Лабораторная работа №1

Работа с git

Гнатюк Анастасия Станиславовна

Содержание

[Цель работы 2](#_Toc158456488)

[Теоретическое введение 2](#_Toc158456489)

[Выполнение лабораторной работы 2](#_Toc158456490)

[1.1 Подготовка 2](#_Toc158456491)

[1.1.1 Установка имени и электронной почты 2](#_Toc158456492)

[1.1.2 Параметры установки окончаний строк 2](#_Toc158456493)

[1.1.3 Установка отображения unicode 3](#_Toc158456494)

[1.2 Создание проекта 3](#_Toc158456495)

[1.2.1 Создайте страницу «Hello, World» 3](#_Toc158456496)

[1.2.2 Создание репозитория и добавление файла в репозиторий 3](#_Toc158456497)

[1.2.3 Проверка состояние репозитория 4](#_Toc158456498)

[1.3 Внесение изменений 4](#_Toc158456499)

[1.3.1 Измените страницу «Hello, World» 4](#_Toc158456500)

[1.4 Индексация изменений 6](#_Toc158456501)

[1.4.1 Коммит изменений 6](#_Toc158456502)

[1.4.2 Добавьте стандартные теги страницы 7](#_Toc158456503)

[1.4.3 История 10](#_Toc158456504)

[1.4.4 Получение старых версий 11](#_Toc158456505)

[1.4.5 Создание тегов версий 12](#_Toc158456506)

[1.8 Удаление коммиттов из ветки 13](#_Toc158456507)

[1.9 Работа с ветками 14](#_Toc158456508)

[1.9.1 Создайте ветку 14](#_Toc158456509)

[1.9.2 Просмотрите текущие ветки 14](#_Toc158456510)

[1.9.3 Слияние веток 16](#_Toc158456511)

[1.9.4 Сброс ветки style 16](#_Toc158456512)

[1.9.5 Перебазирование 17](#_Toc158456513)

[Вывод 18](#_Toc158456514)

# Цель работы

Целью данной работы является изучение работы с git, а именно знакомство с командами, с помощью которых мы сможем работать с репозиториями, файлами, папками и ветками.

# Теоретическое введение

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

Среди проектов, использующих Git, — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Программа является свободной и выпущена под лицензией GNU GPL версии 2. По умолчанию используется TCP-порт 9418.

# Выполнение лабораторной работы

## 1.1 Подготовка

### 1.1.1 Установка имени и электронной почты

git config –global user.name “Your Name”

git config –global user.email “your\_email@whatever.com”

### 1.1.2 Параметры установки окончаний строк

Настройка core.autocrlf с параметрами true и input делает все переводы строк текстовых файлов в главном репозитории одинаковы. core.autocrlf true - git автоматически конвертирует CRLF->LF при коммите и обратно LF->CRLF при выгрузке кода из репозитория на файловую систему (используют в Windows). core.autocrlf input - конвертация CRLF в LF только при коммитах (используют в Mac/Linux).

Если core.safecrlf установлен в true или warm, git проверяет, если преобразование является обратимым для текущей настройки core.autocrlf. core.safecrlf true - отвержение необратимого преобразования lf<->crlf.

Полезно, когда специфические бинарники похожие на текстовые файлы. core.safecrlf warn - печать только предупреждение, но принимает необратимый переход.

Для пользователей Windows:

git config –global core.autocrlf true

git config –global core.safecrlf true

### 1.1.3 Установка отображения unicode

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

git config –global core.quotepath off

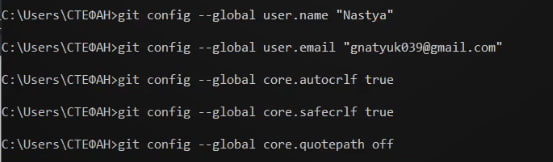


Рис.1:

## 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

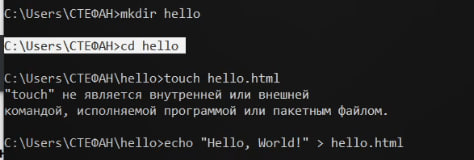


Рис.2:

### 1.2.2 Создание репозитория и добавление файла в репозиторий

Добавим файл в репозиторий.

git add hello.html

git commit -m “Initial Commit”

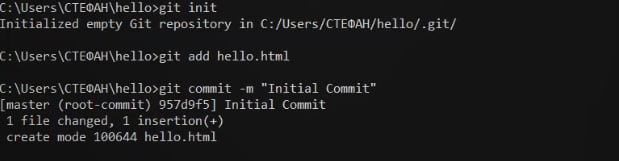


Рис.3:

### 1.2.3 Проверка состояние репозитория

Используйте команду git status, чтобы проверить текущее состояние репозитория.

git status

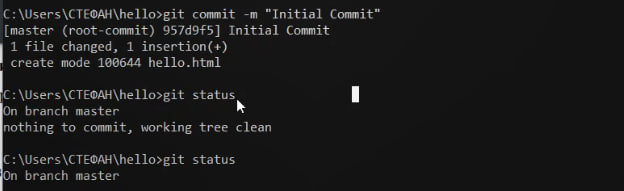


Рис.4:

## 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:

Рис.6:

Рис.6:

Проверьте состояние рабочего каталога.

git status

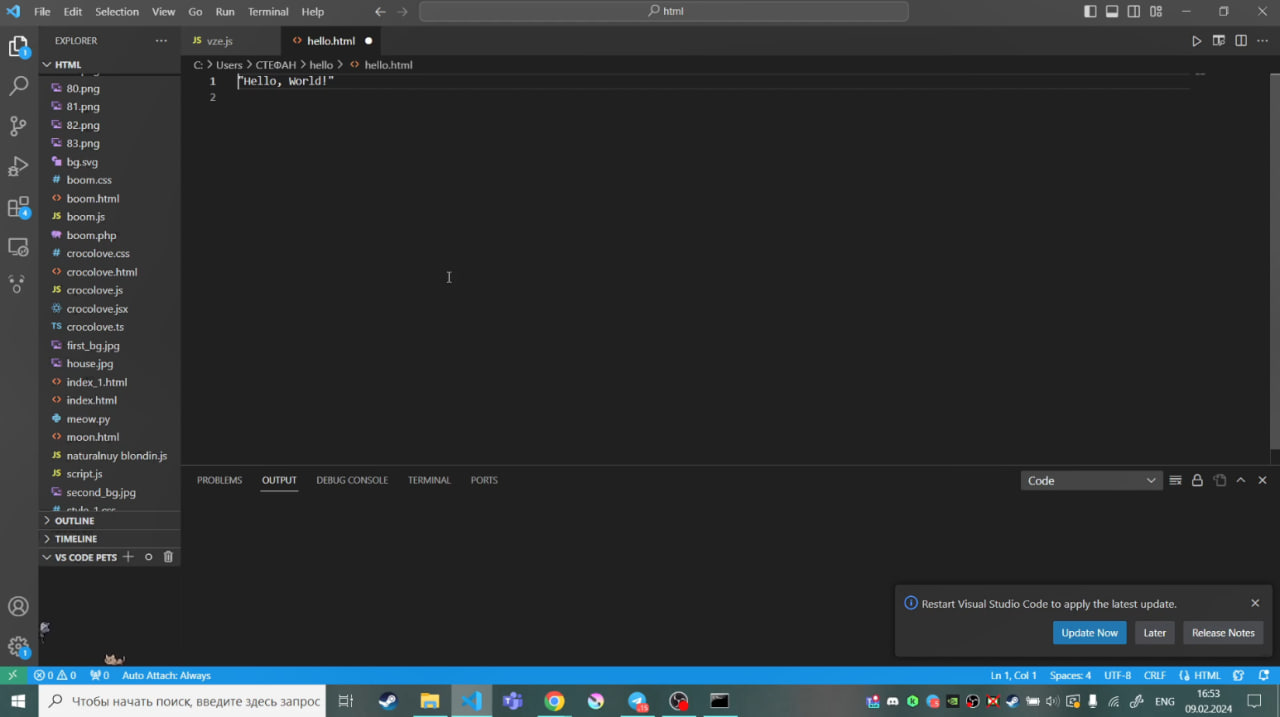


Рис.5:

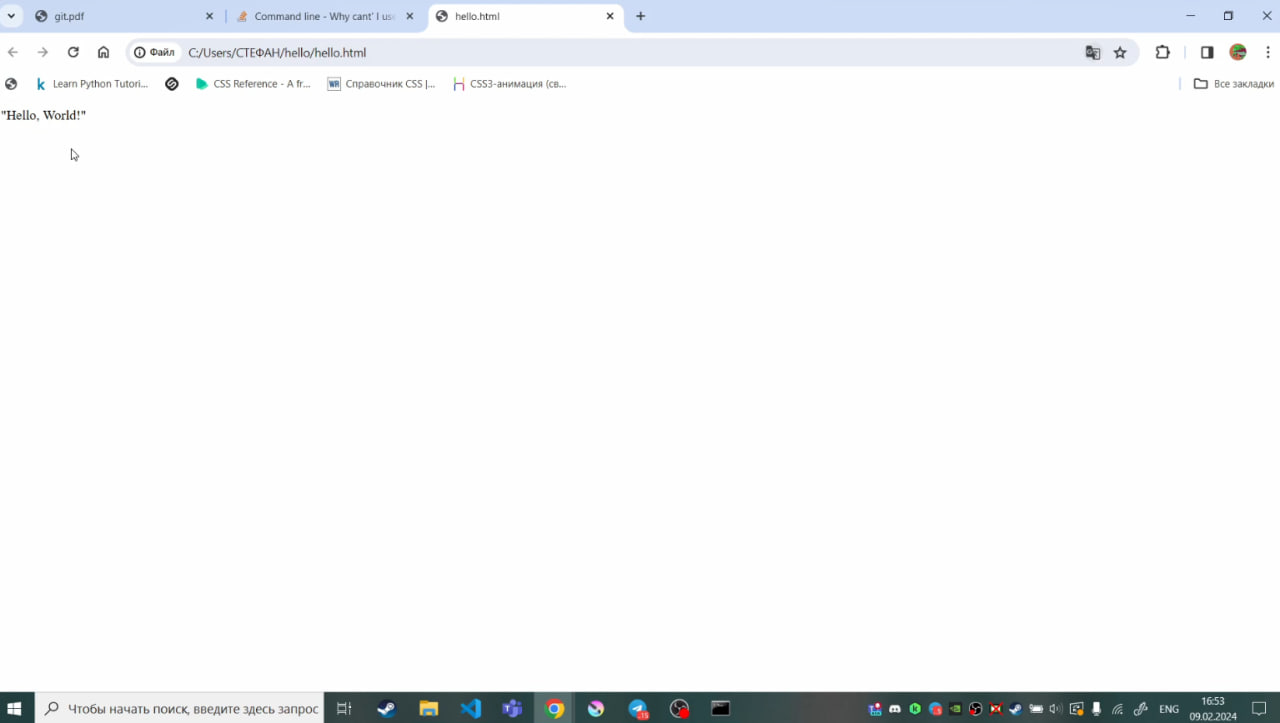


Рис.7:

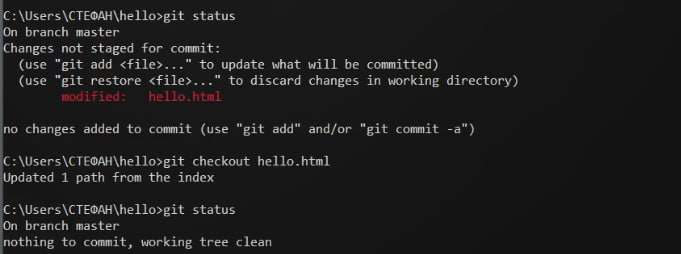


Рис.8:

## 1.4 Индексация изменений

### 1.4.1 Коммит изменений

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

git add hello.html

git status

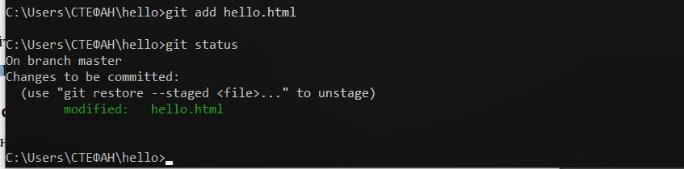


Рис.9:

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

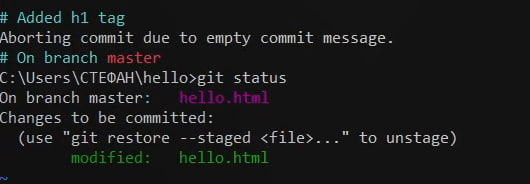


Рис.10:

### 1.4.2 Добавьте стандартные теги страницы

Измените страницу «Hello, World», чтобы она содержала стандартные теги html и body.

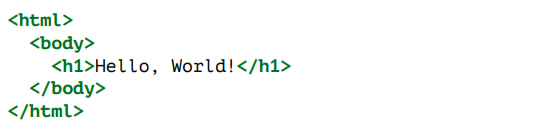


Рис.11:

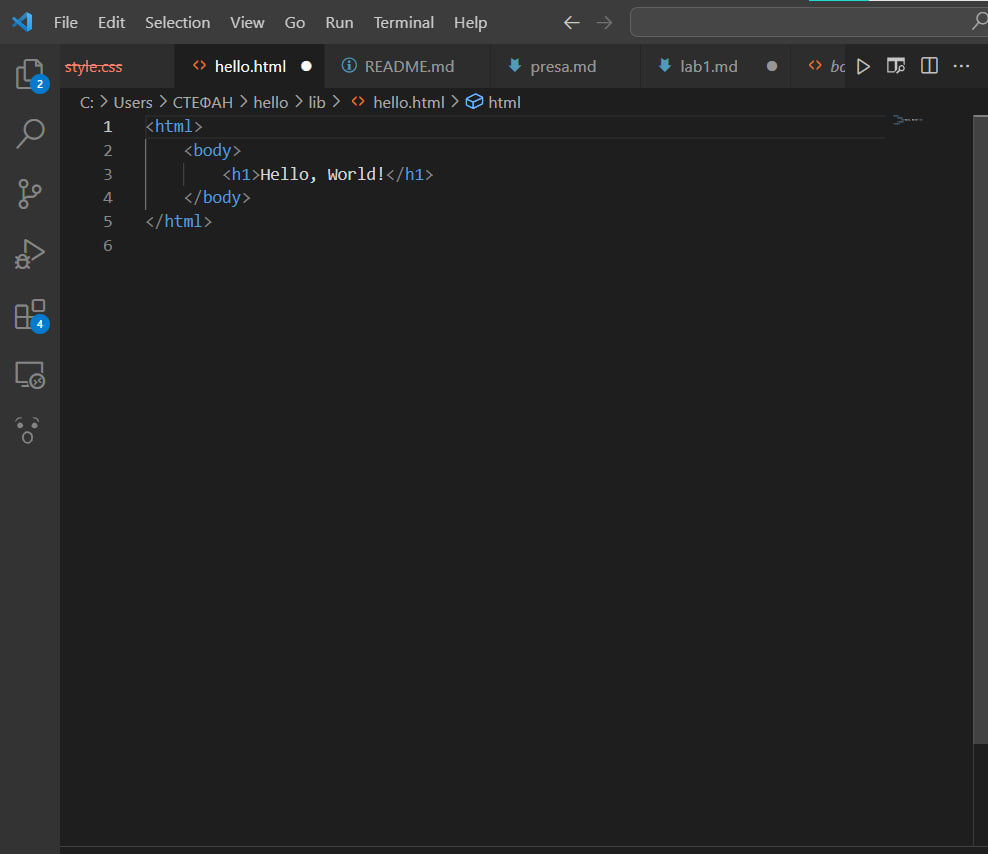


Рис.12:

Теперь добавьте это изменение в индекс git.

git add hello.html

Теперь добавьте заголовки HTML (секцию head) к странице «Hello, World».

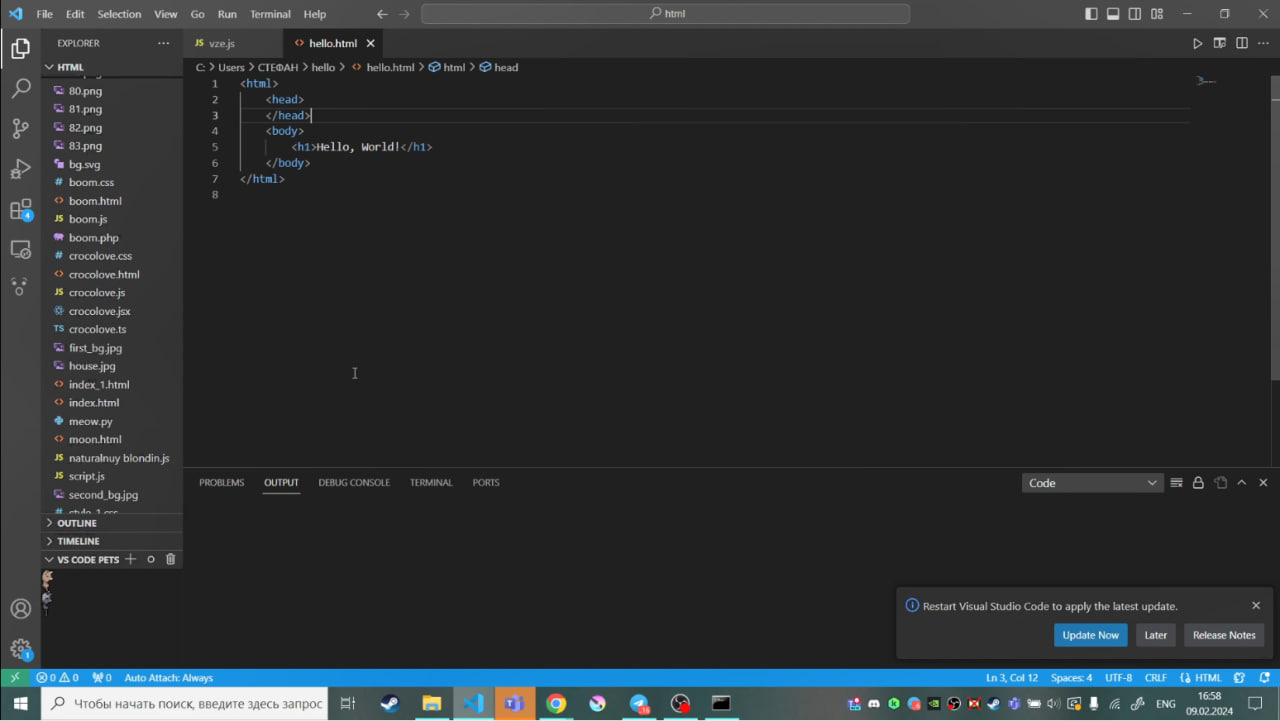


Рис.13:

Проверьте текущий статус:

git status

Обратите внимание на то, что hello.html указан дважды в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) является непроиндексированным. Если бы вы делали коммит сейчас, заголовки не были бы сохранены в репозиторий.

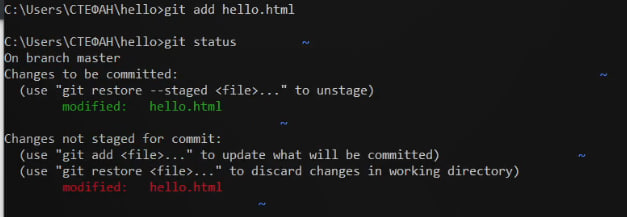


Рис.14:

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

git commit -m “Added standard HTML page tags”

git status

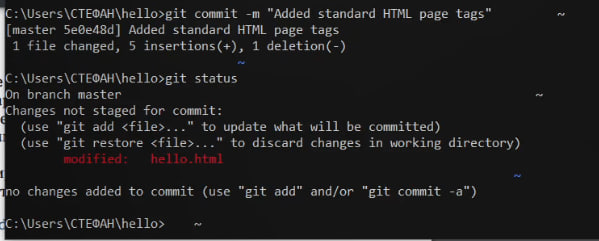


Рис.15:

Состояние команды говорит о том, что hello.html имеет незафиксированные изменения, но уже не в буферной зоне.

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

git add .

git status

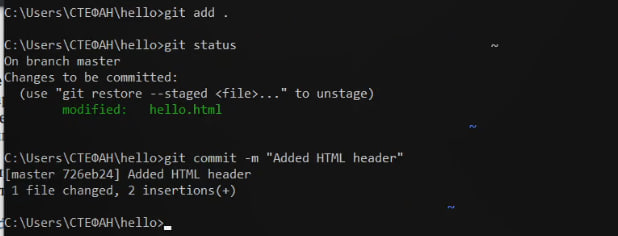


Рис.16:

### 1.4.3 История

Получим список произведенных изменений:

git log

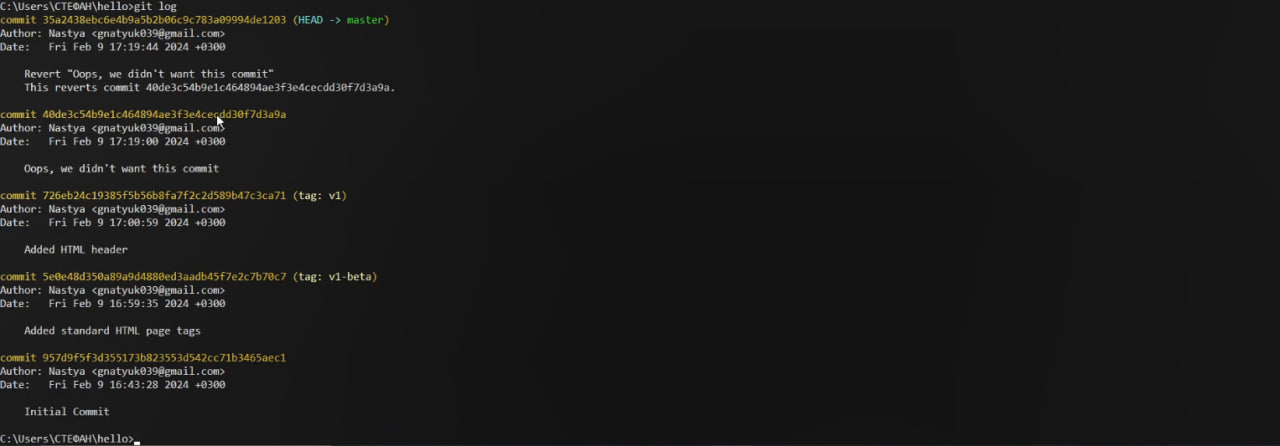


Рис.17:

### 1.4.4 Получение старых версий

Получите хэши предыдущих версий

git log

Изучите данные лога и найдите хэш для первого коммита. Он должен быть в последней строке данных. Используйте этот хэш-код (достаточно первых 7 знаков) в команде ниже. Затем проверьте содержимое файла hello.html.

git checkout “hash”

cat hello.html

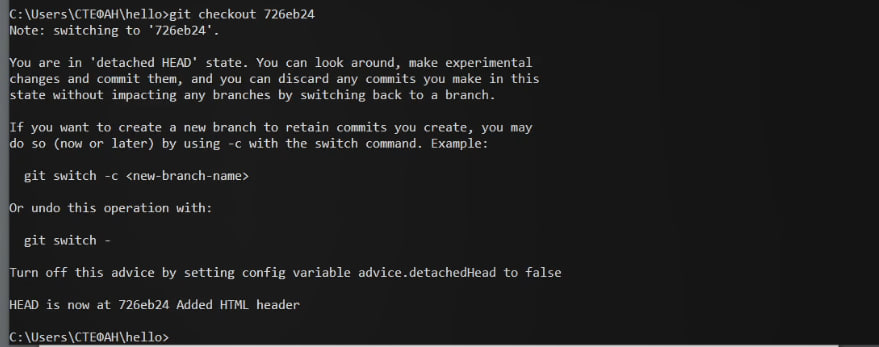


Рис.18:

Вернитесь к последней версии в ветке master

git checkout master

cat hello.html

Рис.19:

Рис.19:

master — имя ветки по умолчанию. Переключая имена веток, вы попадаете на последнюю версию выбранной ветки.

### 1.4.5 Создание тегов версий

Давайте назовем текущую версию страницы hello первой (v1).

Создайте тег первой версии

git tag v1

Теперь текущая версия страницы называется v1.

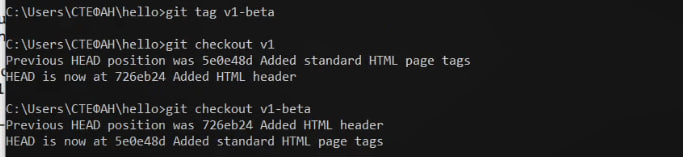
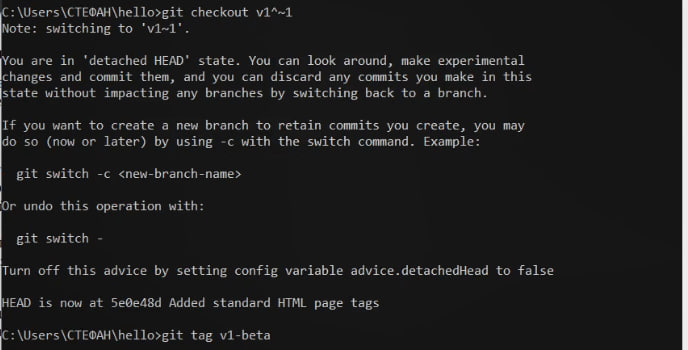
Теги для предыдущих версий Давайте создадим тег для версии, которая идет перед текущей версией и назовем его v1-beta. В первую очередь нам надо переключиться на предыдущую версию. Вместо поиска до хэш, мы будем использовать ^, обозначающее «родитель v1». Вместо обозначения v1^ можно использовать v1~1. Это обозначение можно определить как «первую версию предшествующую v1».

git checkout v1^

cat hello.html

Это версия c тегами html и body, но еще пока без head. Давайте сделаем ее версией v1-beta.

git tag v1-beta

## 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 в истории ветки (при использовании команды git log).

Часто мы делаем коммит, и сразу понимаем, что это была ошибка. Было бы неплохо иметь команду «возврата», которая позволила бы нам сделать вид, что неправильного коммита никогда и не было. Команда «возврата» даже предотвратила бы появление нежелательного коммита в истории git log.

Давайте сделаем быструю проверку нашей истории коммитов. Выполните:

git log

Мы видим, что два последних коммита в этой ветке — «Oops» и «Revert Oops».

Давайте удалим их с помощью сброса.

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

git tag oops

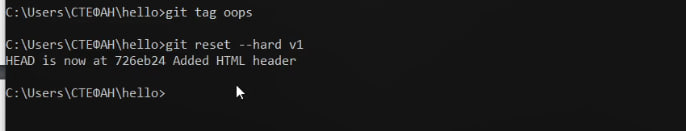


Рис.22:

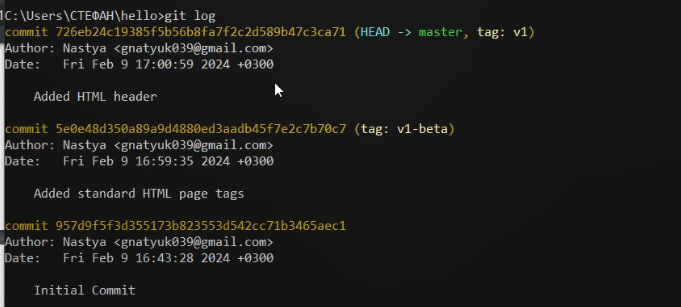


Рис.23:

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

Наша ветка master теперь указывает на коммит v1, а коммитов Oops и Revert Oops в ветке уже нет. Параметр –hard указывает, что рабочий каталог должен быть обновлен в соответствии с новым head ветки.

## 1.9 Работа с ветками

Пора сделать наш hello world более выразительным. Так как это может занять некоторое время, лучше переместить эти изменения в отдельную ветку, чтобы изолировать их от изменений в ветке master.

### 1.9.1 Создайте ветку

Давайте назовем нашу новую ветку «style».

Выполните:

git checkout -b style

git status

git checkout -b

Обратите внимание, что команда git status сообщает о том, что вы находитесь в ветке «style».

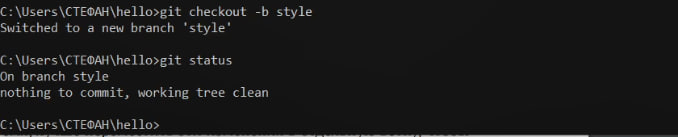


Рис.24:

### 1.9.2 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

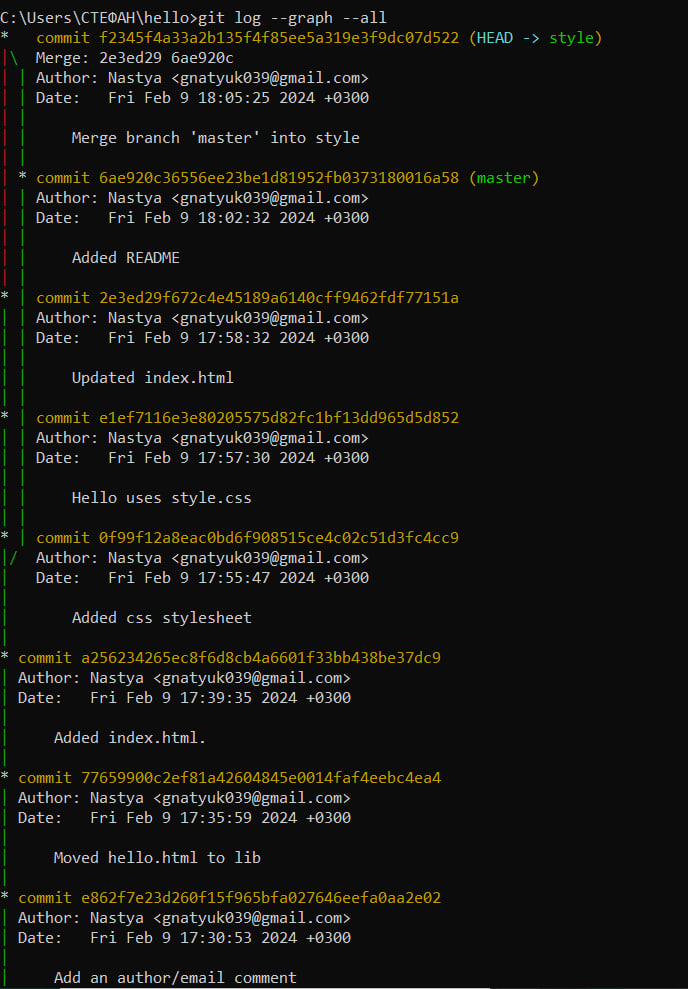


Рис.25:

### 1.9.3 Слияние веток

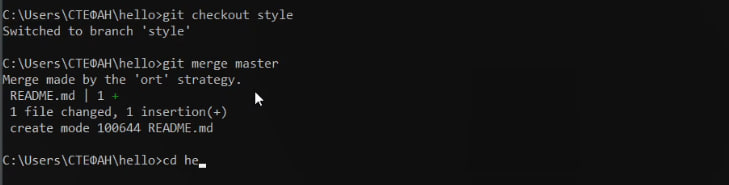
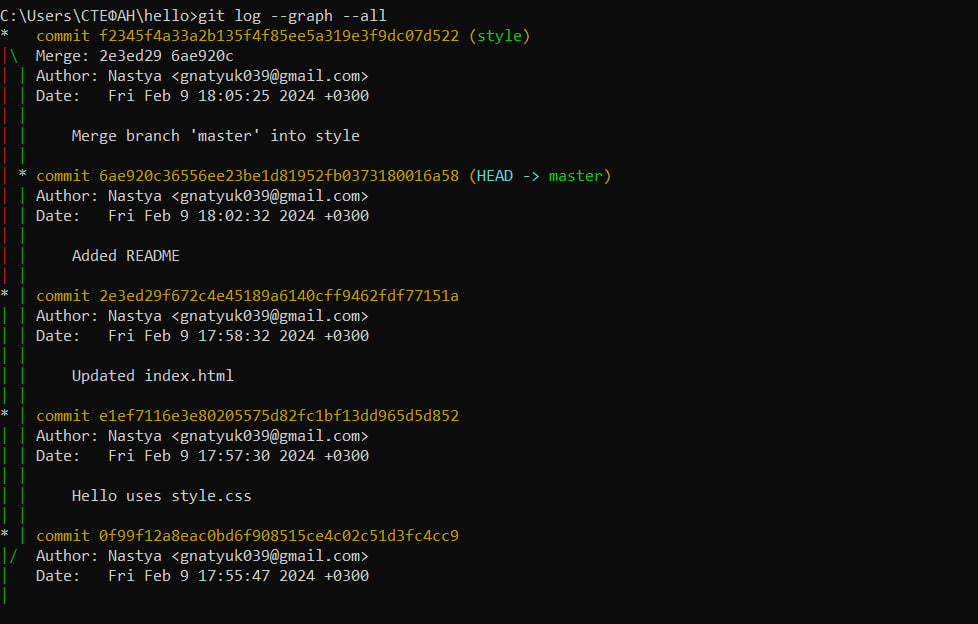
Слияние переносит изменения из двух веток в одну. Давайте вернемся к ветке style и сольем master с style.

Выполните:

git checkout style

git merge master

git log –graph –all

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

### 1.9.4 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.

Выполните:

git checkout style

git log –graph

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”

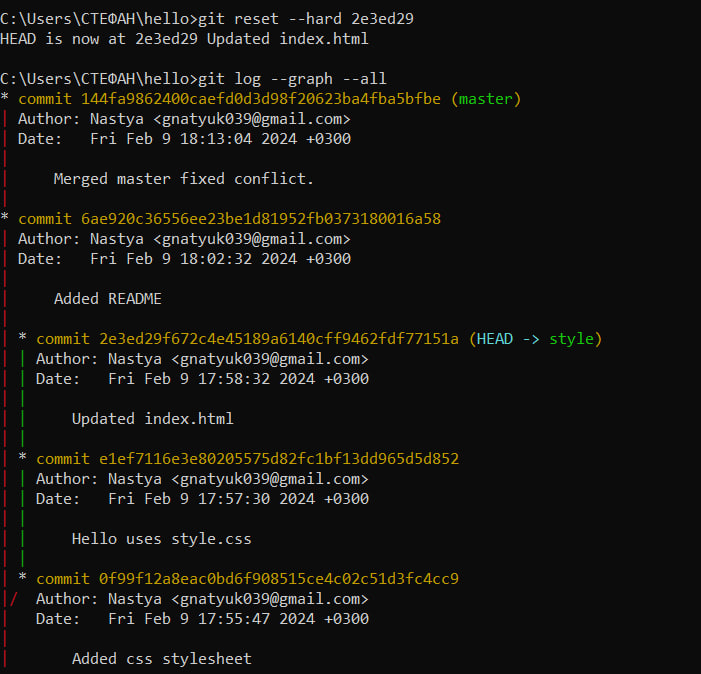


Рис.28:

### 1.9.5 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

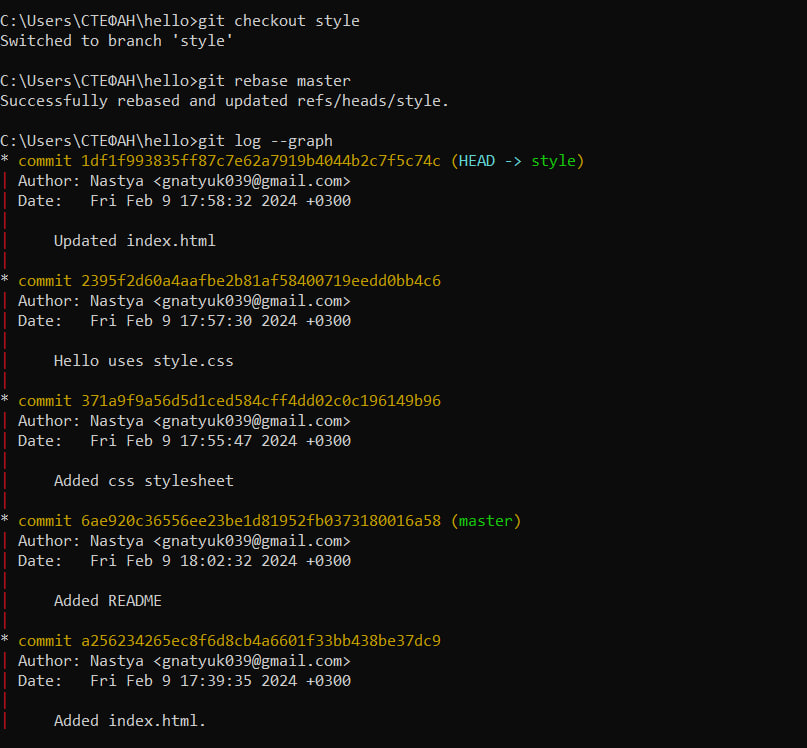


Рис.29:

# Вывод

Мы поработали с git, изучили и попробовали некоторые команды.