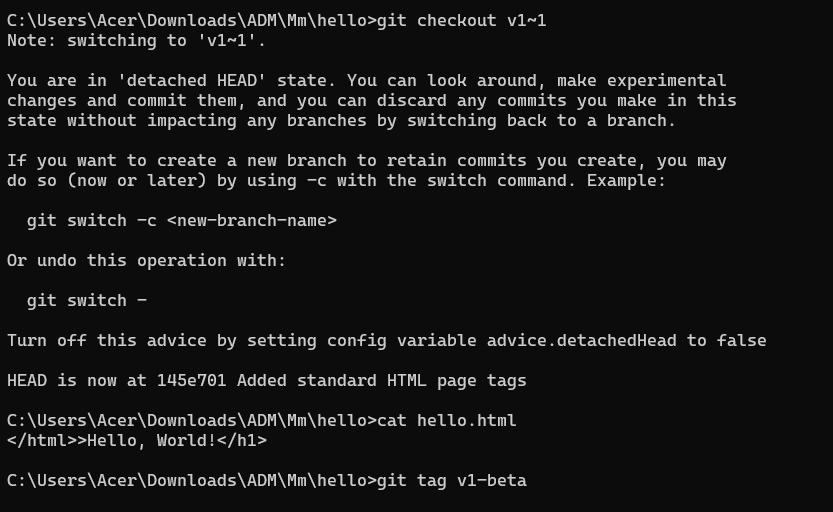
Вернитесь к последней версии в ветке master

git checkout master  
cat hello.html  
master — имя ветки по умолчанию. Переключая имена веток, вы попадаете напоследнюю версию выбранной ветки.  
  
 pиc.14- проверьте содержимое файла hello.html, вернитесь к ветке master

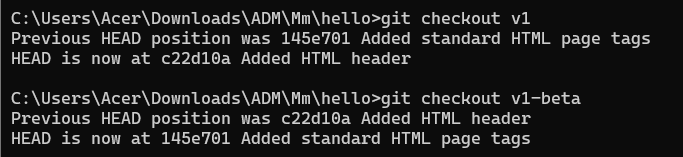
1.4.5 Создание тегов версий  
Давайте назовем текущую версию страницы hello первой (v1).Создайте тег первой версии  
git tag v1  
  
 pиc.15- Создание тегов версий

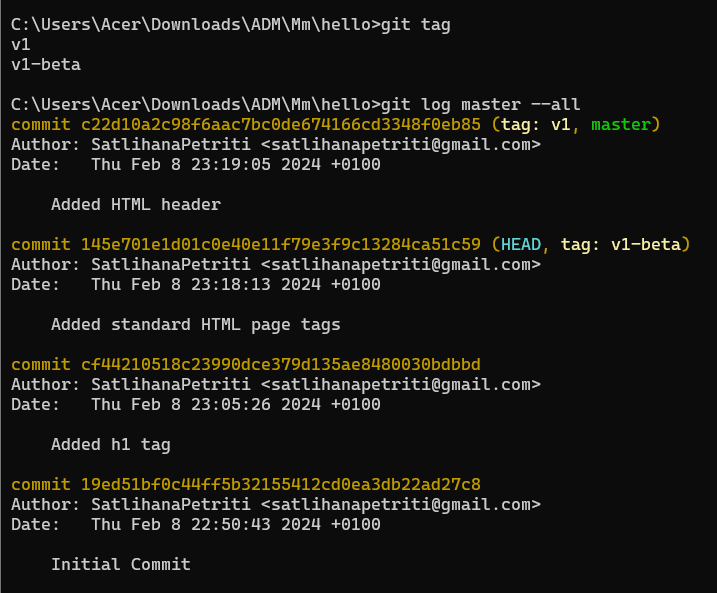
Теперь текущая версия страницы называется v1.  
Теги для предыдущих версий Давайте создадим тег для версии, которая идетперед текущей версией и назовем его v1-beta. В первую очередь нам надо переключиться на предыдущую версию. Вместо поиска до хэш, мы будем использовать^, обозначающее «родитель v1». Вместо обозначения v1^ можно использоватьv1~1. Это обозначение можно определить как «первую версию предшествующуюv1».  
git checkout v1^``  
``cat hello.html

Это версия c тегами и , но еще пока без . Давайте сделаемее версией v1-beta.  
git tag v1-beta

  
 pиc.16-Создание тегов версий, beta, v1

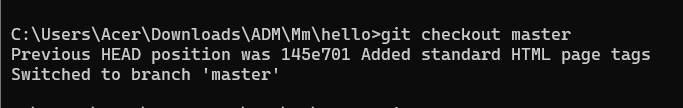
1.4.6 Переключение по имени тега  
Теперь попробуйте попереключаться между двумя отмеченными версиями.  
git checkout v1  
git checkout v1-beta

  
 pиc.17-Переключение по имени тега

1.4.7 Просмотр тегов с помощью команды tag  
Вы можете увидеть, какие теги доступны, используя команду git tag.  
git tag  
Вы также можете посмотреть теги в логе.  
git log master --all  
  
 pиc.18-Просмотр тегов с помощью команды tag

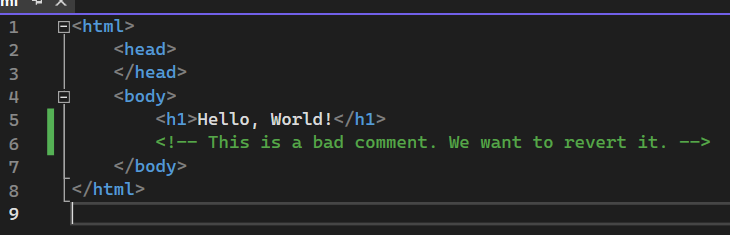
1. **1.5 Отмена локальных изменений (до индексации)**

1.5.1 Переключитесь на ветку master  
Убедитесь, что вы находитесь на последнем коммите ветки master, прежде чем продолжить работу.  
git checkout master

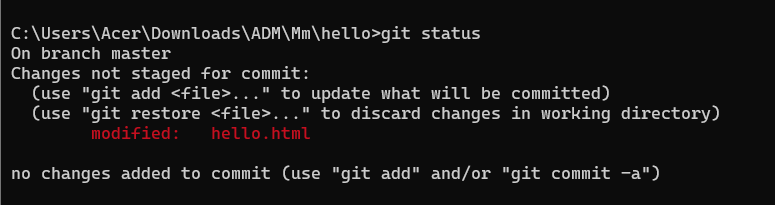
  
 pиc.19-Переключитесь на ветку master

1.5.2 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.  
<html>``  
``<body><h1>Hello, World!</h1>

<!-- This is a bad comment. We want to revert it. -->  
</body>  
</html>

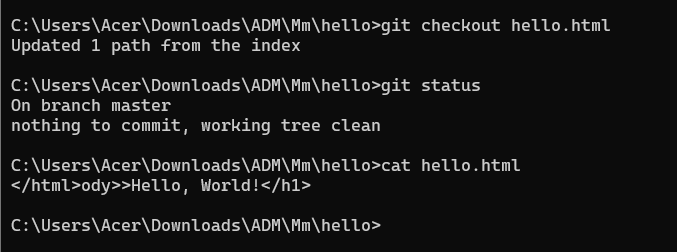
  
 pиc.20-Измените hello.html

1.5.3 Проверьте состояние  
Сначала проверьте состояние рабочего каталога.  
git status

  
 pиc.21-Проверьте состояние рабочего каталога

Мы видим, что файл hello.html был изменен, но еще не проиндексирован.

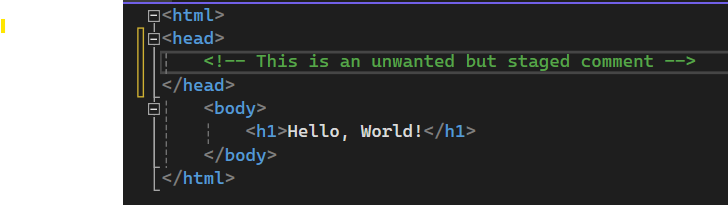
1.5.4 Отмена изменений в рабочем каталоге  
Используйте команду git checkout для переключения версии файла hello.html в репозитории.  
git checkout hello.html  
git status  
cat hello.html

  
 pиc.22-Отмена изменений в рабочем каталоге

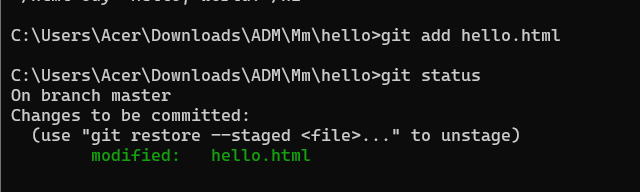
Команда git status показывает нам, что не было произведено никаких изменений, не зафиксированных в рабочем каталоге.

1. **1.6 Отмена проиндексированных изменений (перед коммитом)**

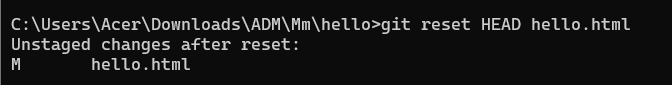
* 1.6.1 Измените файл и проиндексируйте изменения  
  Внесите изменение в файл hello.html в виде нежелательного комментария

  
 pиc.23 изменение в файл hello.html

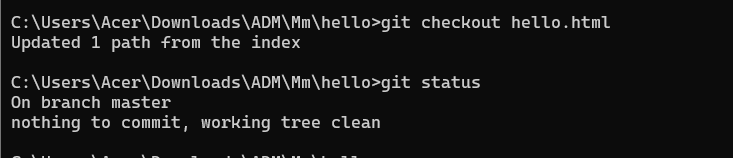
Проиндексируйте это изменение.  
git add hello.html  
 1.6.2 Проверьте состояние  
Проверьте состояние нежелательного изменения.  
git status

  
 pиc.24 индексируем это изменение и проверь состояние

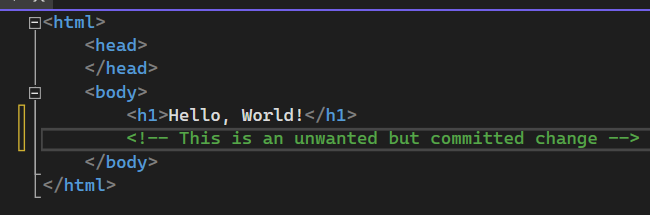
Состояние показывает, что изменение было проиндексировано и готово к коммиту.  
 1.6.3 Выполните сброс буферной зоны  
К счастью, вывод состояния показывает нам именно то, что мы должны сделатьдля отмены индексации изменения.  
git reset HEAD hello.html

  
 pиc.25 сброс буферной зоны

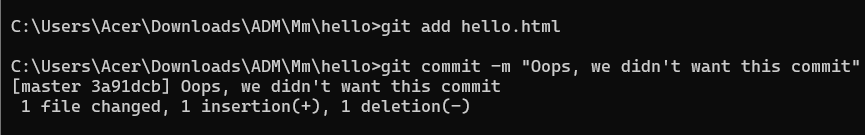
Команда git reset сбрасывает буферную зону к HEAD.  
Команда git reset (по умолчанию) не изменяет рабочий каталог. Поэтому рабочий каталог все еще содержит нежелательный комментарий.   
Мы можем использовать команду git checkout, чтобы удалить нежелательные изменения врабочем каталоге.  
 1.6.4 Переключитесь на версию коммита  
git checkout hello.html  
git status

  
 pиc.26 Переключат на версию коммита

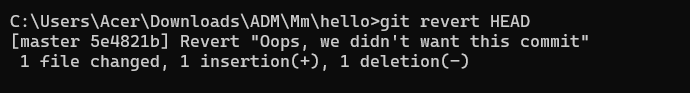
Наш рабочий каталог опять чист.

1. **1.7 Отмена коммитов**  
   1.7.2 Измените файл и сделайте коммит  
   Измените файл hello.html на следующий.  
     
    pиc.27 Измените файл hello.html

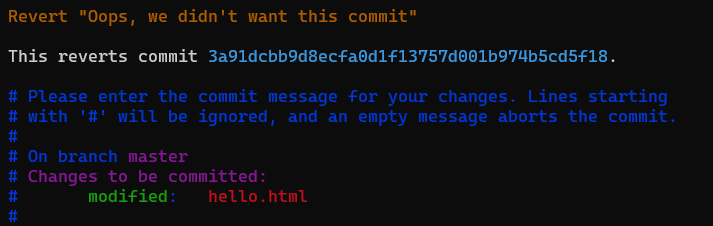
Выполните:  
git add hello.html  
git commit -m "Oops, we didn't want this commit"

  
 pиc.28 Выполним команды

1.7.3 Сделайте коммит с новыми изменениями, отменяющими предыдущие  
Чтобы отменить коммит, нам необходимо сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом.  
git revert HEAD

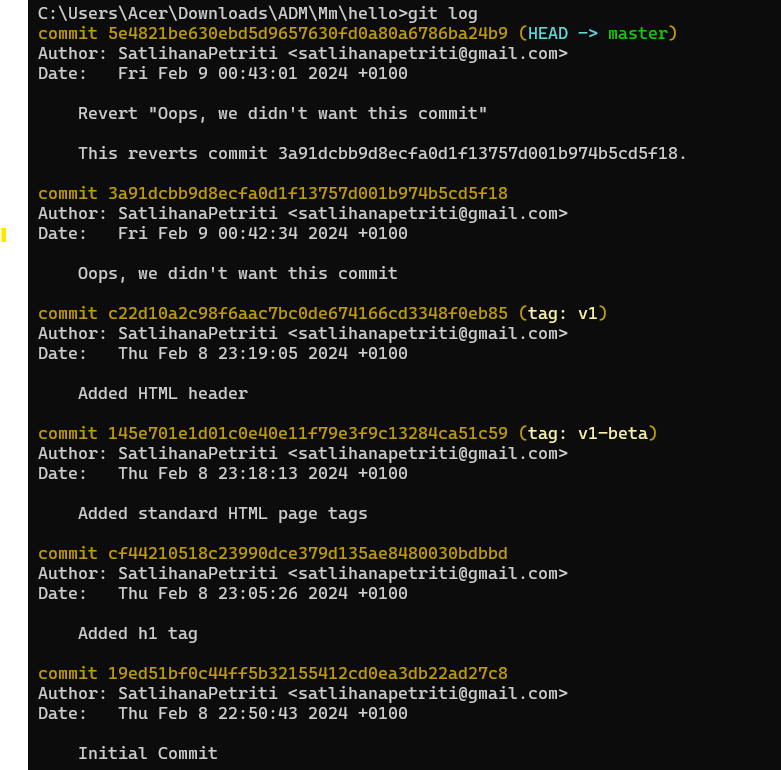
  
 pиc.29 git revert HEAD

Перейдите в редактор, где вы можете отредактировать коммит-сообщение поумолчанию или оставить все как есть. Сохраните и закройте файл.

  
 pиc.30 редактор

Так как мы отменили самый последний произведенный коммит, мы смоглииспользовать HEAD в качестве аргумента для отмены. Мы можем отменить любойпроизвольной коммит в истории, указав его хэш-значение.

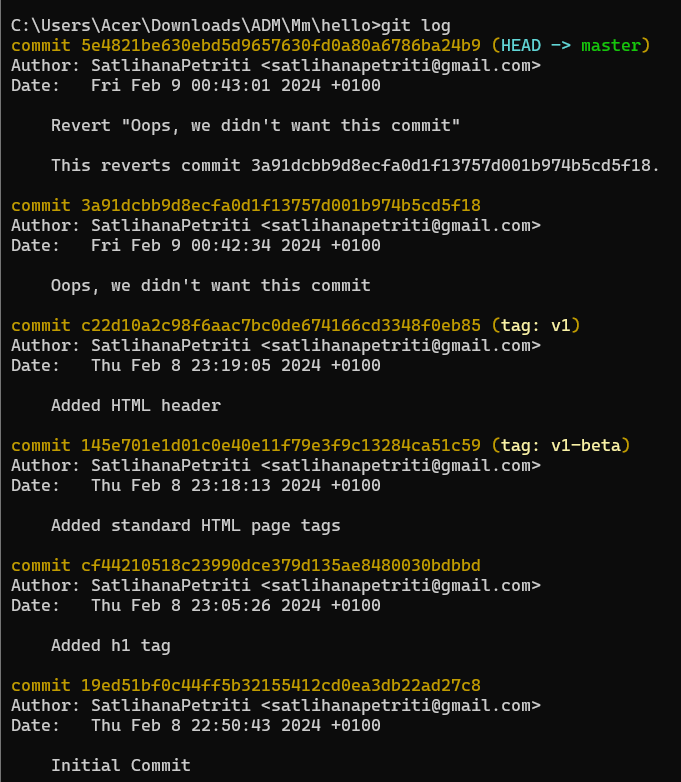
1.7.4 Проверьте лог  
Проверка лога показывает нежелательные и отмененные коммиты в наш репозиторий.  
git log

  
 pиc.31 нежелательные и отмененные коммиты

Эта техника будет работать с любым коммитом

1. **1.8 Удаление коммиттов из ветки**

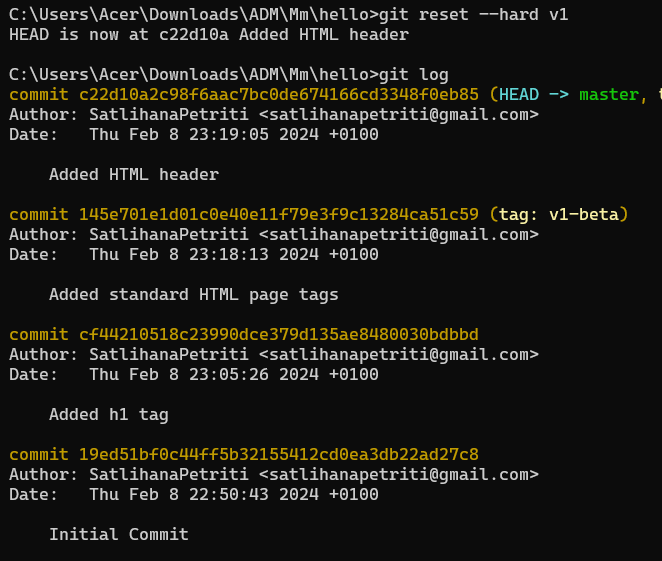
git revert является мощной командой, которая позволяет отменить любые коммиты в репозиторий. Однако, и оригинальный и «отмененный» коммиты видны вистории ветки (при использовании команды git log).  
 1.8.1 Команда git reset  
При получении ссылки на коммит (т.е. хэш, ветка или имя тега), команда gitreset:  
• перепишет текущую ветку, чтобы она указывала на нужный коммит;  
• опционально сбросит буферную зону для соответствия с указанным коммитом;  
• опционально сбросит рабочий каталог для соответствия с указанным коммитом.  
 1.8.2 Проверьте нашу историю  
Давайте сделаем быструю проверку нашей истории коммитов. Выполните:  
git log

  
 pиc.32 проверку нашей истории коммитов

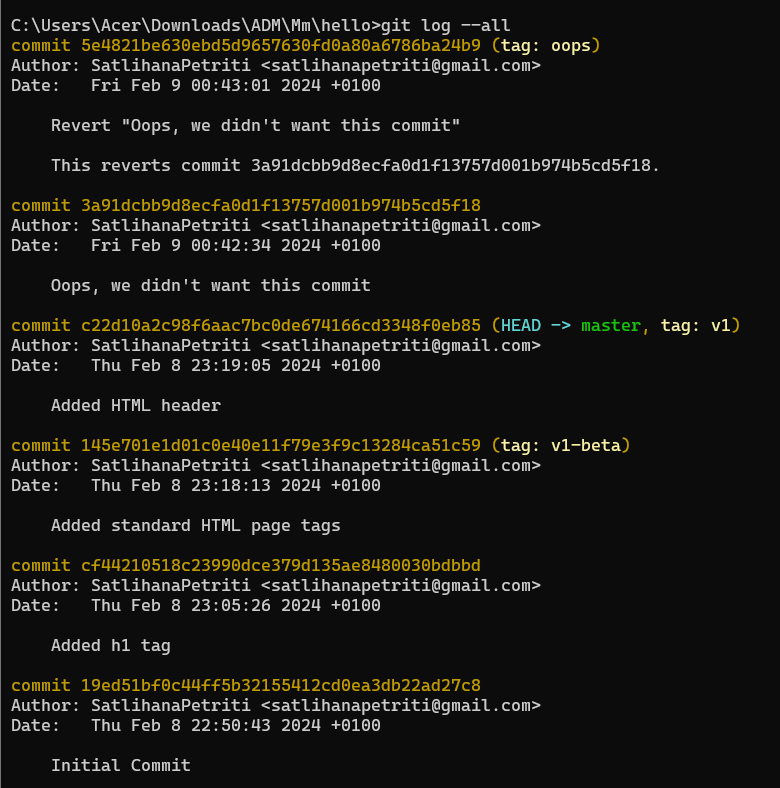
Мы видим, что два последних коммита в этой ветке — «Oops» и «Revert Oops».Давайте удалим их с помощью сброса.  
 1.8.3 Для начала отметьте эту ветку  
Но прежде чем удалить коммиты,давайте отметим последний коммит тегом,чтобы потом можно было его найти.  
git tag oops

fig:  
 pиc.33 отметим последний коммит тегом

1.8.4 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).  
git reset --hard v1  
git log

  
 pиc.34 git reset --hard v1, git log

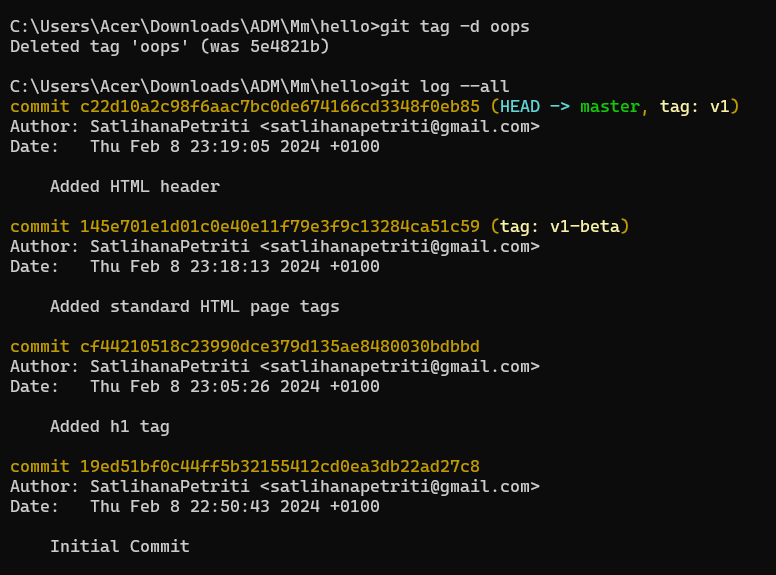
Наша ветка master теперь указывает на коммит v1, а коммитов Oops и RevertOops в ветке уже нет. Параметр --hard указывает, что рабочий каталог долженбыть обновлен в соответствии с новым head ветки.  
 1.8.5 Ничего никогда не теряется  
Что же случается с ошибочными коммитами? Оказывается, что коммиты все ещенаходятся в репозитории. На самом деле, мы все еще можем на них ссылаться.Помните, в начале этого урока мы создали для отмененного коммита тег «oops».Давайте посмотрим на все коммиты.  
git log --all

  
 pиc.35 все коммиты

Мы видим, что ошибочные коммиты не исчезли. Они все еще находятся в репозитории. Просто они отсутствуют в ветке master.

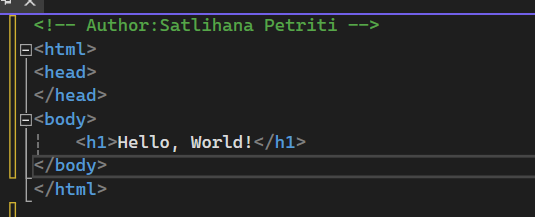
1. **1.9 Удаление тега oops**

1.9.1 Удаление тега oops  
Тег oops свою функцию выполнил. Давайте удалим его и коммиты, на которые онссылался, сборщиком мусора.  
git tag -d oops  
git log --all

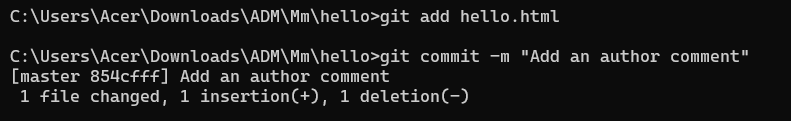
  
 pиc.36 Удаление тега oops

Тег «oops» больше не будет отображаться в репозитории.

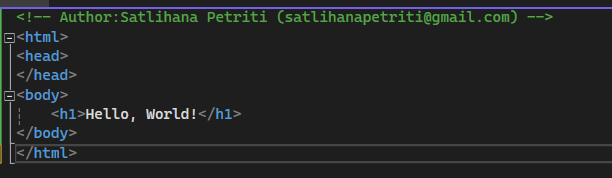
1. **1.10 Внесение изменений в коммиты**

1.10.1 Измените страницу, а затем сделайте коммит  
Добавьте в страницу комментарий автора (вставьте свою фамилию).  
  
 pиc.37 Изменим страницу

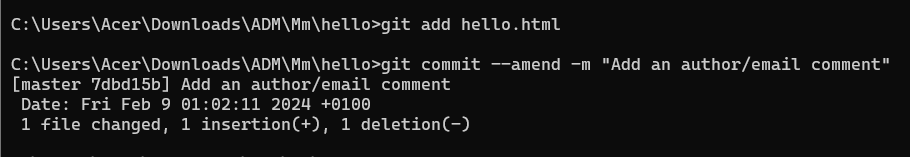
Выполните:  
git add hello.html  
git commit -m "Add an author comment"

  
 pиc.38 выполнения команды

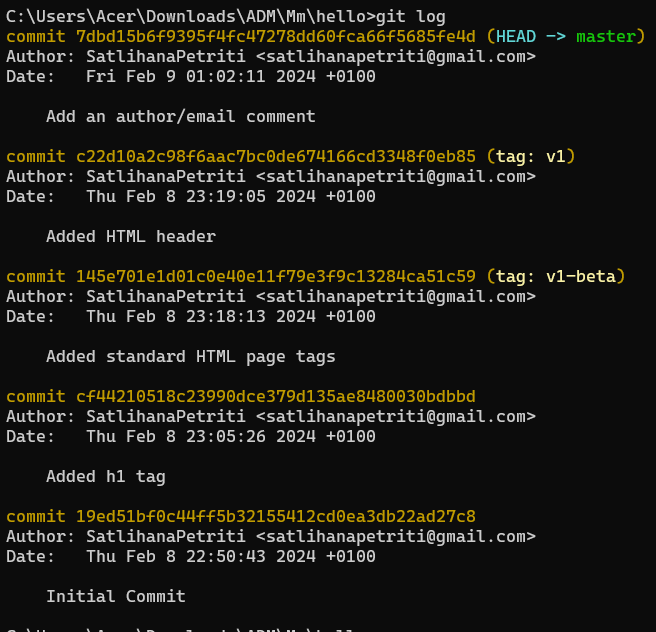
1.10.2 Необходим email

  
 pиc.39 Изменим страницу

После совершения коммита вы понимаете, что любой хороший комментарий должен включать электронную почту автора. Обновите страницу hello, включив внее email.  
 1.10.3 Измените предыдущий коммит  
Мы действительно не хотим создавать отдельный коммит только ради электроннойпочты. Давайте изменим предыдущий коммит, включив в него адрес электронной  
почты.Выполните:  
git add hello.html  
git commit --amend -m "Add an author/email comment"

  
 pиc.40 выполнения команды

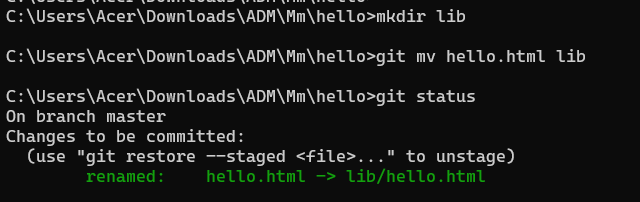
1.10.4 Просмотр истории  
Выполните:  
git log

  
 pиc.41 Просмотр истории

Мы можем увидеть, что оригинальный коммит «автор» заменен коммитом «автор/email». Этого же эффекта можно достичь путем сброса последнего коммита вветке, и повторного коммита новых изменений.

1. **1.11 Перемещение файлов**

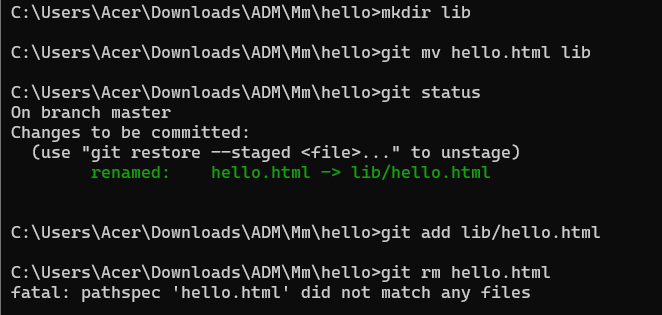
1.11.1 Переместите файл hello.html в каталог lib  
Сейчас мы собираемся создать структуру нашего репозитория. Давайте перенесемстраницу в каталог lib.  
mkdir lib  
git mv hello.html lib  
git status

  
 pиc.42 Переместите файл hello.html в каталог lib

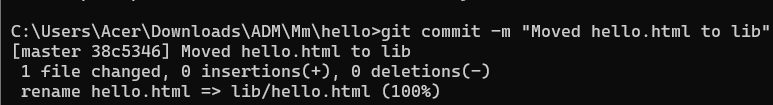
Перемещая файлы с помощью git mv, мы информируем git о 2 вещах:  
• Что файл hello.html был удален.  
• Что файл lib/hello.html был создан.  
• Оба эти факта сразу же проиндексированы и готовы к коммиту. Команда git  
status сообщает, что файл был перемещен

1. **1.12 Второй способ перемещения файлов**

Положительной чертой git является то, что вы можете забыть о версионном контроле до того момента, когда вы готовы приступить к коммиту кода.  
Следующий набор команд идентичен нашим последним действиям. Работыздесь побольше, но результат тот же.  
Мы могли бы выполнить:  
mkdir lib  
mv hello.html lib  
git add lib/hello.html  
git rm hello.html

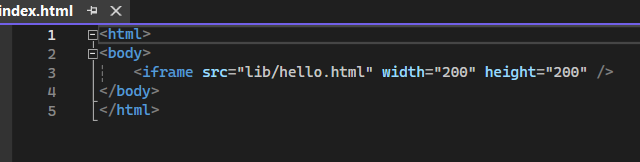
  
 pиc.43 Второй способ перемещения файлов

1.12.1 Коммит в новый каталог  
Давайте сделаем коммит этого перемещения:  
git commit -m "Moved hello.html to lib"

  
 pиc.44 Коммит в новый каталог

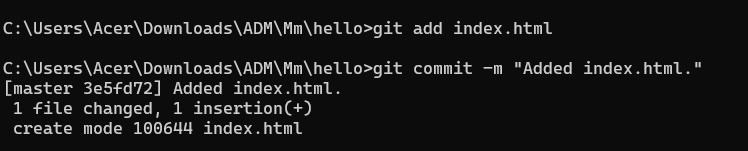
1. **1.13 Подробнее о структуре**

1.13.1 Добавление index.html  
Добавим файл index.html в наш репозиторий

  
 pиc.45 Добавление index.html

  
pиc.46 результат

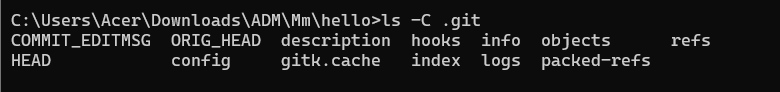
Добавьте файл и сделайте коммит.  
git add index.html  
git commit -m "Added index.html."

  
 pиc.46 Добавьте файл и сделайте коммит

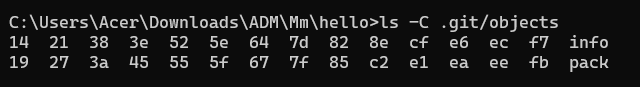
Теперь при открытии index.html, вы должны увидеть кусок страницы hello в маленьком окошке.

1. 1.14 Git внутри: Каталог .git

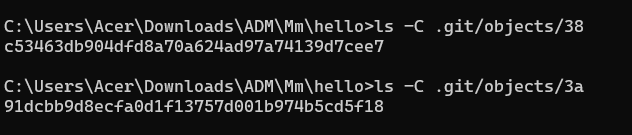
1.14.1 Каталог .git  
Выполните:  
ls -C .git

  
 pиc.47 Каталог .git

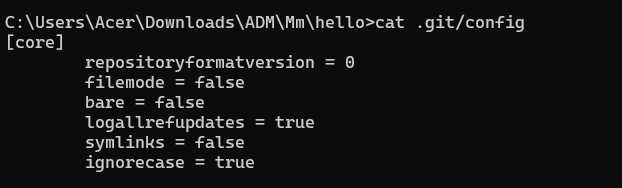
Это каталог, в котором хранится вся информация git.  
 1.14.2 База данных объектов  
Выполните:  
ls -C .git/objects

  
 pиc.48 База данных объектов

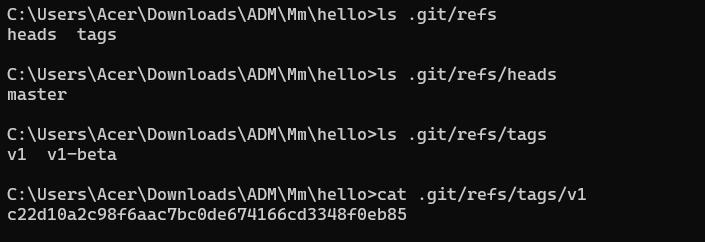
Вы должны увидеть набор каталогов, имена которых состоят из 2 символов. Имена каталогов являются первыми двумя буквами хэша sha1 объекта, хранящегося вgit.  
 1.14.3 Углубляемся в базу данных объектов  
Выполните:  
ls -C .git/objects/<dir>

  
 pиc.49 Углубляемся в базу данных объектов

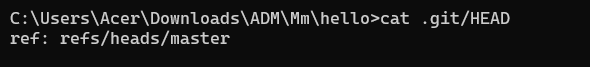
Смотрим в один из каталогов с именем из 2 букв. Вы увидите файлы с именамииз 38 символов. Это файлы, содержащие объекты, хранящиеся в git. Они сжаты изакодированы, поэтому просмотр их содержимого нам мало чем поможет.  
 1.14.4 Config File  
Выполните:  
cat .git/config

  
 pиc.50 Config File

Это файл конфигурации, создающийся для каждого конкретного проекта. Записив этом файле будут перезаписывать записи в файле .gitconfig вашего главногокаталога, по крайней мере в рамках этого проекта.  
 1.14.5 Ветки и теги  
Выполните:  
ls .git/refs  
ls .git/refs/heads  
ls .git/refs/tags  
cat .git/refs/tags/v1

  
 pиc.51 Ветки и теги

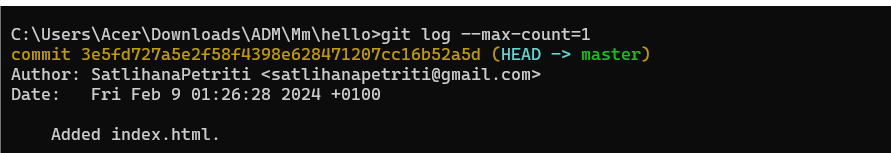
Вы должны узнавать файлы в подкаталоге тегов. Каждый файл соответствует тегу, ранее созданному с помощью команды git tag. Его содержание — это всего лишь хэш коммита, привязанный к тегу.  
 1.14.6 Файл HEAD  
Выполните:  
cat .git/HEAD

  
 pиc.52 Файл HEAD

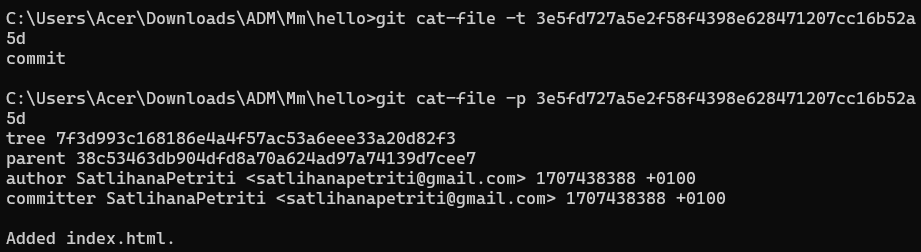
Файл HEAD содержит ссылку на текущую ветку, в данный момент это должна быть ветка master.

1. **1.15 Работа непосредственно с объектами git**

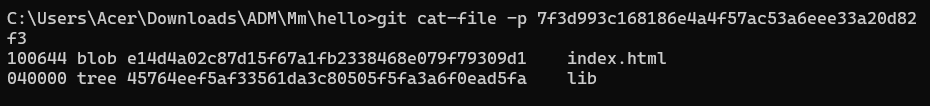
1.15.1 Поиск последнего коммита  
Выполните:  
git log --max-count=1

  
 pиc.53 Поиск последнего коммита

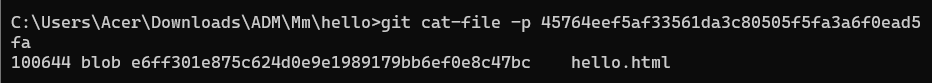
Эта команда должна показать последний коммит в репозиторий. SHA1 хэш ввашей системе, вероятно, отличается от моего, но вы увидите что-то наподобиеэтого.  
 1.15.2 Вывод последнего коммита с помощью SHA1 хэша  
Выполните:  
git cat-file -t <hash>  
git cat-file -p <hash>

  
 pиc.54 Вывод последнего коммита с помощью SHA1 хэша

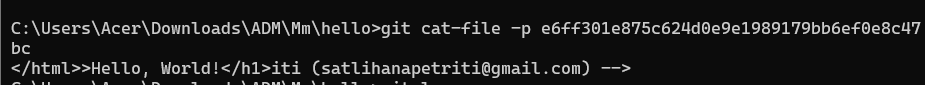
1.15.3 Поиск дерева  
Мы можем вывести дерево каталогов, ссылка на который идет в коммите. Это должно быть описание файлов (верхнего уровня) в нашем проекте (для конкретногокоммита). Используйте SHA1 хэш из строки «дерева», из списка выше.  
Выполните:  
git cat-file -p <treehash>

  
 pиc.55 Поиск дерева

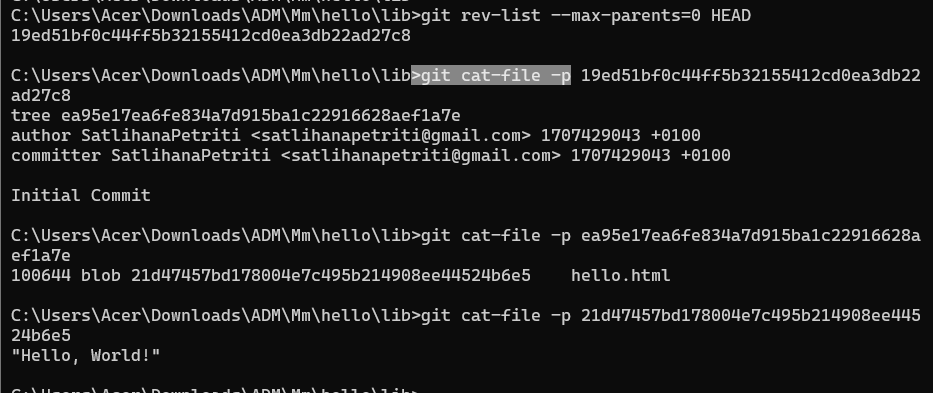
1.15.4 Вывод каталога lib  
Выполните:  
git cat-file -p <libhash>

  
 pиc.56 Вывод каталога lib

1.15.5 Вывод файла hello.html  
Выполните:  
git cat-file -p <hellohash>

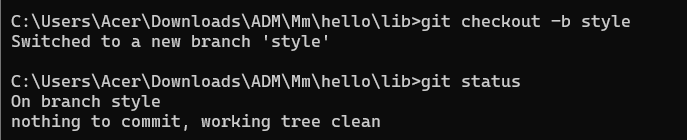
  
 pиc.57 Вывод файла hello.html

1.15.6 Исследуйте самостоятельно  
Исследуйте git репозиторий вручную самостоятельно. Смотрите, удастся ли вам найти оригинальный файл hello.html с самого первого коммита вручную по ссылкам SHA1 хэша в последнем коммите.

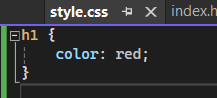
  
 pиc.58 оригинальный файл hello.html с самого первого коммита

1. **1.16 Создание ветки**

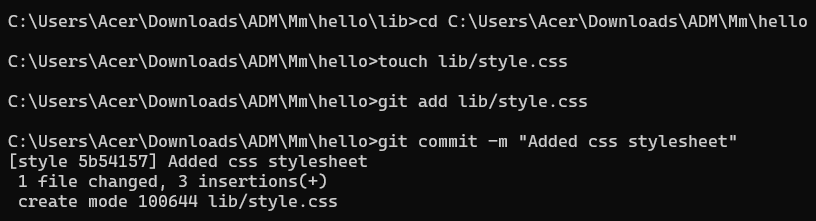
Пора сделать наш hello world более выразительным. Так как это может занятьнекоторое время, лучше переместить эти изменения в отдельную ветку, чтобыизолировать их от изменений в ветке master.  
 1.16.1 Создайте ветку  
Давайте назовем нашу новую ветку «style».Выполните:  
git checkout -b style  
git status

  
 pиc.59 Создайте ветку

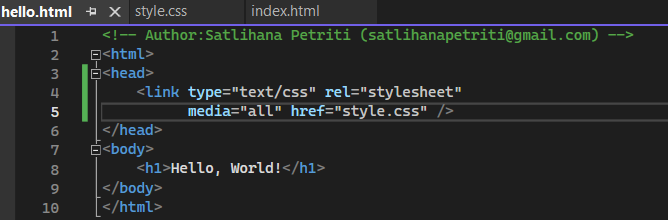
git checkout -b <имя*ветки> является шорткатом для git branch<имя*ветки> за которым идет git checkout <имя\_ветки>.  
Обратите внимание, что команда git status сообщает о том, что вы находитесь в ветке «style».  
 1.16.2 Добавьте файл стилей style.css

  
pиc.60 Добавьте файл стилей style.css

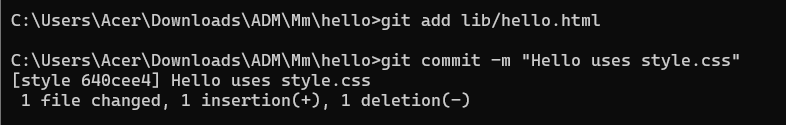
Выполните:  
touch lib/style.css  
Файл lib/style.css:  
h1 {  
color: red;  
}  
Выполните:  
git add lib/style.css  
git commit -m "Added css stylesheet"

  
 pиc.61 выполнение команды

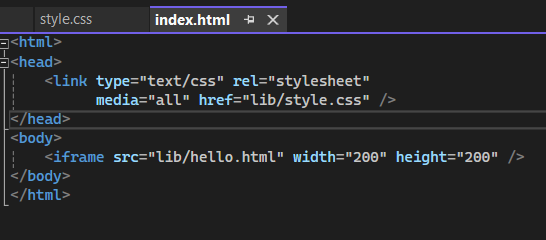
1.16.3 Измените основную страницу  
Обновите файл hello.html, чтобы использовать стили style.css

  
 pиc.62 Измените основную страницу

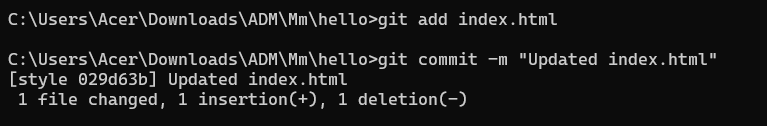
Выполните:  
git add lib/hello.html  
git commit -m "Hello uses style.css"

  
 pиc.63 выполнение команды

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

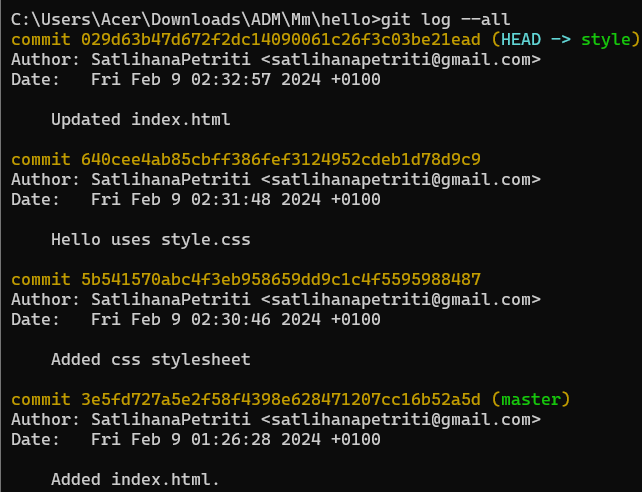
  
 pиc.64 Измените index.html

Выполните:  
git add index.html  
git commit -m "Updated index.html"

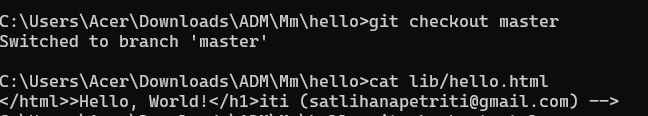
  
 pиc.65 выполнение команды

1. **1.17 Навигация по веткам**

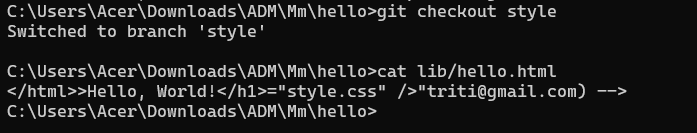
Теперь в вашем проекте есть две ветки: Выполните:  
git log --all

  
 pиc.66 Навигация по веткам- есть две ветки

1.17.1 Переключение на ветку master  
Используйте команду git checkout для переключения между ветками:  
git checkout master  
cat lib/hello.html

  
 pиc.67 переключения между ветками

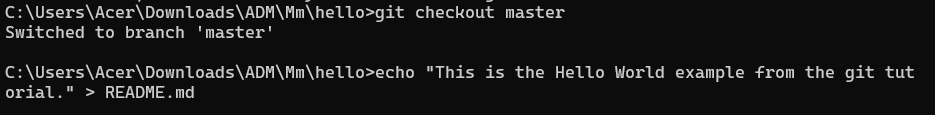
Сейчас мы находимся на ветке master. Это заметно по тому, что файл  
hello.html не использует стили style.css.  
 1.17.2 Вернемся к ветке style  
Выполните:  
git checkout style  
cat lib/hello.html

  
 pиc.68 Вернемся к ветке style

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

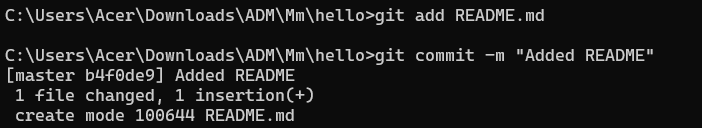
1. **1.18 Изменения в ветке master**

Пока вы меняли ветку style, кто-то решил обновить ветку master. Они добавили файл README.md.  
 1.18.1 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

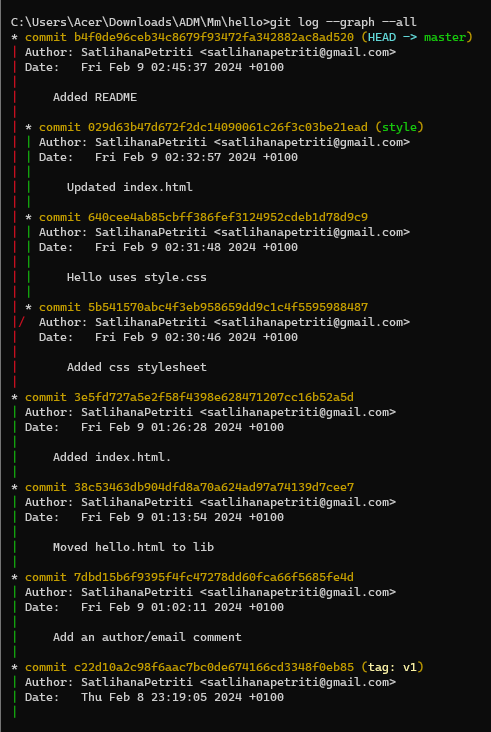
  
 pиc.69 Изменения в ветке master

1. **1.19 Сделайте коммит изменений README.md в ветку master**.

Выполните:  
git add README.mdgit commit -m "Added README"

  
 pиc.70 коммит изменений README.md в ветку master

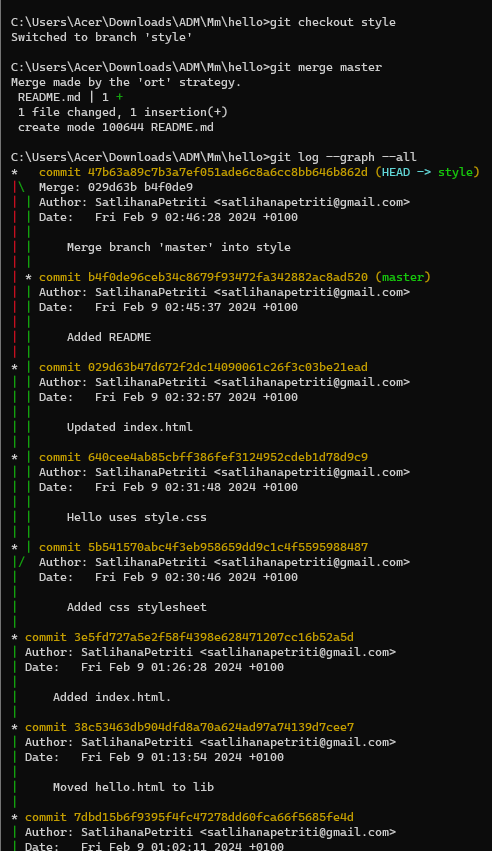
1.19.1 Просмотр отличающихся веток  
 1.19.2 Просмотрите текущие ветки  
Теперь у нас в репозитории есть две отличающиеся ветки. Используйте следующую  
лог-команду для просмотра веток и их отличий. Выполните:  
git log --graph --all

  
pиc.71 просмотра веток и их отличий

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.

1. **1.20 Слияние**

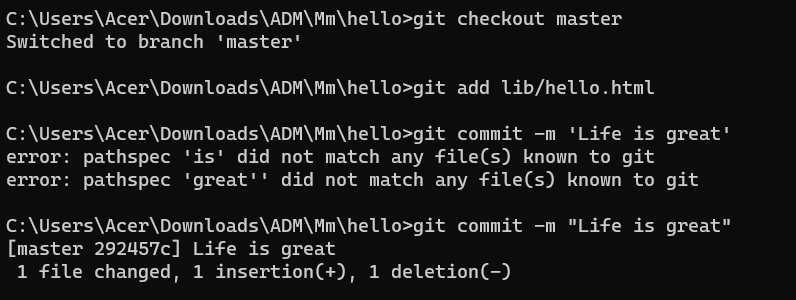
1.20.1 Слияние веток  
Слияние переносит изменения из двух веток в одну. Давайте вернемся к веткеstyle и сольем master с style.  
Выполните:  
git checkout style  
git merge master  
git log --graph --all

  
 pиc.72 Слияние веток

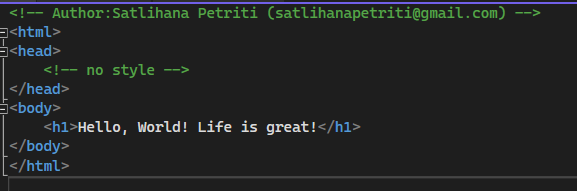
Путем периодического слияния ветки master с веткой style вы можете переносить из master любые изменения и поддерживать совместимость измененийstyle с изменениями в основной ветке.

1. **1.21 Создание конфликта**

1.21.1 Вернитесь в master и создайте конфликт  
Вернитесь в ветку master и внесите следующие изменения:  
git checkout master

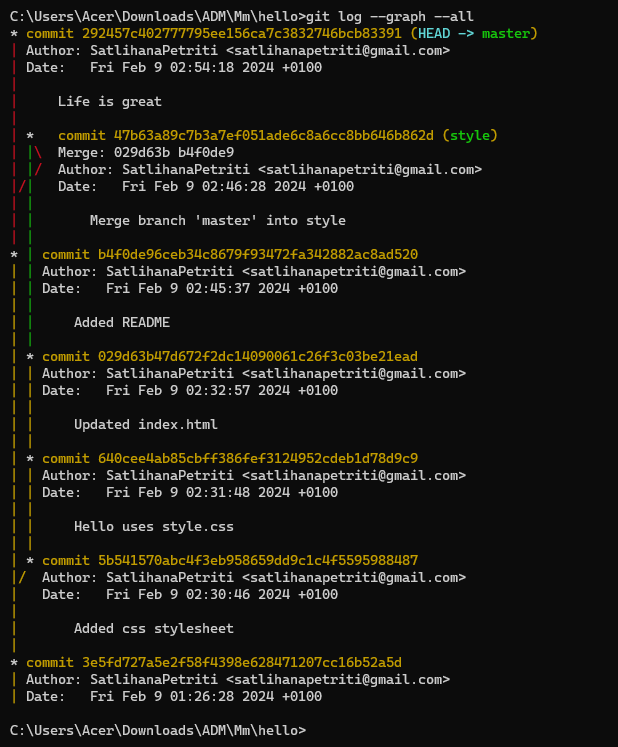
  
 pиc.73 Вернитесь в master+выполнение команды

Файл lib/hello.html

  
 pиc.74 Файл lib/hello.html

Выполните:  
git add lib/hello.html  
git commit -m 'Life is great'

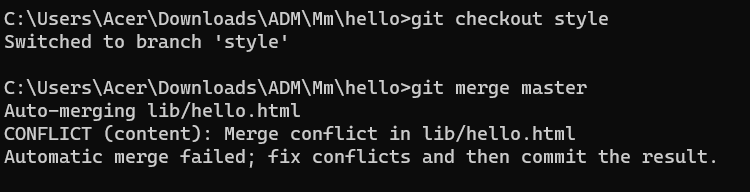
1.21.2 Просмотр веток  
Выполните:  
git log --graph --all

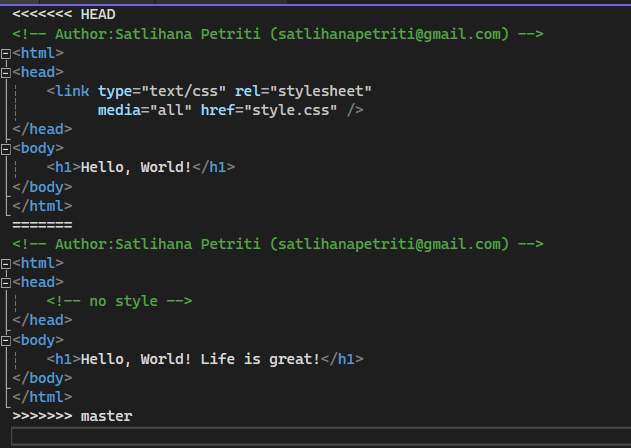
  
 pиc.75 Просмотры веток

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.

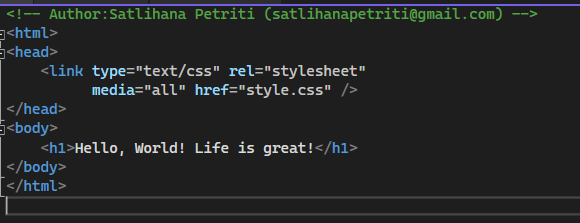
1. **1.22 Разрешение конфликтов**

1.22.1 Слияние master с веткой style  
Теперь вернемся к ветке style и попытаемся объединить ее с новой веткойmaster.  
Выполните:  
git checkout style  
git merge master

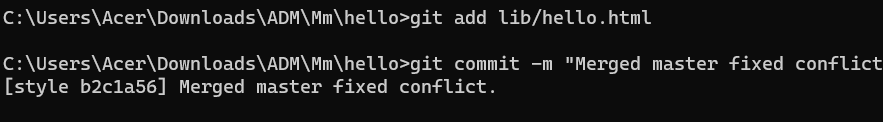
  
 pиc.76 Слияние master с веткой style

Если вы откроете lib/hello.html, вы увидите:  
  
 pиc.77 результаты выполнения команды

Первый раздел — версия текущей ветки (style). Второй раздел — версия веткиmaster.  
 1.22.2 Решение конфликта  
Вам необходимо вручную разрешить конфликт. Внесите изменения в lib/hello.htmlдля достижения следующего результата.

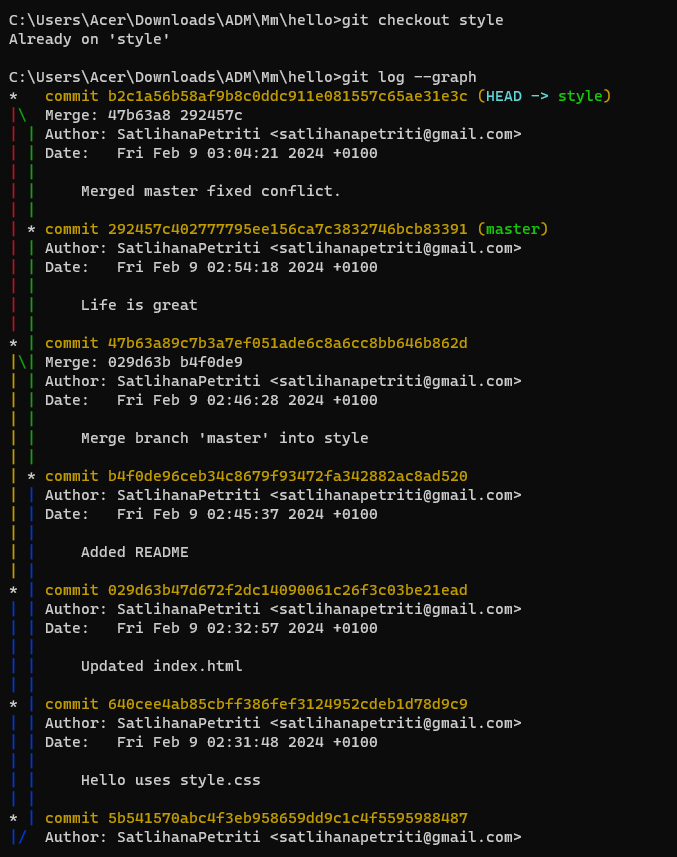
  
 pиc.78 Решение конфликта

1.22.3 Сделайте коммит решения конфликта  
Выполните:  
git add lib/hello.html  
git commit -m "Merged master fixed conflict."

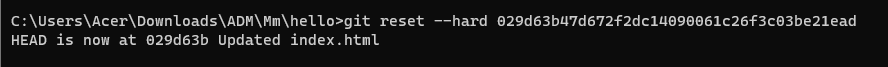
  
 pиc.79 коммит решения конфликта

1. **1.23 Сброс ветки style**

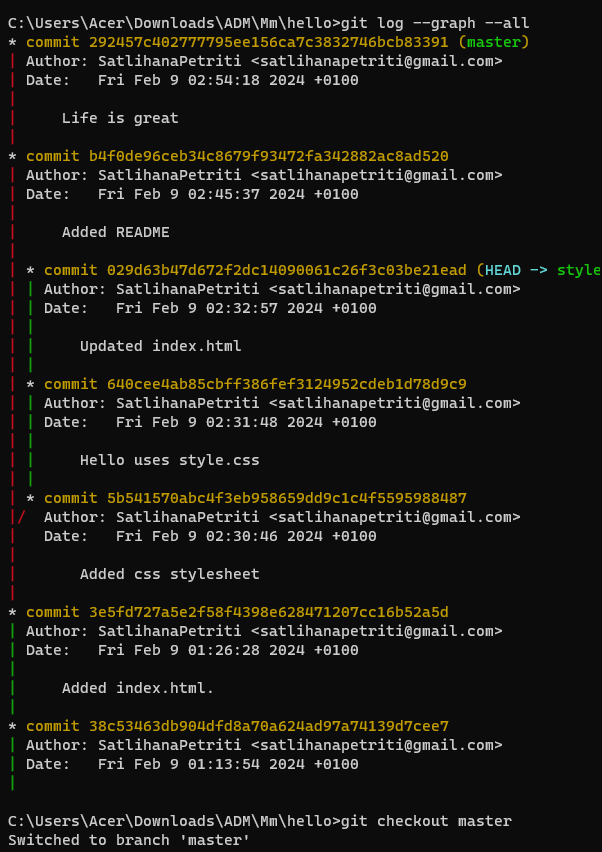
1.23.1 Сброс ветки style  
Вернемся на ветке style к точке перед тем, как мы слили ее с веткой master. Мыможем сбросить ветку к любому коммиту. По сути, это изменение указателя ветки  
на любую точку дерева коммитов.В этом случае мы хотим вернуться в ветке style в точку перед слиянием с master.  
Нам необходимо найти последний коммит перед слиянием. Выполните:  
git checkout style  
git log --graph

  
 pиc.80 Сброс ветки style

Мы видим, что коммит «Updated index.html» был последним на ветке style перед слиянием. Давайте сбросим ветку style к этому коммиту.  
Выполните:  
git reset --hard <hash>

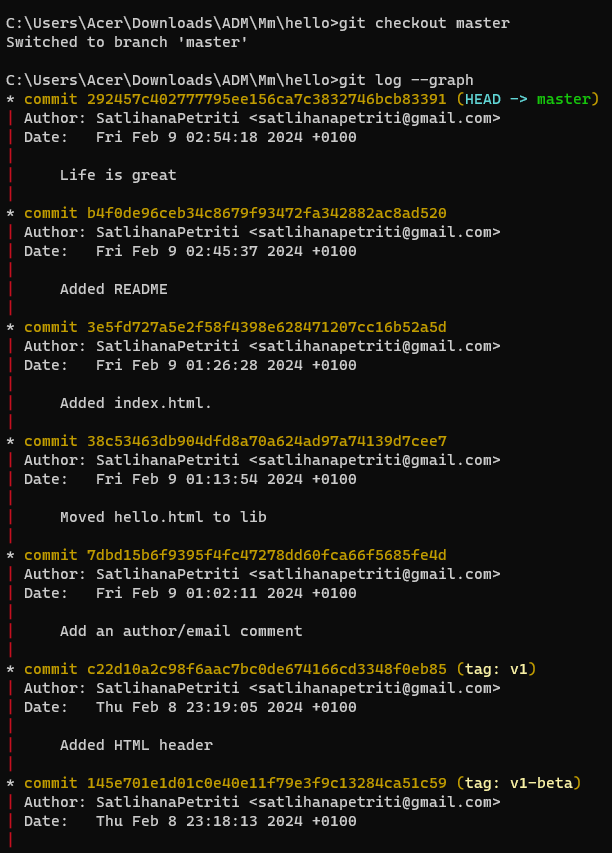
  
 pиc.81 сбросим ветку style к этому коммиту

1.23.2 Проверьте ветку.  
Поищите лог ветки style. У нас в истории больше нет коммитов слияний.Выполните:  
git log --graph --all

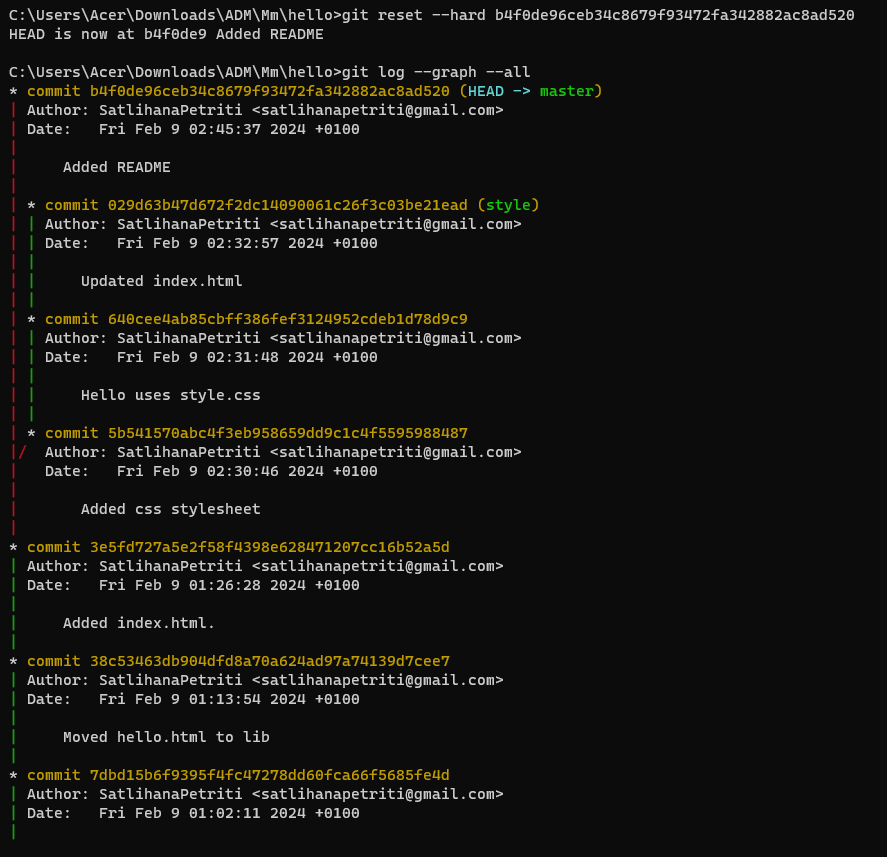
  
 pиc.82 Проверка ветки

1. **1.24 Сброс ветки master**

1.24.1 Сброс ветки master  
Добавив интерактивный режим в ветку master, мы внесли изменения, конфликтующие с изменениями в ветке style. Давайте вернемся в ветке master в точкуперед внесением конфликтующих изменений. Это позволяет нам продемонстрировать работу команды git rebase, не беспокоясь о конфликтах.  
Выполните:  
git checkout master  
git log --graph

  
 pиc.83 Сброс ветки master

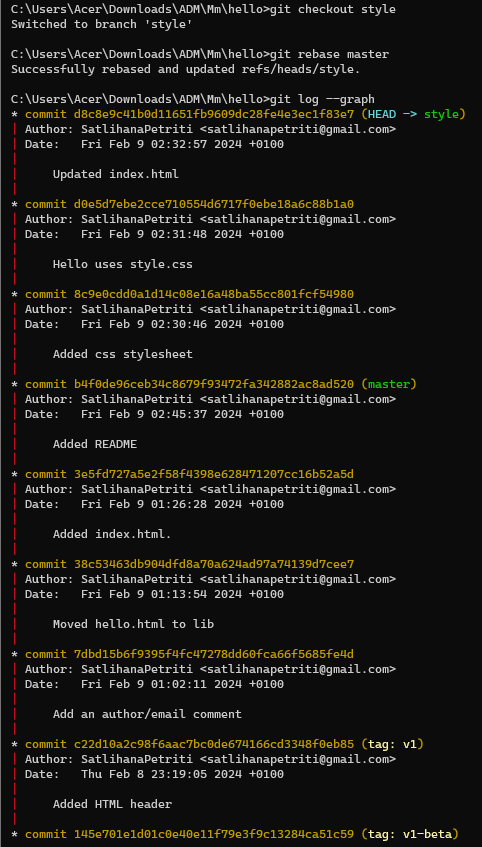
Коммит «Added README» идет непосредственно перед коммитом конфликтующего интерактивного режима. Мы сбросим ветку master к коммиту «AddedREADME».  
Выполните:  
git reset --hard <hash>  
git log --graph --all

  
 pиc.84 сбросим ветку master к коммиту «AddedREADME»

Просмотрите лог. Он должен выглядеть, как будто репозиторий был перемотан назад во времени к точке до какого-либо слияния.

1. **1.25 Перебазирование**

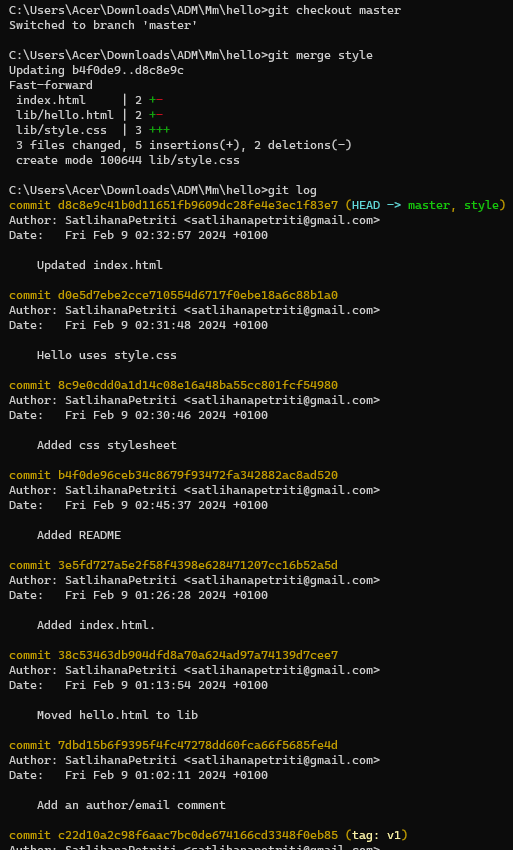
Используем команду rebase вместо команды merge. Мы вернулись в точку до первого слияния и хотим перенести изменения из ветки master в нашу ветку style.На этот раз для переноса изменений из ветки master мы будем использовать команду git rebase вместо слияния.  
Выполните:  
git checkout style  
git rebase master  
git log --graph

  
 pиc.85 Перебазирование

1.25.1 Слияние VS перебазирование  
Конечный результат перебазирования очень похож на результат слияния. Веткаstyle в настоящее время содержит все свои изменения, а также все измененияветки master. Однако, дерево коммитов значительно отличается. Дерево коммитов ветки style было переписано таким образом, что ветка master являетсячастью истории коммитов. Это делает цепь коммитов линейной и гораздо болеечитабельной.  
Не используйте перебазирование:  
• если ветка является публичной и расшаренной, поскольку переписывание  
общих веток будет мешать работе других членов команды;  
• когда важна точная история коммитов ветки, так как команда rebase переписывает историю коммитов;  
Учитывая приведенные выше рекомендации, рекомендуется использовать gitrebase для кратковременных, локальных веток, а слияние для веток в публичномрепозитории.

1. **1.26 Слияние в ветку master**

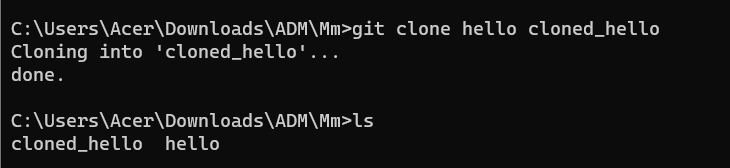
Мы поддерживали соответствие ветки style с веткой master (с помощью rebase),теперь давайте сольем изменения style в ветку master.  
 1.26.1 Слияние style в master  
Выполните:  
git checkout master  
git merge style

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.  
 1.26.2 Просмотрите логи  
Выполните:  
git log  
Теперь ветки style и master идентичны.  
  
 pиc.86 Слияние в ветку master

1. **1.27 Клонирование репозиториев**

1.27.1 Перейдите в рабочий каталог  
Перейдите в рабочий каталог и сделайте клон вашего репозитория hello.Выполните:  
cd ..  
pwd  
ls

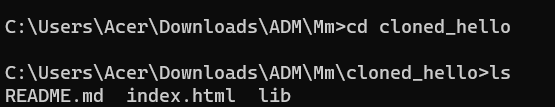
fig:  
 pиc.87 Перейдите в рабочий каталог  
Сейчас мы находимся в рабочем каталоге.  
В этот момент вы должны находиться в «рабочем» каталоге. Здесь должен бытьединственный репозиторий под названием «hello».  
 1.27.2 Создайте клон репозитория hello  
Создадим клон репозитория.Выполните:  
git clone hello cloned\_hello  
ls

  
 pиc.88 Создайте клон репозитория hello

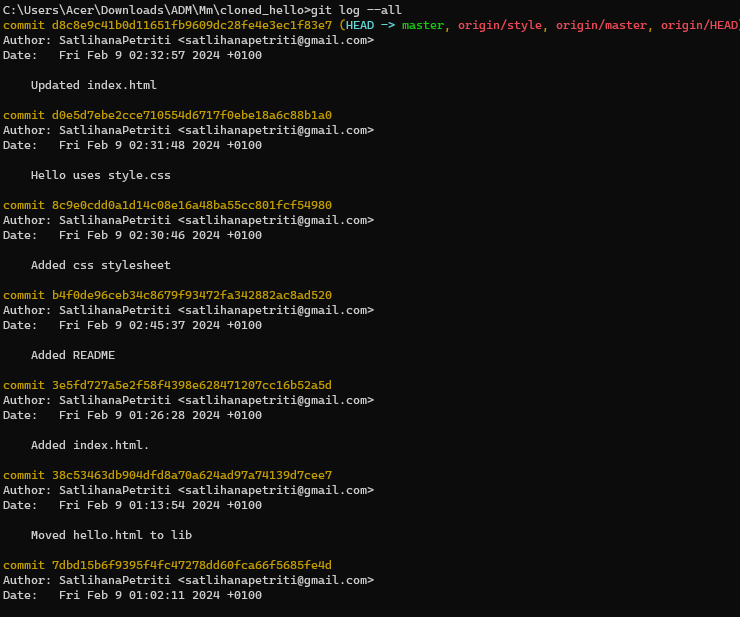
В вашем рабочем каталоге теперь должно быть два репозитория: оригинальный репозиторий «hello» и клонированный репозиторий «cloned\_hello»

1. **1.28 Просмотр клонированного репозитория**

1.28.1 Давайте взглянем на клонированный репозиторий.  
Выполните:  
cd cloned\_hello  
ls

  
 pиc.89 взглянем на клонированный репозиторий

Вы увидите список всех файлов на верхнем уровне оригинального репозиторияREADME.md, index.html и lib.  
 1.28.2 Просмотрите историю репозитория  
Выполните:  
git log --all

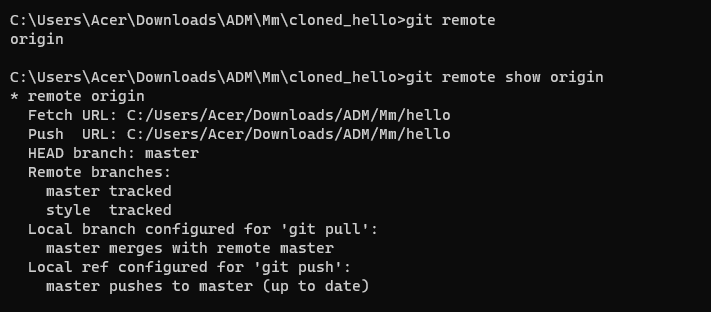
  
 pиc.90 история репозитории

Вы увидите список всех коммитов в новый репозиторий, и он должен (более или менее) совпадать с историей коммитов в оригинальном репозитории. Единственная разница должна быть в названиях веток.

Вы увидите ветку master (HEAD) в списке истории. Вы также увидите ветки со странными именами (origin/master, origin/style и origin/HEAD).

1. **1.29 Что такое origin?**

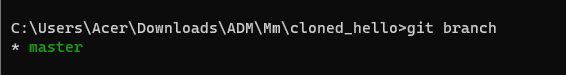
Выполните:  
git remote  
Мы видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию об имени по умолчанию:  
Выполните:  
git remote show origin

  
 pиc.91 получить более подробную информацию об имени по умолчанию

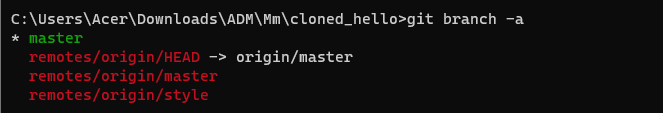
Удаленные репозитории обычно размещаются на отдельной машине, возможно, централизованном сервере. Однако, как мы видим здесь, они могут с тем же успехом указывать на репозиторий на той же машине. Нет ничего особенного в имени «origin», однако существует традиция использовать «origin» в качестве имени первичного централизованного репозитория (если таковой имеется).

1. **1.30 Удаленные ветки**

Давайте посмотрим на ветки, доступные в нашем клонированном репозитории. Выполните:  
git branch  
Как мы видим, в списке только ветка master. Команда gitbranch выводит только список локальных веток по умолчанию.

  
 pиc.92 Удаленные ветки

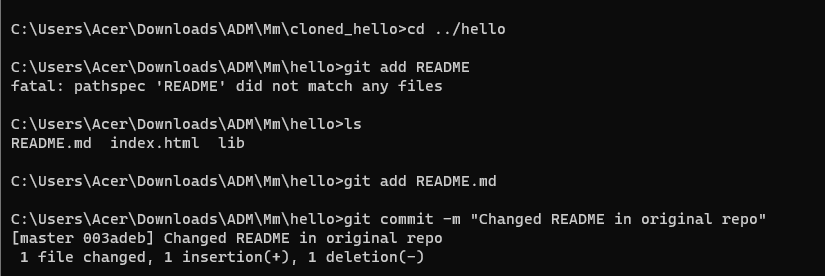
1.30.1 Список удаленных веток  
Для того, чтобы увидеть все ветки, попробуйте следующую команду:  
git branch -a

  
 pиc.93 Список удаленных веток

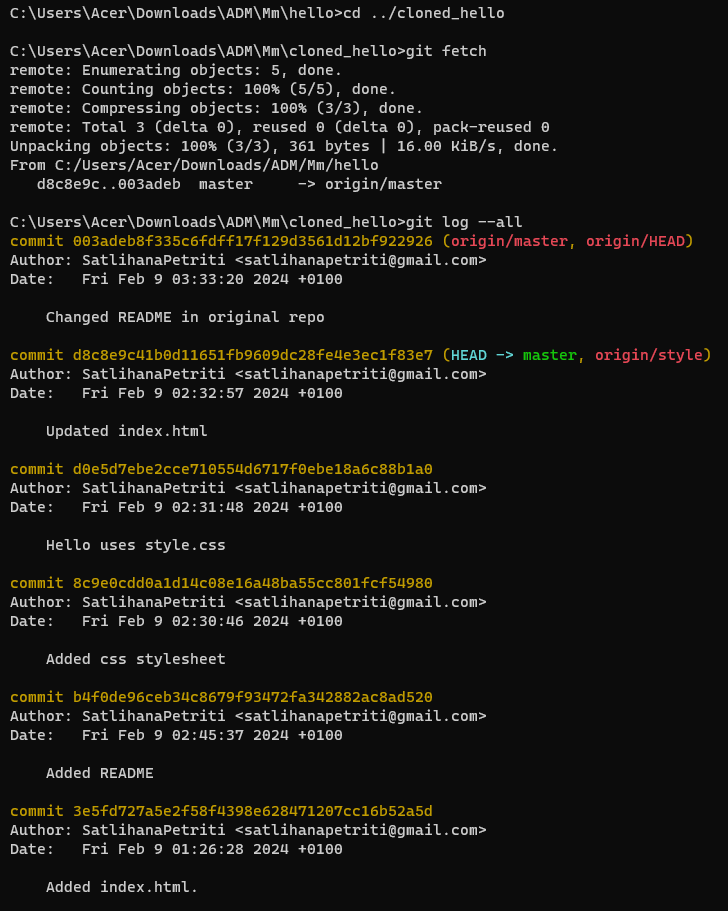
Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные. Если мы хотим собственную веткуstyle, мы должны сами ее создать. Через минуту вы увидите, как это делается.

1. **1.31 Изменение оригинального репозитория**

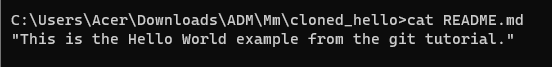
Внесите некоторые изменения в оригинальный репозиторий, чтобы затем попытаться извлечь и слить изменения из удаленной ветки в текущую  
 1.31.1 Внесите изменения в оригинальный репозиторий hello  
Выполните:  
cd ../hello

  
 pиc.94 Внесите изменения в оригинальный репозиторий hello

Примечание: Сейчас мы находимся в репозитории helloВнесите следующие изменения в файл README.md:Файл README.md  
This is the Hello World example from the git tutorial.  
Теперь добавьте это изменение и сделайте коммитВыполните:  
git add README  
git commit -m "Changed README in original repo"  
Теперь в оригинальном репозитории есть более поздние изменения, которыхнет в клонированной версии. Далее мы извлечем и сольем эти изменения в клонированный репозиторий.  
 1.31.2 Извлечение изменений  
Научиться извлекать изменения из удаленного репозитория. Выполните:  
cd ../cloned\_hello  
git fetch  
git log --all

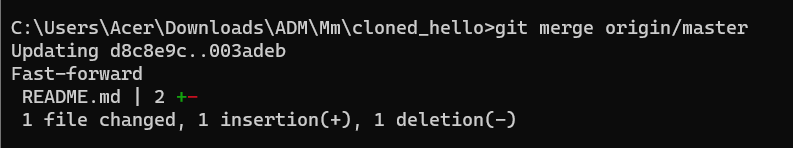
  
 pиc.95 Извлечение изменений

1.31.3 Проверьте README.md  
Мы можем продемонстрировать, что клонированный файл README.md не изменился. Выполните:  
cat README

  
 pиc.96 Проверка README.md

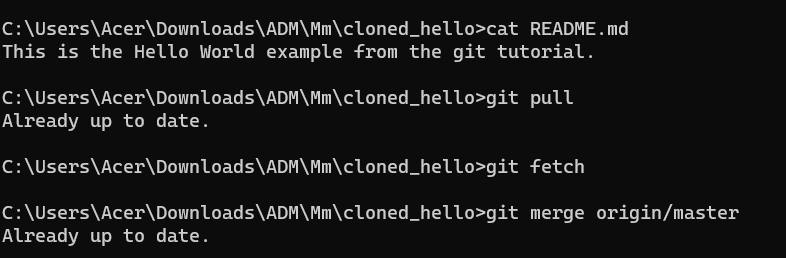
1. **1.32 Слияние извлеченных изменений**

1.32.1 Слейте извлеченные изменения в локальную ветку master  
Выполните:  
git merge origin/master

  
 pиc.97 Слейте извлеченные изменения в локальную ветку master

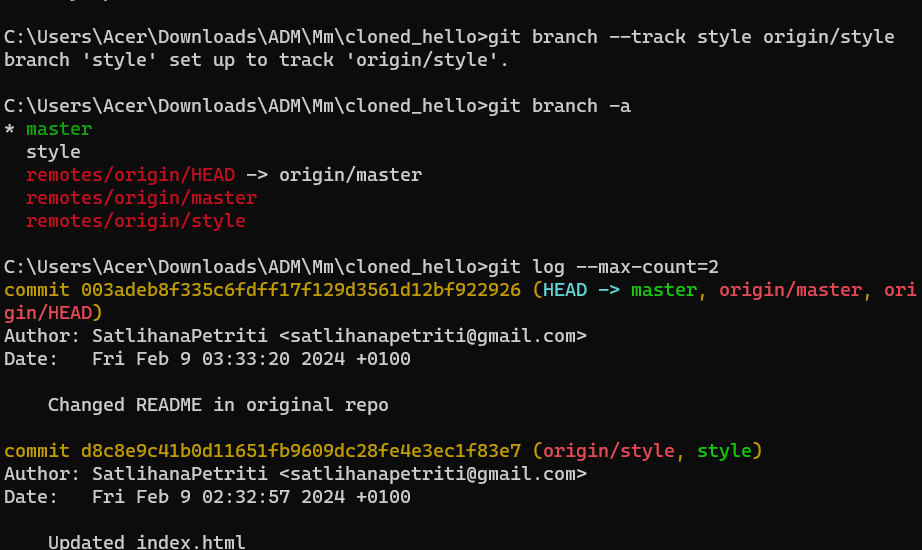
1.32.2 Еще раз проверьте файл README.md  
Сейчас мы должны увидеть изменения.Выполните:  
cat README.md

Хотя команда git fetch не сливает изменения, мы можем вручную слить изменения из удаленного репозитория.  
Теперь давайте рассмотрим объединение fetch и merge в одну команду. Выполнение:  
git pull  
эквивалентно двум следующим шагам:  
git fetch  
git merge origin/master

  
 pиc.98 Проверка README.md

1. **1.33 Добавление ветки наблюдения**

Ветки, которые начинаются с remotes/origin являются ветками оригинального репозитория. Обратите внимание,что у вас больше нет ветки под названием style,но система контроля версий знает, что в оригинальном репозитории ветка styleбыла.  
 1.33.1 Добавьте локальную ветку, которая отслеживает удаленную ветку  
Выполните:  
git branch --track style origin/style  
git branch -a  
git log --max-count=2

  
 pиc.99 Добавьте локальную ветку, которая отслеживает удаленную ветку

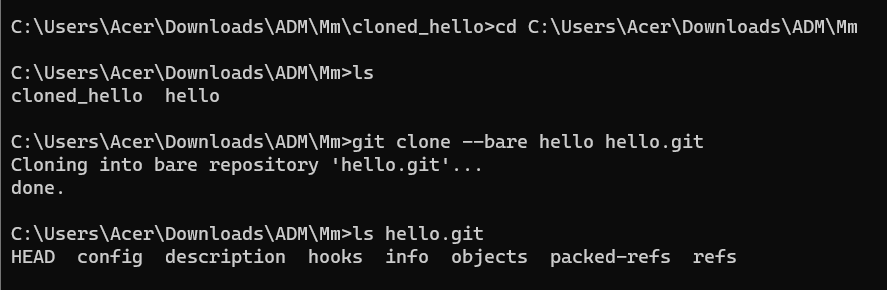
Теперь мы можем видеть ветку style в списке веток и логе

1. **1.34 Чистые репозитории**

Чистые репозитории (без рабочих каталогов) обычно используются для расшаривания. Обычный git-репозиторий подразумевает, что вы будете использовать его как рабочую директорию, поэтому вместе с файлами проекта в актуальной версии, git хранит все служебные, «чисто-репозиториевские» файлы в поддиректории .git. В удаленных репозиториях нет смысла хранить рабочие файлы на диске(как это делается в рабочих копиях), а все что им действительно нужно — это дельты изменений и другие бинарные данные репозитория. Вот это и есть «чистый репозиторий».

1. **1.35 Создайте чистый репозиторий**

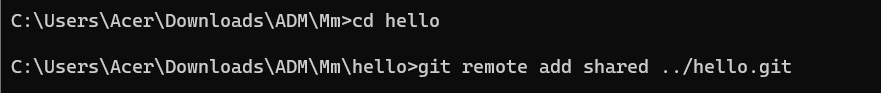
cd ..  
git clone --bare hello hello.git  
ls hello.git

  
 pиc.100 Создам чистый репозиторий

Сейчас мы находимся в рабочем каталоге  
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.

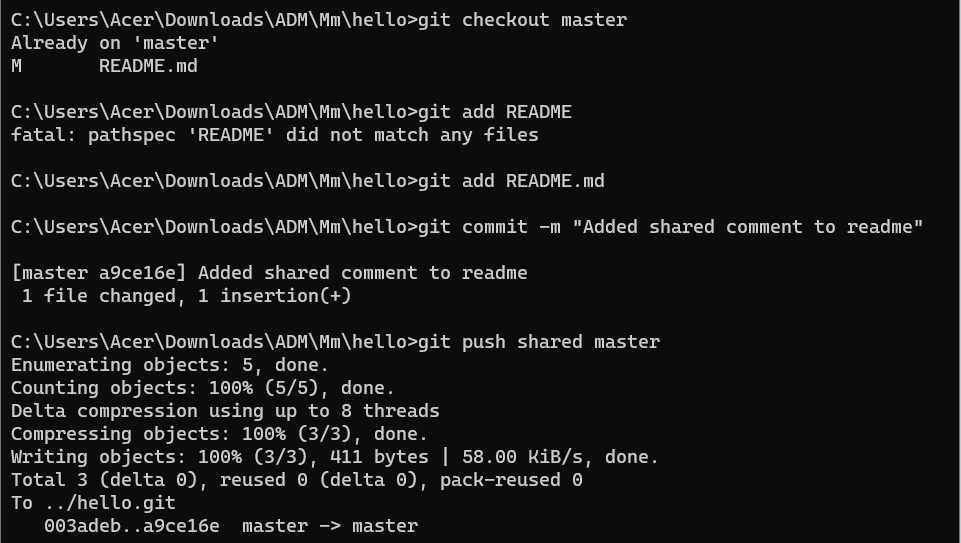
1. **1.36 Добавление удаленного репозитория**

Давайте добавим репозиторий hello.git к нашему оригинальному репозиторию.  
cd hello  
git remote add shared ../hello.git

  
 pиc.101 Добавление удаленного репозитория

1. **1.37 Отправка изменений**

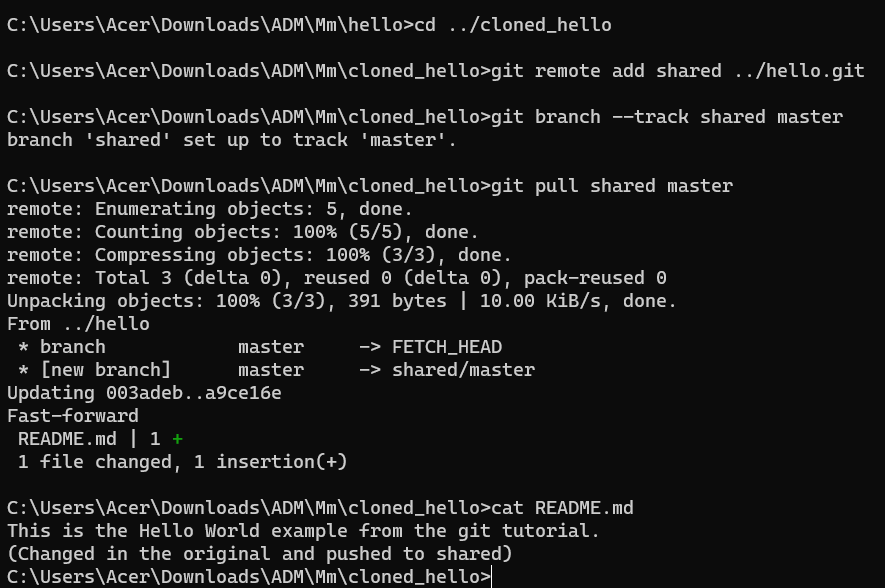
Так как чистые репозитории, как правило, расшариваются на каком-нибудь сетевом сервере, нам необходимо отправить наши изменения в другие репозитории. Начнем с создания изменения для отправки. Отредактируйте файл README.md исделайте коммит  
Файл README.md:  
This is the Hello World example from the git tutorial.  
(Changed in the original and pushed to shared)  
Выполните:  
git checkout master  
git add README  
git commit -m "Added shared comment to readme"  
Теперь отправьте изменения в общий репозиторий. Выполните:  
git push shared master

  
 pиc.102 Отправка изменений

Общим называется репозиторий, получающий отправленные нами изменения.

1. **1.38 Извлечение общих изменений**

Научиться извлекать изменения из общего репозитория. Быстро переключитесь вклонированный репозиторий и извлеките изменения, только что отправленные в  
общий репозиторий. Выполните:  
cd ../cloned\_hello  
Сейчас мы находимся в репозитории cloned\_hello.Выполните:  
git remote add shared ../hello.git  
git branch --track shared master  
git pull shared master  
cat README.md

  
 pиc.102 Извлечение общих изменений

## Вывод

Я научилась работать с Git и познакомилась с его инструментами и утилитами. Это включает в себя развитие практического понимания рабочего процесса Git для эффективного контроля версий при разработке программного обеспечения.