***Практическая работа № 2***

**Система контроля версий Git**

**Цель работы:** изучение принципов работы систем контроля версий, приобретение навыков работы в системе Git.

***Краткие теоретические сведения***

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Под версионный контроль можно поместить файлы практически любого типа.

СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Если, пользуясь СКВ, испортятся или потеряются файлы, всё можно будет легко восстановить. Накладные расходы будут очень маленькими.

***Практическая часть***

***Установка Git***

Рассмотрим последовательность действий по установке *Git*.

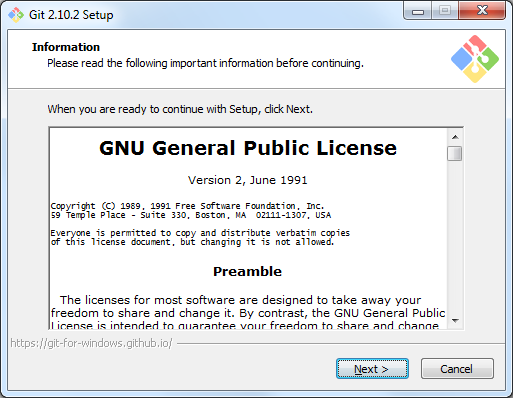


Рисунок 1 – Окно информации

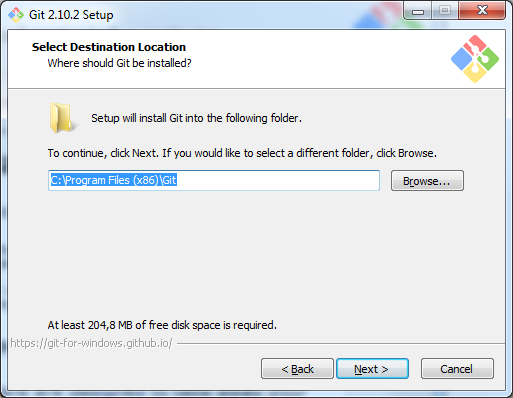


Рисунок 2 – Окно задания размещения

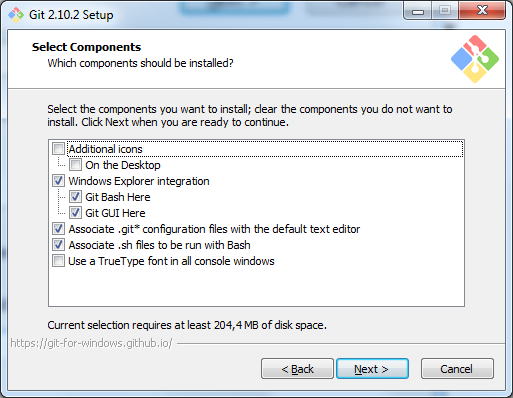


Рисунок 3 – Окно выбора компонентов

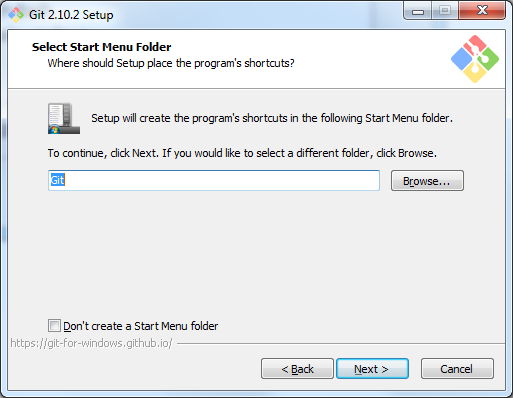


Рисунок 4 – Окно задания программного меню

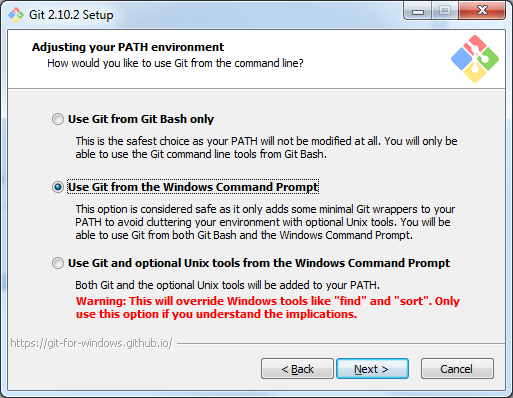


Рисунок 5 – Окно установки

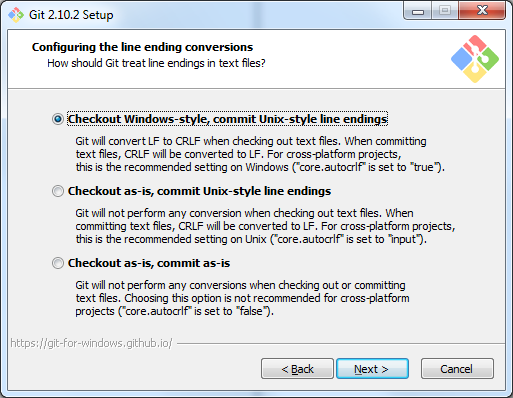


Рисунок 6 – Окно конфигурирования

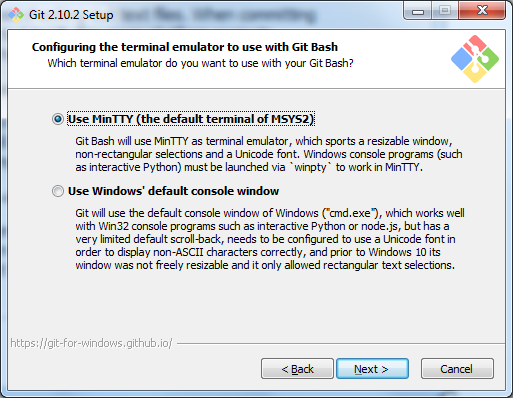


Рисунок 7 – Окно задания терминала

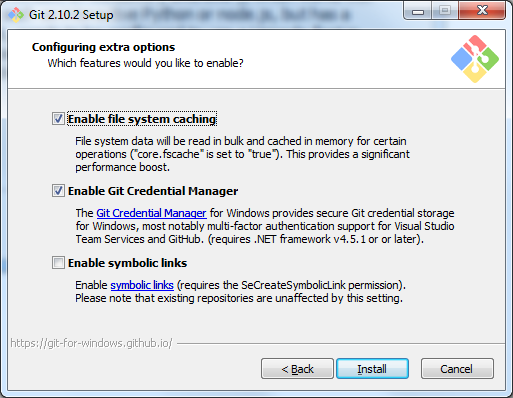


Рисунок 8 – Окно задания дополнительных опций

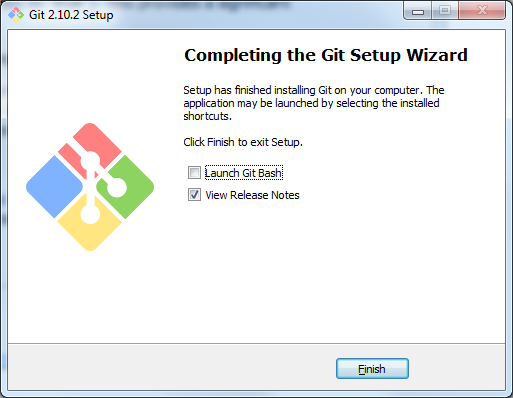


Рисунок 9 – Окно завершения установки

***Выполнение работы***

**0. Конфигурирование**

Перед тем как начать работу с *Git* следует установить имя и адрес электронной почты (рисунок 10). Это выполняется с командной строки.

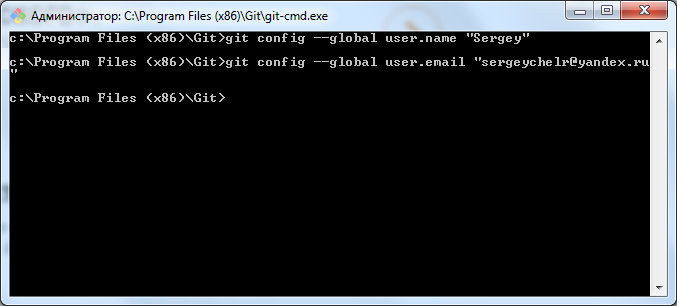


Рисунок 10 – Окно конфигурирования

Необходимо задать параметры установки окончаний строк (рисунок 11).

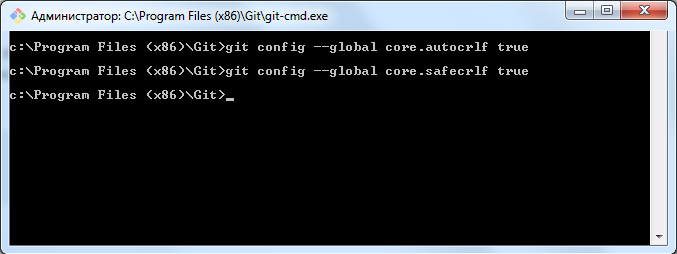


Рисунок 11 – Окно установки окончаний строк

**1. Создание проекта**

**Задание**: Создать страницу «Hello World».

**Решение**: Необходимо начать работу в пустом рабочем каталоге, например, *c:\work2*. В нем создать пустой каталог с именем *hello*, затем войти в него и создать файл с именем *hello.html*, содержание которого показано на рисунке 12.

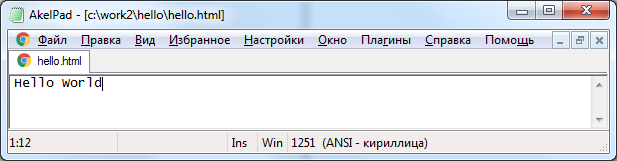


Рисунок 12 – Файл hello.html

**Задание**: Создать репозитарий.

**Решение**: для создания git-репозитария нужно перейти в каталог репозитария (рисунок 13).

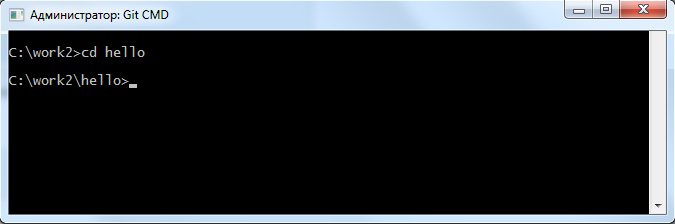


Рисунок 13 – Переход в каталог репозитария

В каталоге репозитария следует выполнить команду *git init* (рисунок 14).

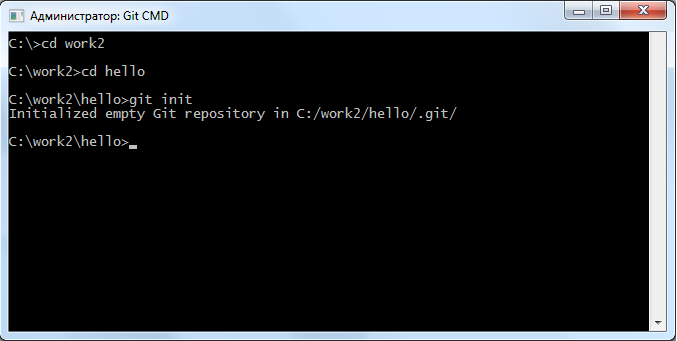


Рисунок 14 – Выполнение команды *git init*

**Задание**: Добавить страницу в репозитарий.

**Решение**: Для добавления страницы в репозитарий необходимо выполнить команду *git add* (рисунок 15).

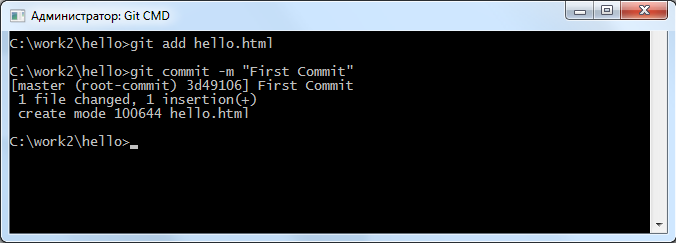


Рисунок 15 – Выполнение команды *git add*

**2. Проверка состояния репозитария**

**Задание**: Проверить состояние репозитария.

**Решение**: Для проверки состоянии репозитария следует использовать команду *git status* (рисунок 16).

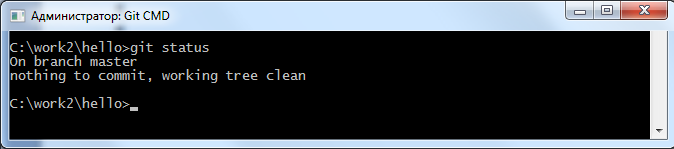


Рисунок 16 – Выполнение команды *git status*

Команда проверки сообщила (см. рисунок 16), что коммитить нечего. Это означает, что в репозитарии хранится текущее состояние рабочего каталога, и нет никаких изменений, ожидающих записи.

**3. Внесение изменений**

**Задание**: Изменить страницу «Hello World».

**Решение**: Добавим тег <h1> к приветствию. Получившаяся страница показана на рисунке 17.

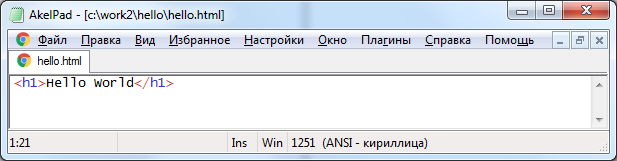


Рисунок 17 – Измененная страница

**Задание**: Проверить состояние после изменения страницы.

**Решение**: Состояние рабочего каталога следует проверить командой *git status*. Результат проверки показан на рисунке 18.

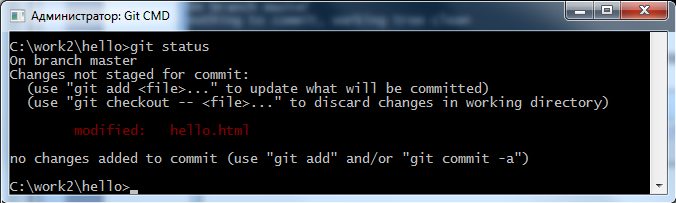


Рисунок 18 – Выполнение команды *git status* после изменения страницы

Из рисунка 18 видно, что *git* знает, что файл *hello.html* был изменен, но при этом эти изменения еще не были зафиксированы в репозитарии.

**4. Индексация изменений**

**Задание**: Необходимо проиндексировать изменения.

**Решение**: Для индексации изменений необходимо выполнить команду *git add*. Результат выполнения команды индексации показан на рисунке 19.

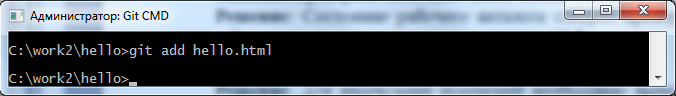


Рисунок 19 – Выполнение команды *git add*

Для того, чтобы увидеть результат индексации изменений, необходимо выполнить команду *git status* (рисунок 20).

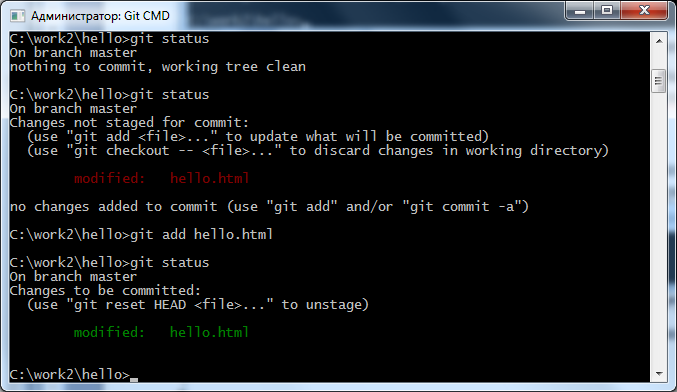


Рисунок 20 – Результат индексации изменений

Как видно из рисунка 20, изменения файла *hello.html* были проиндексированы. Это означает, что *git* теперь знает об изменении, но изменение пока не **перманентно** (т.е. навсегда) записано в репозитарий. Следующий коммит будет включать в себя проиндексированные изменения.

Если программист решил, что не хочет коммитить изменения, то команда состояния напомнит о том, что с помощью команды *git reset* можно снять индексацию этих изменений.

**5. Коммит изменений**

**Задание**: Необходимо закоммитить изменения.

**Решение**: Для коммита первоначальной версии файла *hello.html* в репозитарий использована команда *git commit* с меткой *–m*, которая делает комментарий в командной строке. Команда *commit* позволяет интерактивно редактировать комментарии для коммита. Проверим это.

Если опустить метку *–m* из командной строки, *git* перенесет пользователя в редактор по выбору. Редактор выбирается из следующего списка (в порядке приоритета):

1) переменная среды GIT\_EDITOR;

2) параметр конфигурации core.editor;

3) переменная среды VISUAL;

4) переменная среды EDITOR.

Необходимо сделать коммит по команде *git commit*. Открывшееся окно редактора показано на рисунке 21.

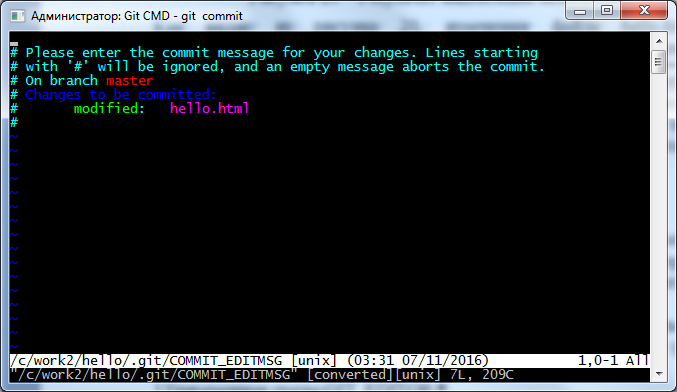


Рисунок 21 – Открывшееся окно редактора

В первой строке (см. рисунок 21) следует ввести комментарий: «Added h1 tag» (рисунок 22). После этого нужно сохранить файл и выйти из редактора. Чтобы выйти из редактора следует нажать *ESC*, затем ввести *:wq* и нажать *Enter*.

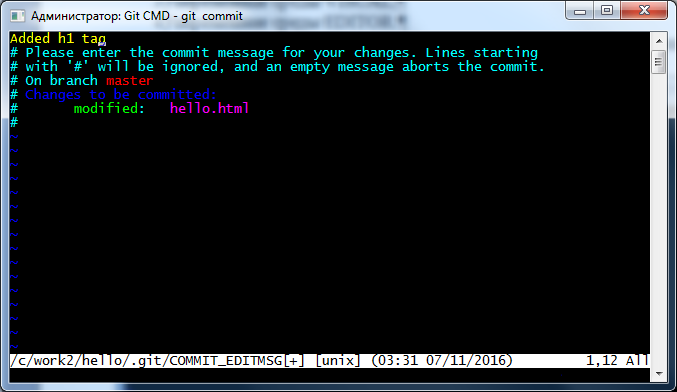


Рисунок 22 – Добавление комментария

На рисунке 23 показан результат.

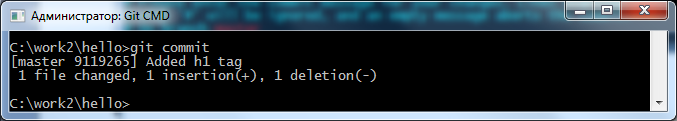


Рисунок 23 – Результат выполнения команды *git commit*

Далее командой *git status* необходимо проверить состояние (рисунок 24).

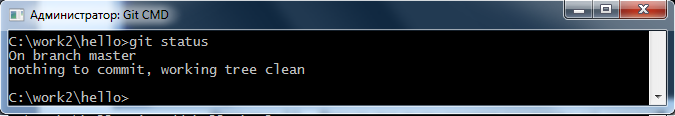


Рисунок 24 – Результат выполнения команды *git status*

Как видно из рисунка 24, рабочий каталог чистый, поэтому можно продолжать работу.

**6. Работа с изменениями, а не с файлами**

Большинство систем версионного контроля работают с файлами. Механизм таков: программист добавляет файл в версионный контроль, а система отслеживает изменения файла с этого момента.

*Git* фокусируется на **изменениях в файле**, а не на самом файле. Когда осуществляется команда *git add file*, программист не говорит *git* о добавлении файла в репозитарий. Программист говорит о том, что нужно отметить текущее состояние файла, коммит которого будет произведен позже.

**Задание**: Добавить в страницу *«Hello World»* стандартные теги *<html>* и *<body>*. Добавить это изменение.

**Решение**: Файл *hello.html* должен принять следующий вид (рисунок 25).

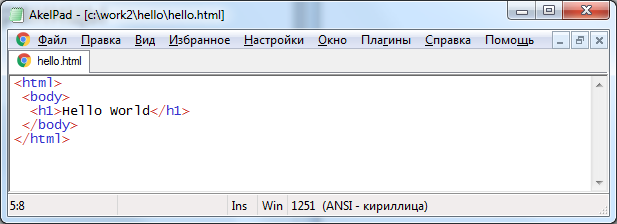


Рисунок 25 – Измененный файл страницы

Далее необходимо добавить это изменение в индекс *git* при помощи выполнения команды *git add hello.html* (рисунок 26).

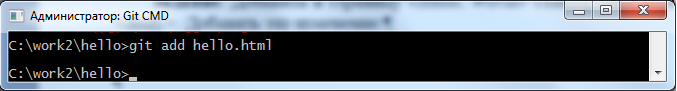


Рисунок 26 – Результат выполнения команды *git add*

**Задание**: Добавить в страницу *«Hello World»* заголовки HTML (секцию <head>). Проверить статус.

**Решение**: Файл *hello.html* должен принять следующий вид (рисунок 27).

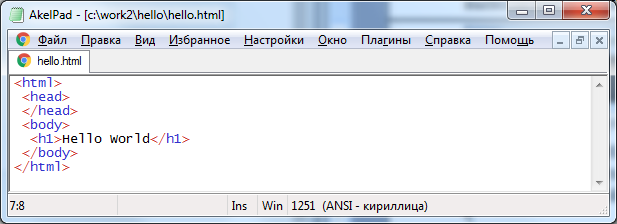


Рисунок 27 – Измененный файл страницы

Далее командой *git status* следует проверить текущий статус (рисунок 28).

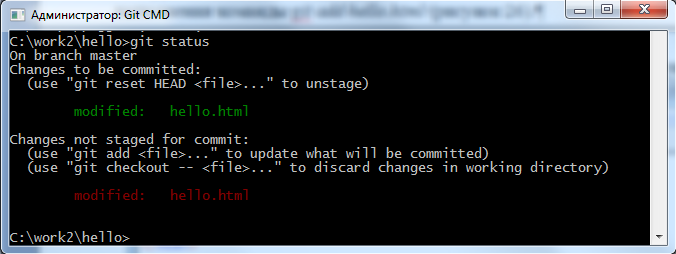


Рисунок 28 – Результат выполнения команды *git status*

Из рисунка 28 видно, что файл *hello.html* дважды указан в состоянии. Первое изменение (добавление стандартных тегов) проиндексировано и готово к коммиту. Второе изменение (добавление заголовков HTML) является непроиндексированным. Если сделать коммит сейчас, заголовки не были бы сохранены в репозитарий.

Проверим это.

**Задание**: Произвести коммит проиндексированного изменения (значение по умолчанию), а затем еще раз проверить состояние.

**Решение**: Выполним команду *git commit* как это показано на рисунке 29.

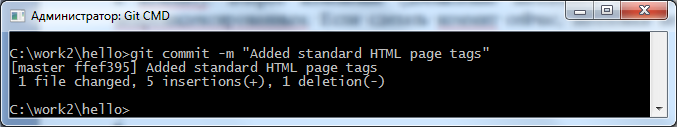


Рисунок 29 – Результат выполнения команды *git commit*

Проверим состояние при помощи команды *git status* (рисунок 30).

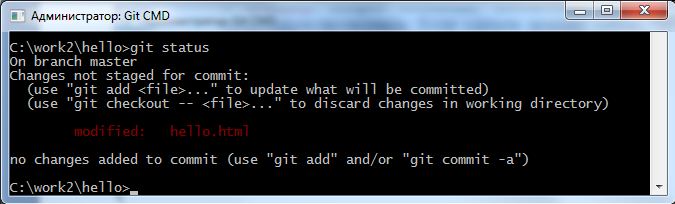


Рисунок 30 – Результат выполнения команды *git status*

Состояние команды (см. рисунок 30) говорит о том, что файл *hello.html* имеет незафиксированные значения, но уже не в буферной зоне.

**Задание**: Добавить второе изменение и сделать его коммит.

**Решение**: Добавим второе изменение в индекс, а затем проверим состояние с помощью команды *git status* (рисунок 31).

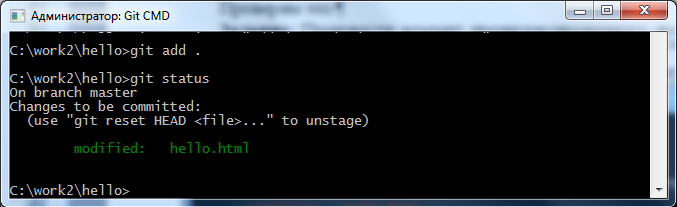


Рисунок 31 – Результат выполнения команд

**Примечание**: В качестве файла для добавления использован текущий каталог («.»). Это самый краткий и удобный путь для добавления всех изменений в файлы текущего каталога и его подкаталоги. Но поскольку он добавляет всё, не лишним будет проверить состояние перед запуском *add*, просто чтобы убедиться, что не добавлен какой-то файл, который добавлять не нужно.

Как видно из рисунка 31, второе изменение было проиндексировано и готово к коммиту.

Далее нужно сделать коммит второго изменения (рисунок 32).

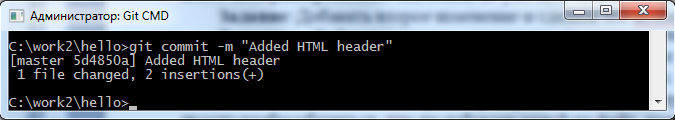


Рисунок 32 – Результат выполнения коммита

**7. Просмотр истории проекта**

**Задание**: просмотреть историю проекта.

**Решение**: с помощью команды *git log* можно просмотреть историю проекта (рисунок 33).

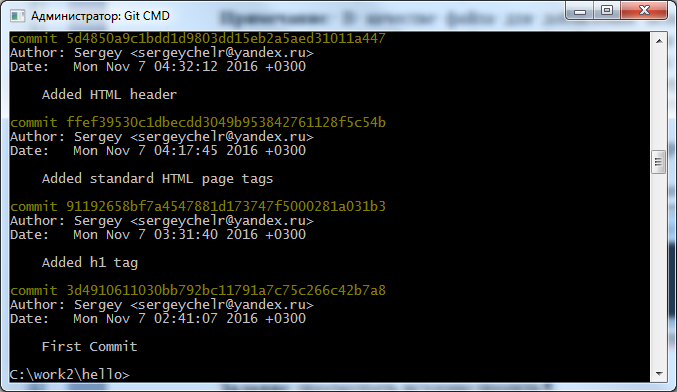


Рисунок 33 – Результат выполнения команды *git log*

На рисунке 33 выведен список всех четырех коммитов в репозитарий, которые были совершены в предыдущих заданиях.

**8. Использование алиасов**

**Задание**: использовать общие алиасы.

**Решение**: выполнить следующие команды (рисунок 34).

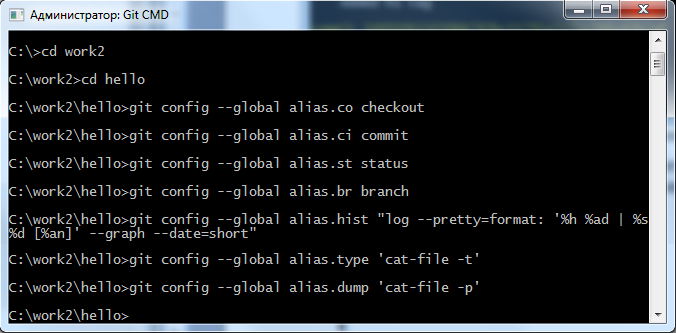


Рисунок 34 – Задание общих алиасов

Из рисунка 34 следует, что теперь вместо команды *git status* можно вводить *git st*, вместо *git checkout* вводить *git co* и т.д.

Выполним команду git status при помощи алиаса (рисунок 35).

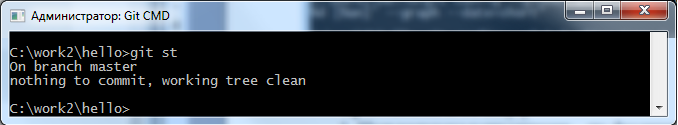


Рисунок 35 – Выполнение команды *git status* при помощи алиаса

**9. Получение старых версий**

**Задание**: Получить хэши предыдущих версий.

**Решение**: Выполнить команду *git log* (рисунок 36).

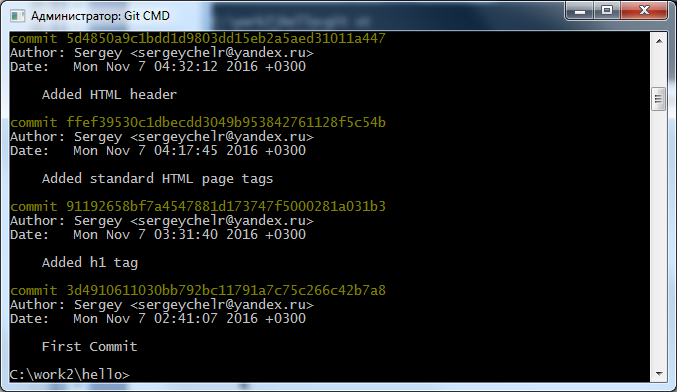


Рисунок 36 – Результат выполнения команды *git log*

На рисунке 36 после каждого коммита показан хэш. Этот хэш-код (достаточно первых 7 знаков) может быть использован в последующих командах.

Следует выполнить команду, показанную на рисунке 37.

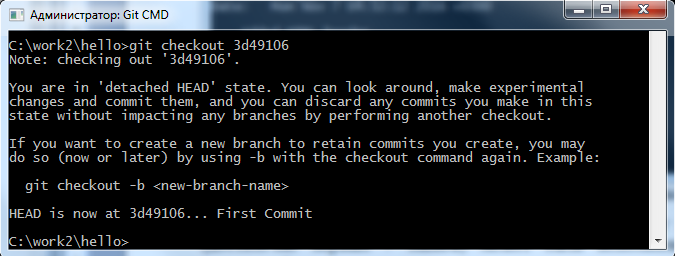


Рисунок 37 – Результат выполнения команды *git checkout 3d49106*

**Задание**: Вернуться к последней версии в ветке *master*.

**Решение**: Выполнить команды, показанные на рисунке 38.

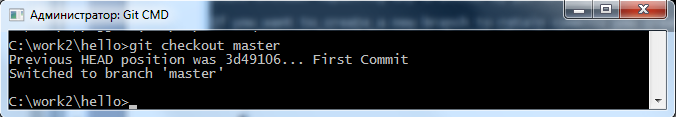


Рисунок 38 – Результат выполнения команды *git checkout master*

«Master» - это имя ветки по умолчанию. Переключая имена веток, можно попасть на последнюю версию выбранной ветки.

**10. Создание тегов версий**

**Задание**: Создать тег текущей версии страницы (дав ей название *v1*).

**Решение**: Выполнить команду *git tag v1* (рисунок 39).

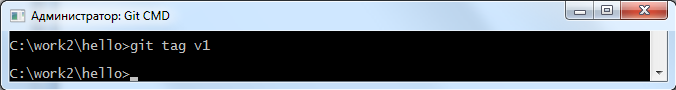


Рисунок 39 – Результат выполнения команды *git tag v1*

Теперь текущая версия страницы называется *v1*.

**Задание**: Создать теги для предыдущих версий.

**Решение**: Создадим тег для версии, которая идет перед текущей версией и назовем его *v1-beta*. Для этого в первую очередь нужно переключиться на предыдущую версию. Вместо поиска до хэш, можно использовать ~, означающее «родитель *v1*». Необходимо выполнить команду *git checkout v1~1* (рисунок 40).

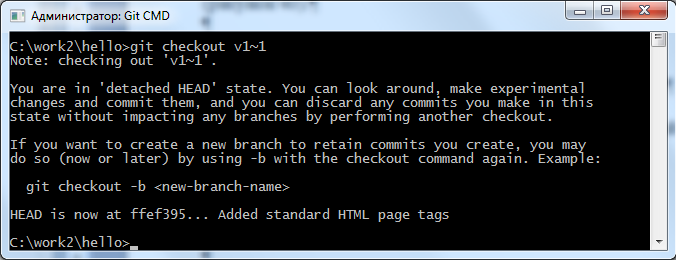


Рисунок 40 – Результат выполнения команды *git checkout v1~1*

Обозначение «*v1~1*» означает как «первая версия предшествующая *v1*».

Эту версию сделаем версией *v1-beta* (рисунок 41).

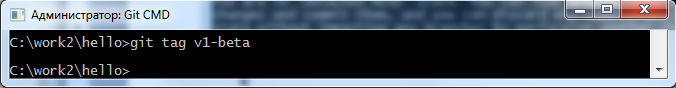


Рисунок 41 – Результат выполнения команды *git tag v1-beta*

**Задание**: Осуществить переключение между двумя отмеченными версиями по имени тега.

**Решение**: Выполнить команды (рисунок 42).

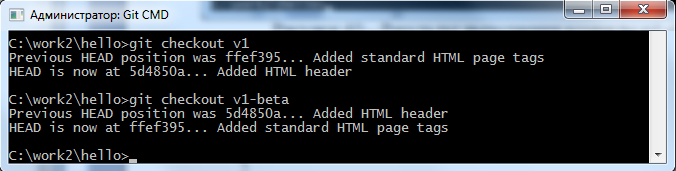


Рисунок 42 – Результат выполнения команд *git checkout v1* и *git checkout v1-beta*

**Задание**: Осуществить просмотр тегов с помощью команды *tag*.

**Решение**: Выполнить команду *git tag* (рисунок 43).

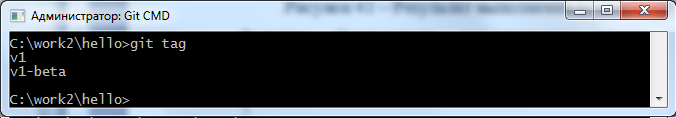


Рисунок 43 – Результат выполнения команды *git tag*

**11 Отмена коммитов**

**Задание**: Отменить коммит в локальный репозиторий.

**Решение**: Коммит будет отменен путем создания нового коммита, отменяющего нежелательные изменения.

Следует изменить файл *hello.h* на файл, показанный на рисунке 44.

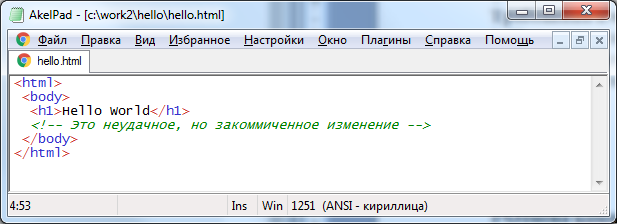


Рисунок 44 – Измененный файл *hello.html*

Следует выполнить команду *git add hello*, показанную на рисунке 45.

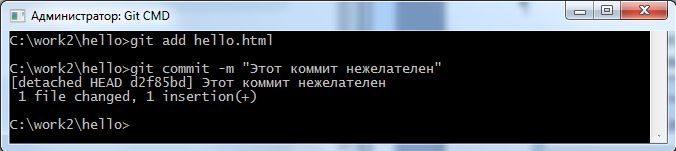


Рисунок 45 – Добавление внепланового коммита

Чтобы отменить коммит, требуется сделать коммит, который удаляет изменения, сохраненные нежелательным коммитом. Следует выполнить команду:

*git revert HEAD*

После чего откроется окно редактора, показанное на рисунке 46.

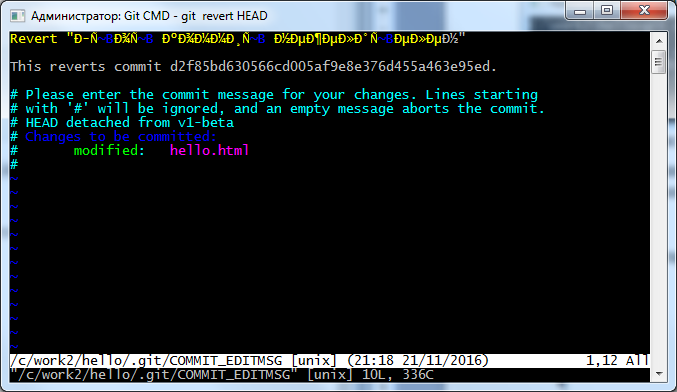


Рисунок 46 – Добавление внепланового коммита

Так как сообщение на русском языке не распознается редактором, следует заменить сообщение с использованием латинских символов (рисунок 47).

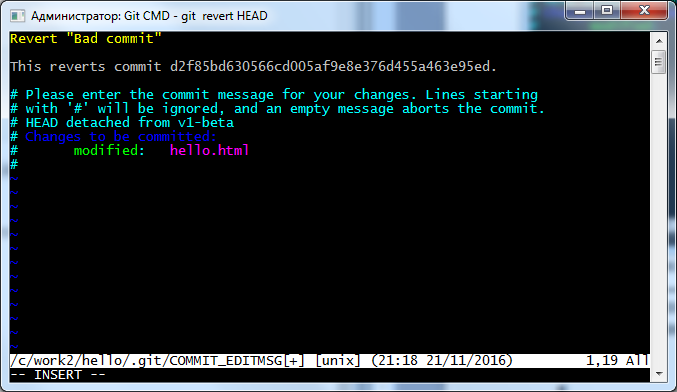


Рисунок 47 – Редактирование комментария в редакторе

Необходимо сохранить изменения в редакторе и выйти из него. Результат отмены коммита показан на рисунке 48.

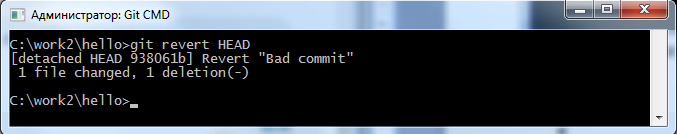


Рисунок 48 – Результат отмены коммита

При помощи проверки лога (команда *git log*) (рисунок 49) можно показать нежелательные и отмененные коммиты в репозиторий.

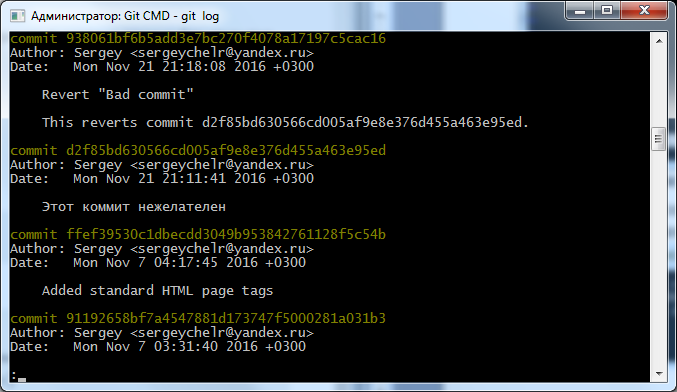


Рисунок 49 – Результат проверки лога (лог показан не до конца)

**12 Удаление коммитов**

**Задание**: Удалить последний коммит из ветки.

**Решение**: Рассмотренная команда *revert* позволяет отменить любые коммиты в репозиторий. Однако, и оригинальный и отмененный коммиты видны в истории ветки (при использовании команды *git log*).

Команда *reset* при получении ссылки на коммит выполнит следующие действия:

1. Перепишет текущую ветку, чтобы она указывала на нужный коммит.

2. Опционально сбросит буферную зону для соответствия с указанным коммитом.

3. Опционально сбросит рабочий каталог для соответствия с указанным коммитом.

На рисунке 6 виден коммит с комментарием “Этот коммит нежелателен” и коммит с комментарием Revert “Bad commit”. Требуется удалить их при помощи команды сброса.

Следует отметить последний коммит тегом, чтобы его можно было найти (рисунок 50).

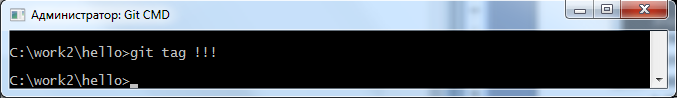


Рисунок 50 – Последний коммит отмечен тегом “!!!”

Пусть требуется сбросить ветку с коммитами до точки с тегом «v1» (она предшествовала ошибочному коммиту и была задана в работе выше). Для этого требуется выполнить команду *reset* с параметром *hard* (рисунок 51).

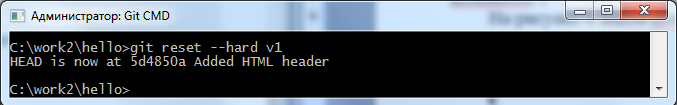


Рисунок 51 – Последний коммит отмечен тегом “!!!”

Параметр *--hard* указывает, что рабочий каталог должен быть обновлен в соответствии с новым *head* ветки.

Теперь следует выполнить команду *git log* (рисунок 52).

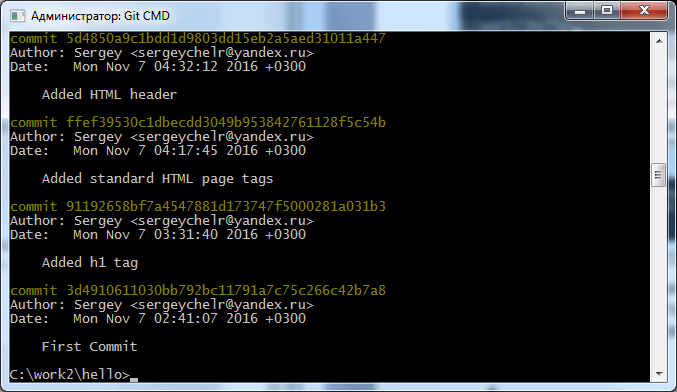
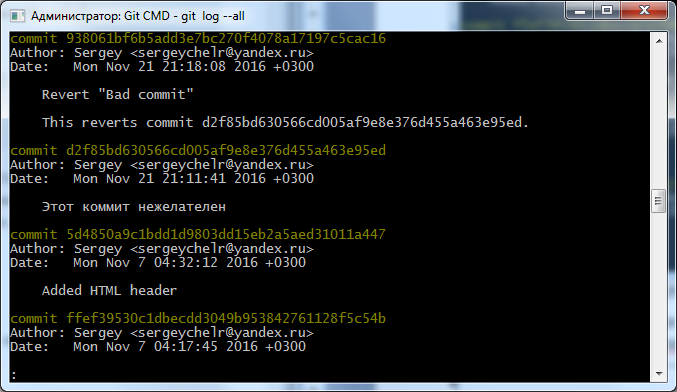


Рисунок 52 – Результат выполнения команды *git log*

Как видно из рисунка 52, последних коммитов в ветке нет. Однако, все коммиты находятся в репозитории. На эти коммиты можно ссылаться. Следует выполнить команду *git log --all* (рисунок 53).



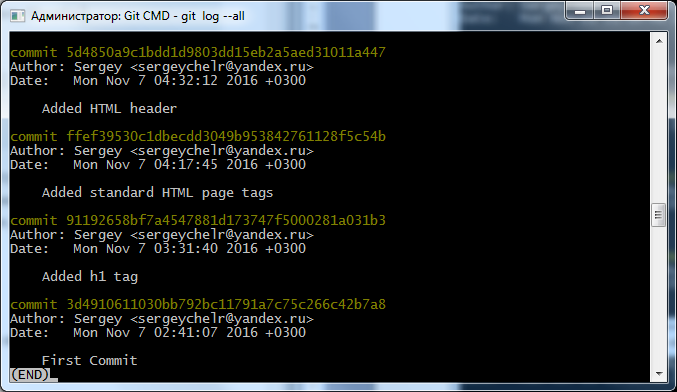


Рисунок 53 – Результат выполнения команды *git log --all*

Как видно из рисунка 53, ошибочные коммиты не исчезли. Они находятся в репозитории, но сослаться на них теперь можно только по хэшу. Коммиты, на которые нет ссылок, остаются в репозитории до тех пор, пока не будет запущен сборщик мусора.

Сброс в локальных ветках, как правило, безопасен. Последствия любой «аварии» как правило, можно восстановить сбросом с помощью нужного коммита. Однако, если ветка «расшарена» на удаленных репозиториях, сброс может сбить с толку других пользователей ветки.

**13 Удаление тега “!!!” и коммитов, на которые он ссылался**

**Задание:** удалить тег “!!!”.

**Решение:** Следует удалить тег “!!!” и его коммиты сборщиком мусора, для чего выполним команды, показанные на рисунке 54.

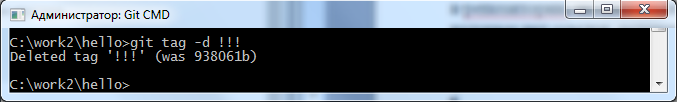
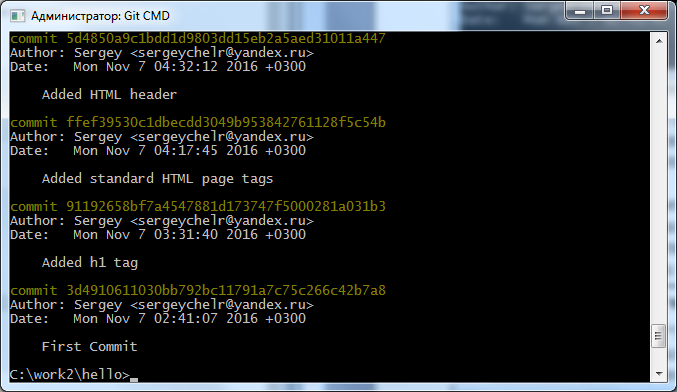


Рисунок 54 – Результат выполнения команды *git tag -d !!!*

После этого выполним команду *git log --all* (рисунок 55).



Теперь тег «!!!» не отображается в репозитории.

**Требования к отчету**

1. Имя каталога репозитария – по фамилии студента.

2. Проделать все действия.

3. Отчет должен содержать скрин-шоты всех действий и пояснения к ним.

4. Вместо hello.html использовать файл со своим проектом.

**Контрольные вопросы**

1. Понятие СКВ.

2. Классификация СКВ.

3. Подход Git к хранению данных.

4. Командная строка Git.

5. Создание репозитория в существующей директории.

6. Клонирование существующего репозитория.

7. Состояния файлов в Git.

8. Отслеживание новых файлов.

9. Индексация изменённых файлов.

10. Сокращенный вывод статуса.

11. Игнорирование файлов.

12. Просмотр индексированных и неиндексированных изменений.

13. Фиксация изменений.

14. Игнорирование индексации.

15. Удаление файлов.

16. Понятие ветки в *Git.*

17. Создание новой ветки.

18. Переключение веток.

Успехов!