**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая Кибернетика и Информационные технологии»

Лабораторная работа №1

**Основы работы с системой контроля версий**

Выполнил: Студент группы

БВТ2203

Иляков Станислав

Москва

2023

**Цель работы:** изучение базовых функций и подходов к командной разработке в системе контроля версий Git

**Задание:** определить проект (состоящий из нескольких файлов), который будет добавлен под версионный контроль, и произвести ряд операций в СКВ, включающих:

1. Создание репозитория или клонирование существующего
2. Внесение ряда изменений в файл(ы) и выполнение коммитов (кол-во не меньше 2)
3. Откат изменений до одного из предыдущих коммитов двумя разными способами
4. Создание, переименование и удаление ветки
5. Слияние ветки с веткой мастер с решением конфликта содержания файлов
6. Настройка игнорирования git’ом файлов определенного расширения с использованием glob-шаблонов
7. Вывод лога коммитов в измененном пользовательском формате
8. Настройка удалённого репозитория: получение и отправка изменений в репозиторий
9. Организация совместной работы над проектом: клонирование чужого репозитория, выполнение коммитов, отправка изменений

**Ход работы**:

Coздаём проект и инициализируем новый репозиторий (git init) добавляем файлы (git add .) в данном файле используется уже файл .gitignore поэтому добавятся только 3 файла (.gitignore, lab.py, pyvenv.cfg). Проверяем всё ли верно при помощи команды ‘git status’

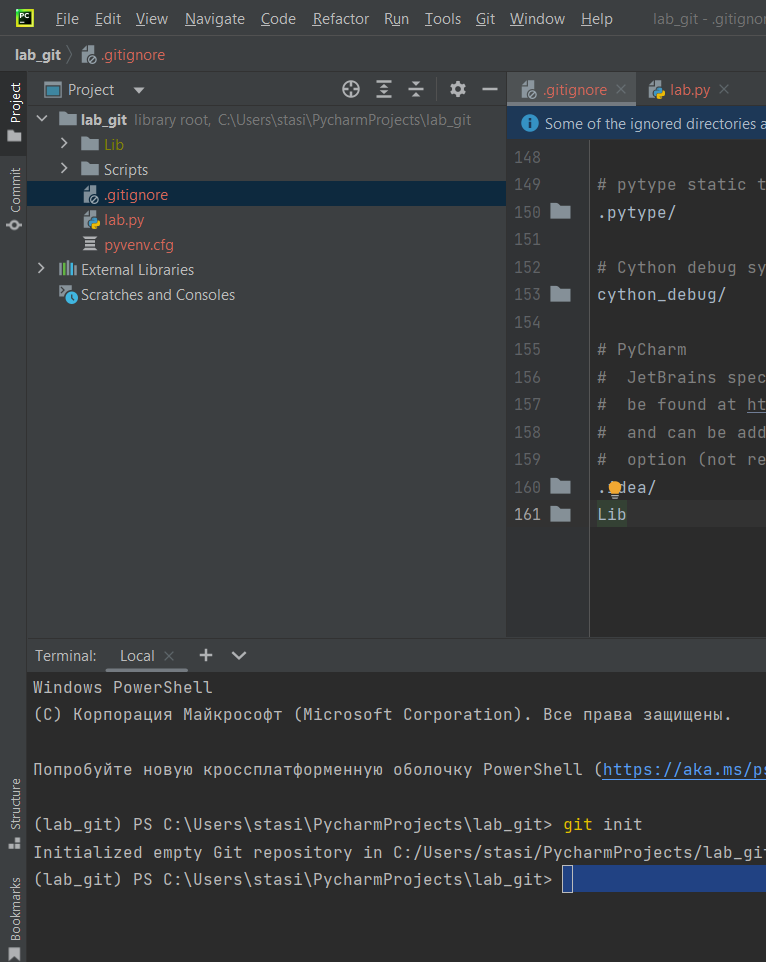


Рис.1 – инициализация репозитория

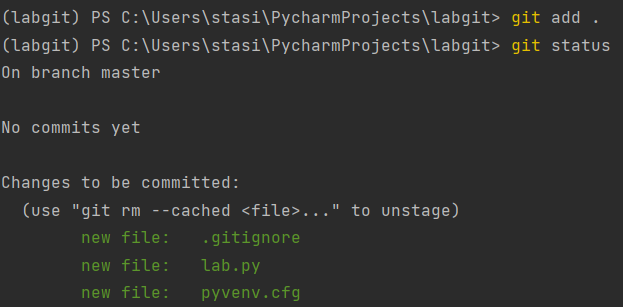


Рис.2 – добавление и проверка файлов

Делаем первый коммит (git commit –m <.название коммита.>)

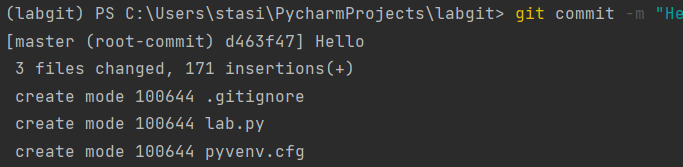


Рис.3 – добавление коммита

Далее делаем изменения в файле добавляем их и коммитим.

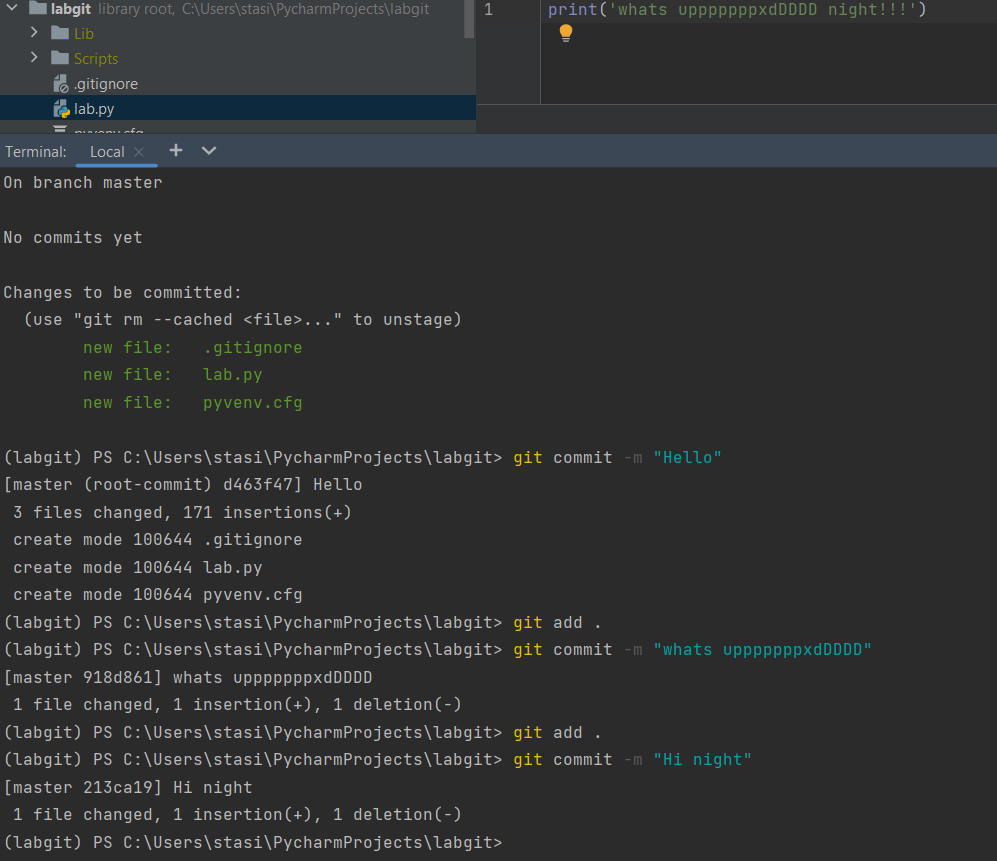


Рис.4 – добавление двух изменений и их коммиты

Далее чтобы точно проверить, что всё верно выведем список коммитов(git log).

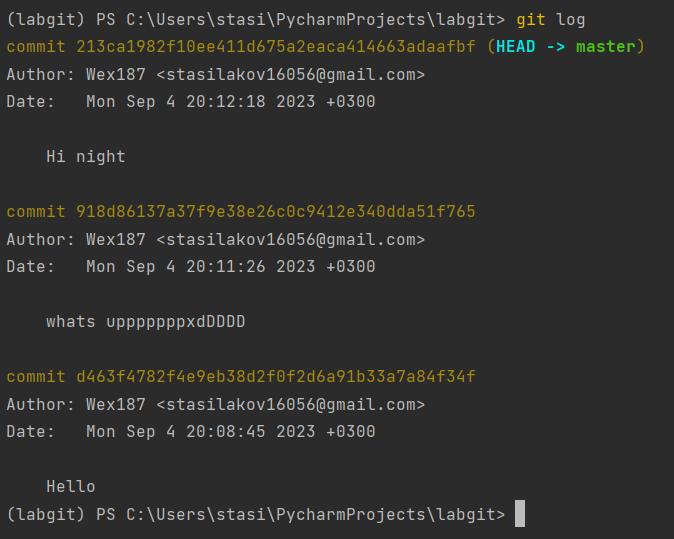


Рис.5 – список коммитов

Далее откатимся на один коммит при помощи команды ‘git reset –hard HEAD~1’, которая удалит предыдущий коммит.

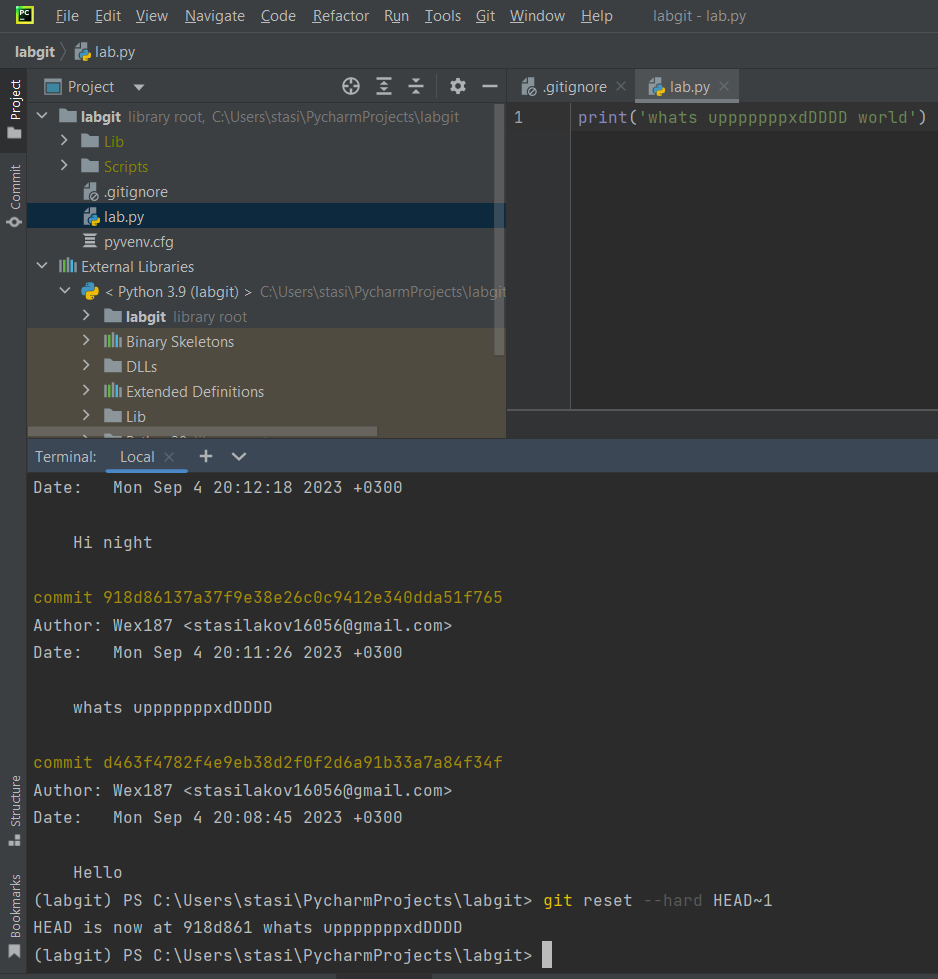


Рис.6 – откат с помощью ‘git reset –hard HEAD~1’

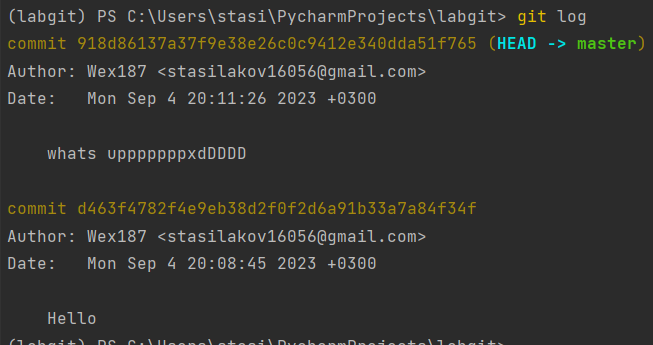


Рис.7 – последний коммит удалился

Чтобы откатить другим способом используем команду 'git revert HEAD –no-edit', которая откатит последние коммит посредством добавления нового коммита, который будет являться репликой выбранного нами коммита и проверяем это.

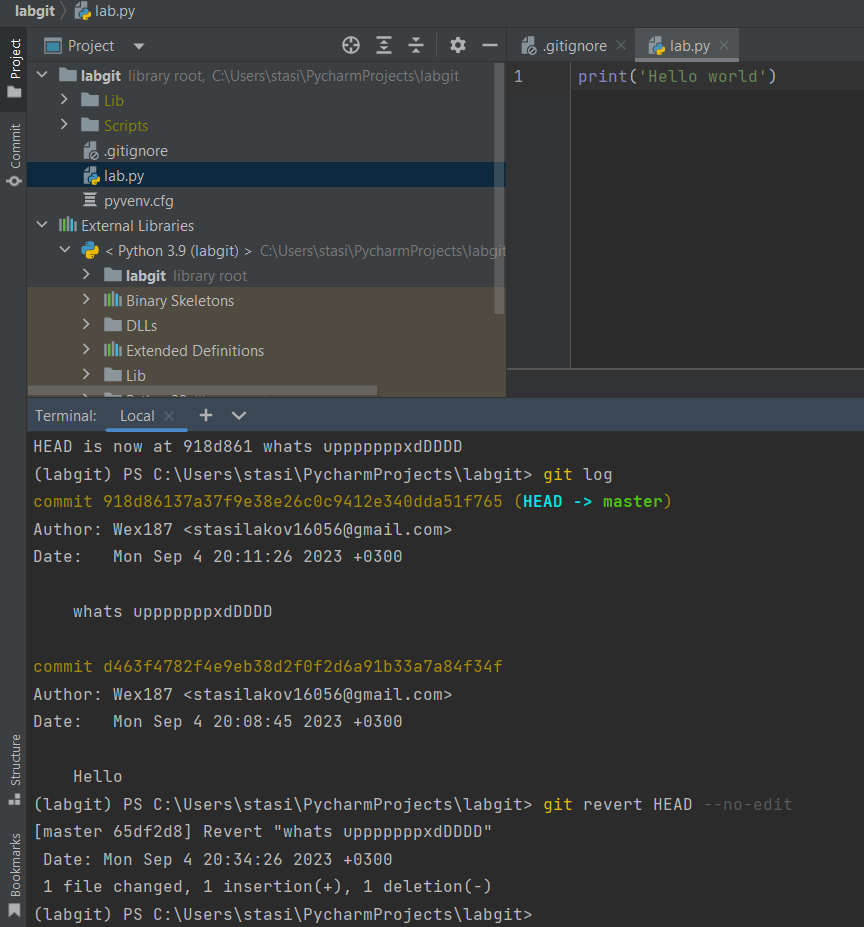


Рис.8 – откат с помощью 'git revert HEAD –no-edit'

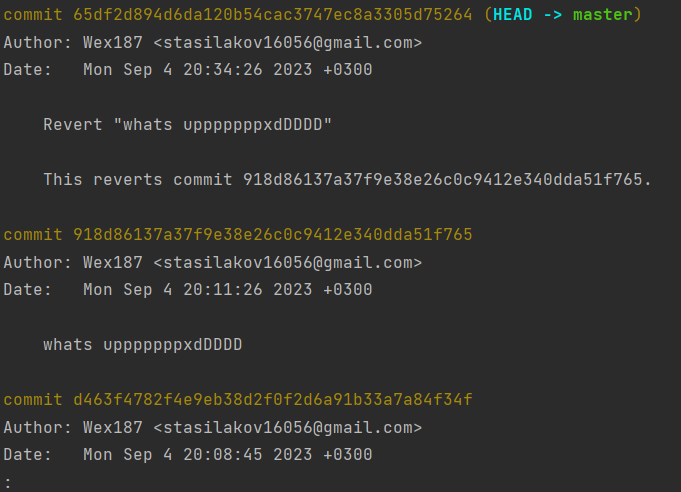


Рис.9 – реплика коммита

Создаём новую ветку и называем Mortal\_Combat (git branch Mortal\_Combat). Переходим на неё (git checkout Mortal\_Combat) и переименовываем её (git branch –m MC), а делее проверяем всё ли сделано верно (git branch). Чтобы удалить ветку возвращаемся на ветку ‘master’ и вводим команду ‘git branch –d MC’ и проверяем всё ли верно.

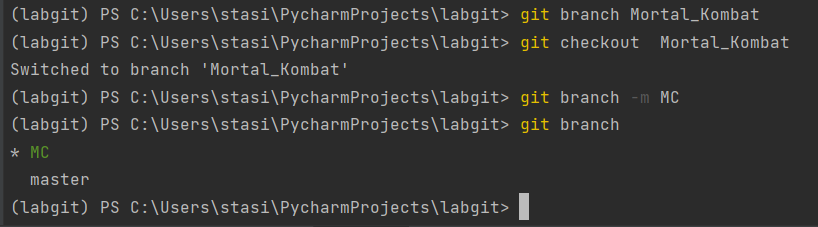


Рис.10 – добавление, переход и изменение имени ветки

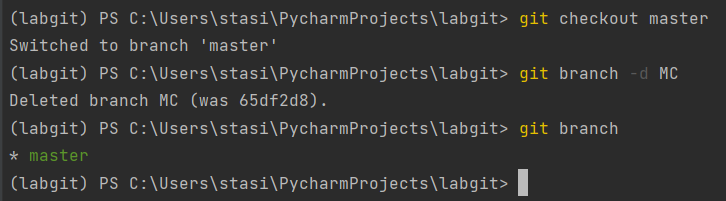


Рис.11 – удаление ветки

Далее повторяем все предыдущие шаги тк. Нам нужно слить две ветки при помощи команды ‘git merge MC’, где у нас возникает конфликт, который мы можем решить посредством изменения файла их слияния, а далее добавляя и коммитя, то что мы решили оставить.

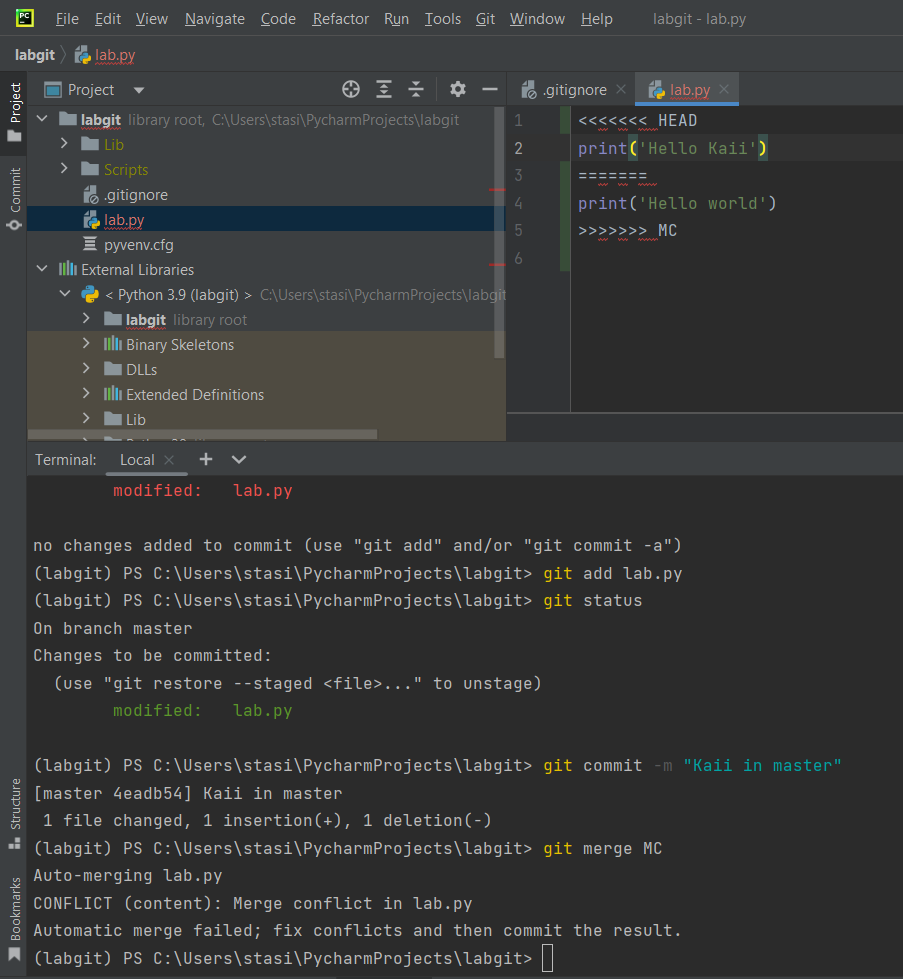


Рис.12 – конфликт слияния веток

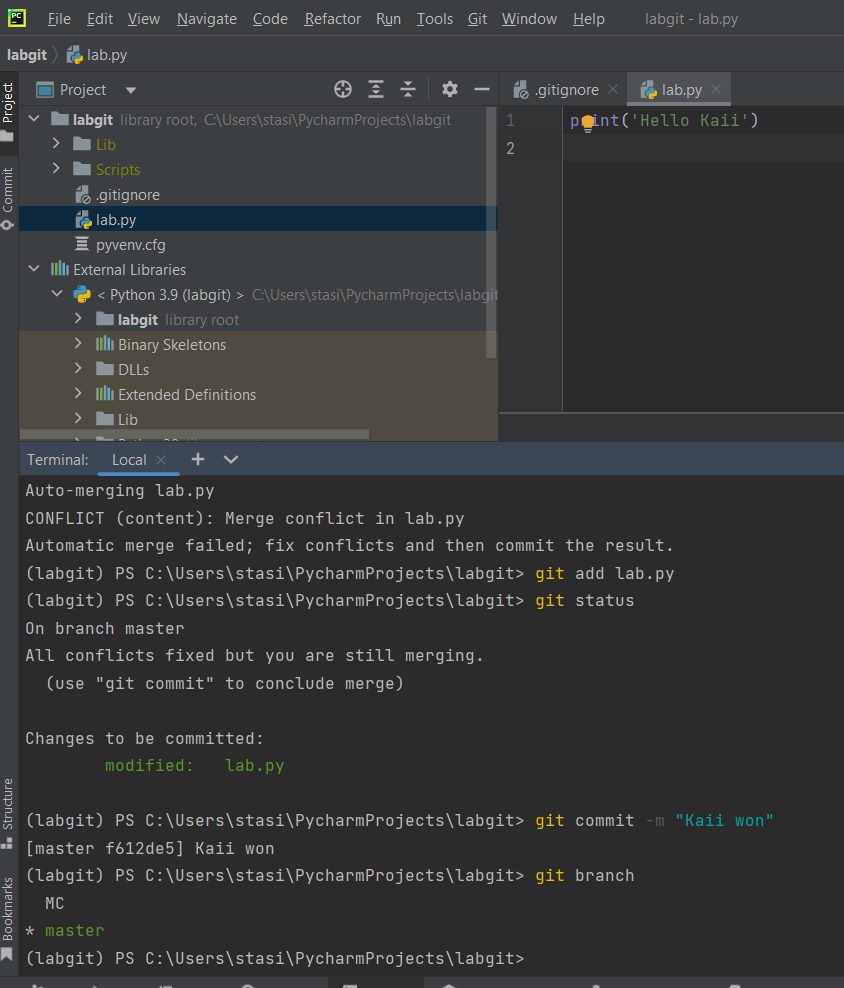


Рис.13 – решение конфликта при помощи выборочного редактирования или выбора одного из вариантов

Далее выводим логи коммитов в пользовательском формате(git log –prety=format:”%h - %n, %ad:%s”)

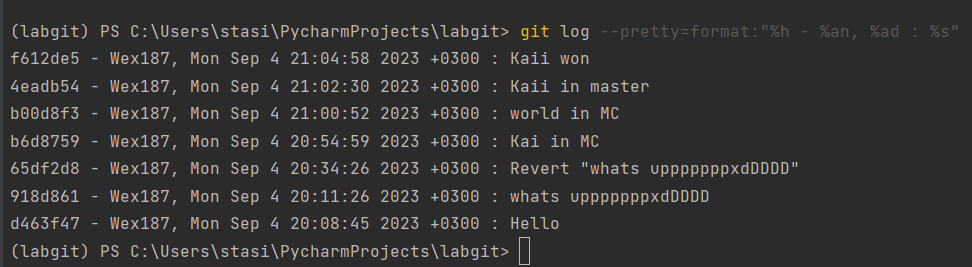


Рис.14 – пользовательский формат логов

Cоздаём удалённый репозиторий на платформе GitHub, связываем удалённый репозиторий с локальным (git remote add origin <url\_удалённого репозитория>) и отправляем изменения из локального репозитория в удалённый (git push origin master).

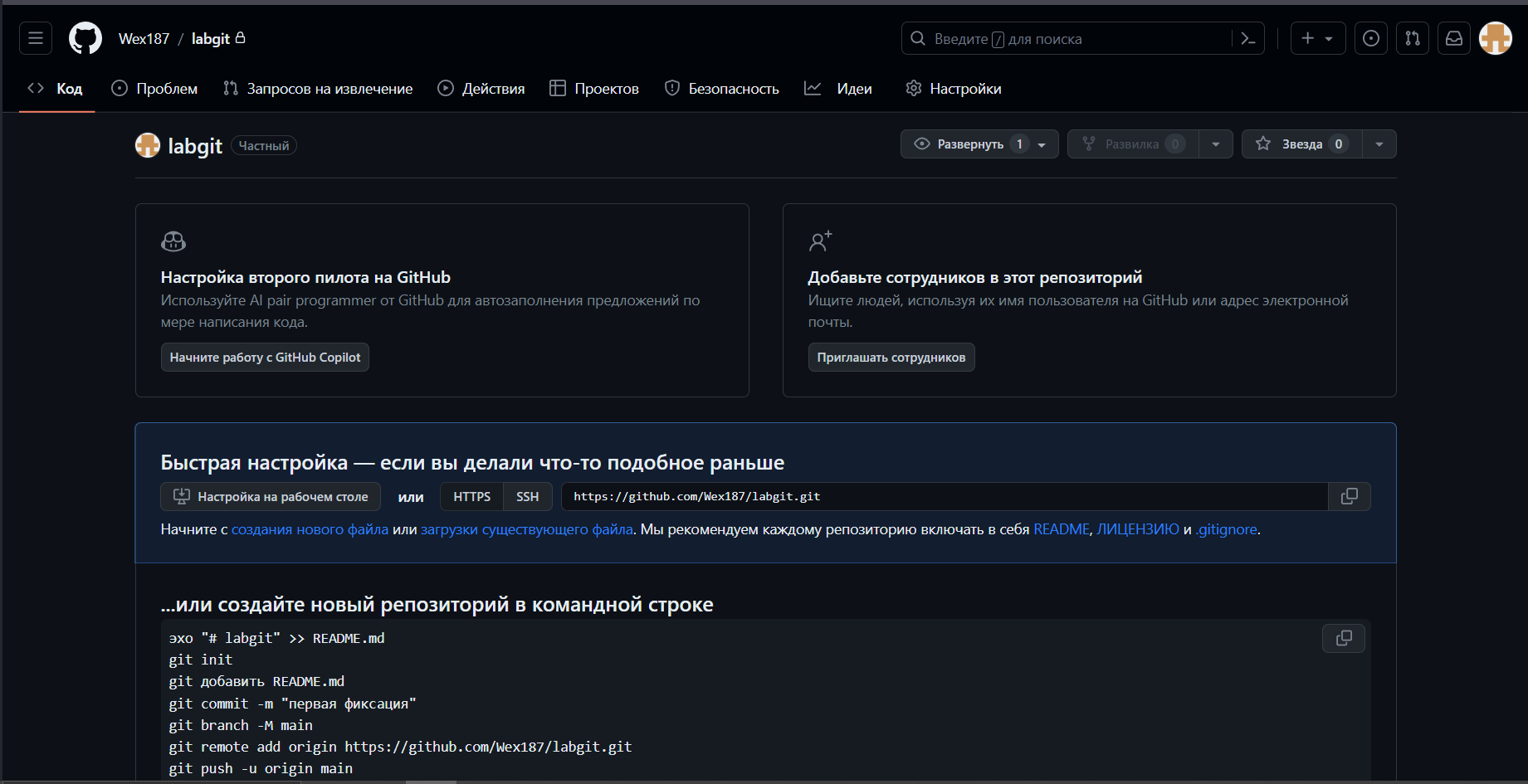


Рис.15 – репозиторий на GitHub

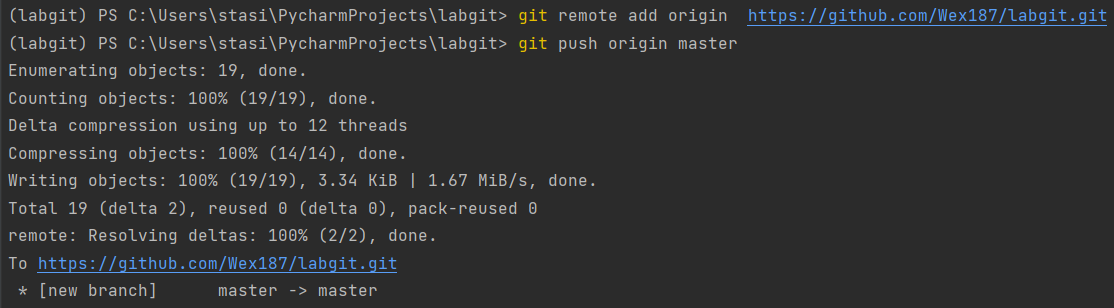


Рис.15 – push на удалённый репозиторий

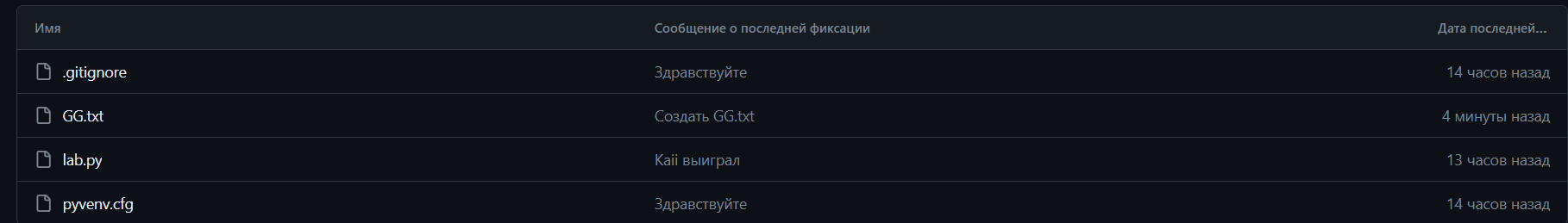
Заходим в GitHub и проверяем появились ли изменения. Далее создаём в удалённом репозитории новый файл и пытаемся получить его в локальный (git pull origin master) 

Рис.16 – Создание текстового файла ‘GG’ в удалённом репозитории

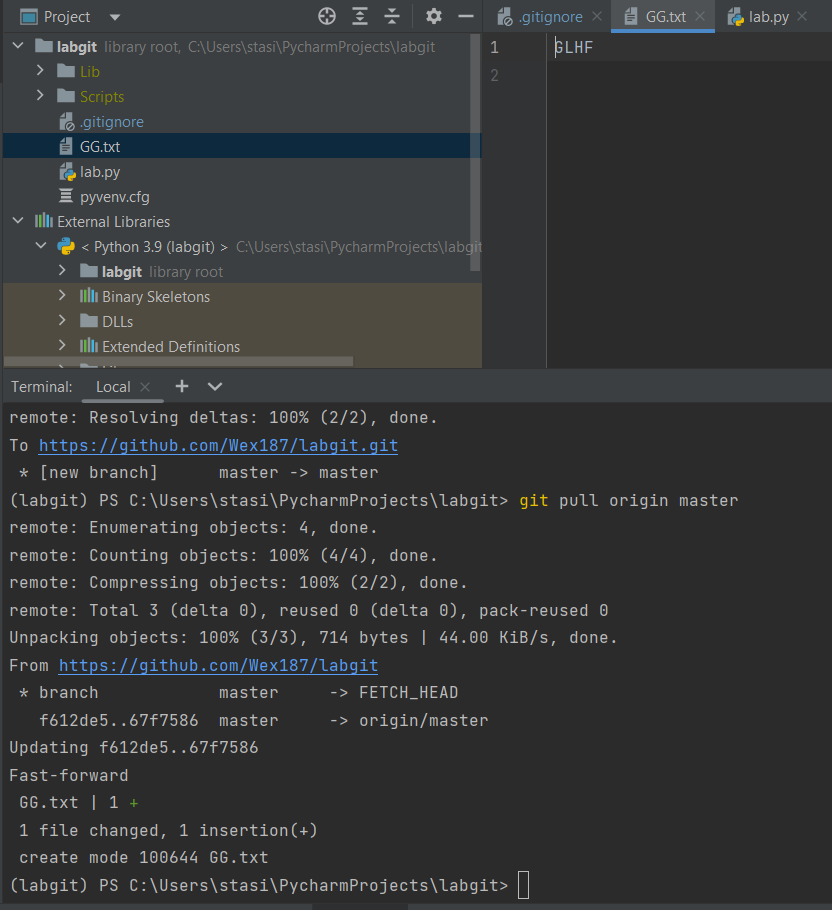
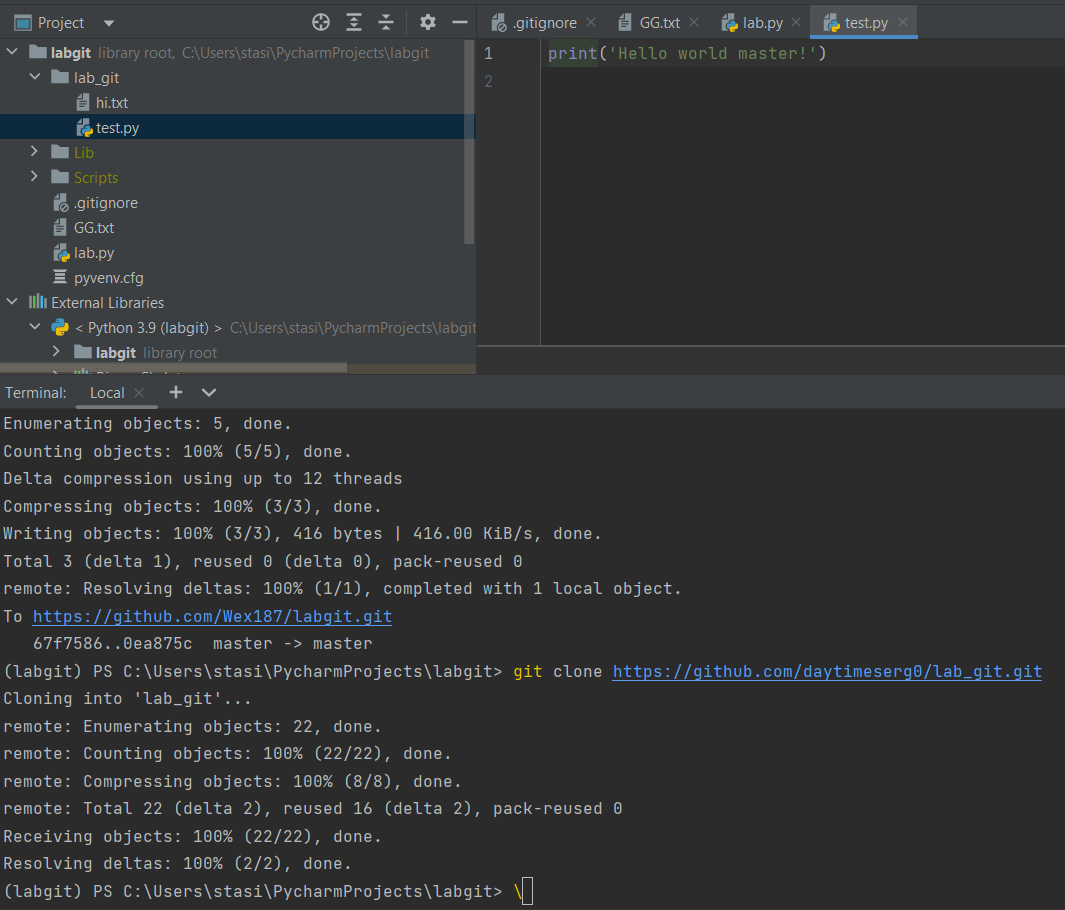


Рис.15 – получение данных из удалённого репозитория

Так как следующие задание требует совместной работы мы с одногруппником обменялись предоставили друг другу доступ к проектам.

Копируем ссылку на его репозиторий и клонируем его с помощью команды 'git clone <url\_ссылка друга>' в новую директорию, вносим изменения в его файл ‘test.py’ добавляем изменения, коммитим и пушим в его репозиторий.

 Рис.16 – клонируем репозиторий друга в проект

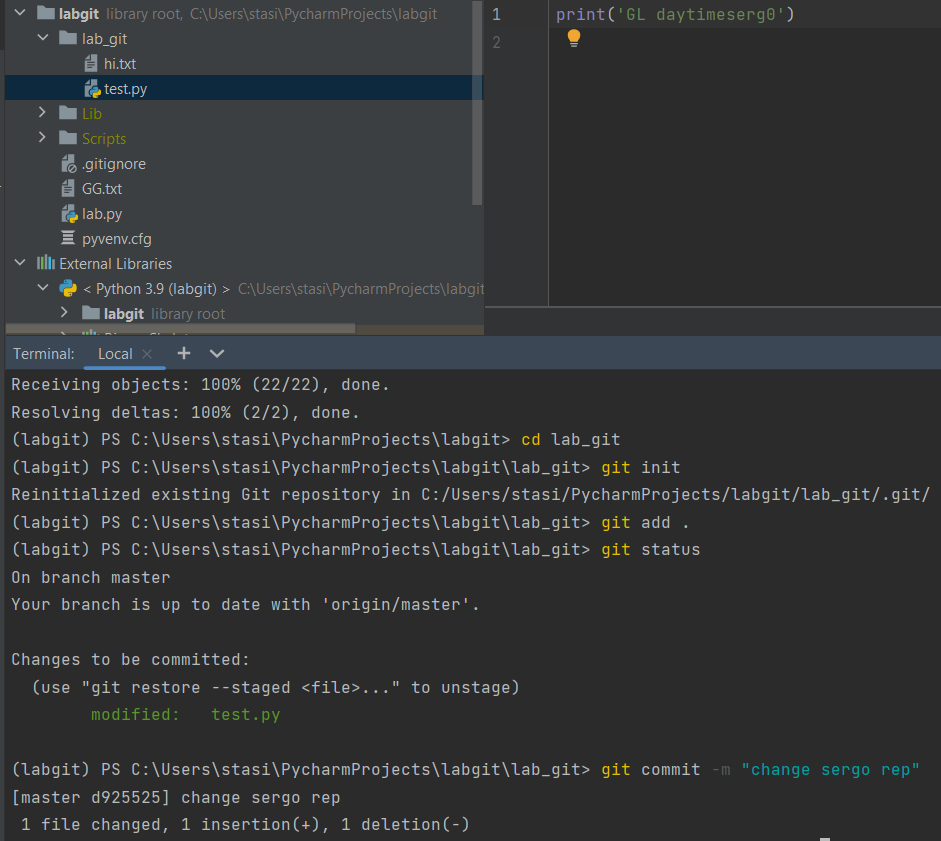


Рис.17 – делаем изменения, добавляем их и коммитим

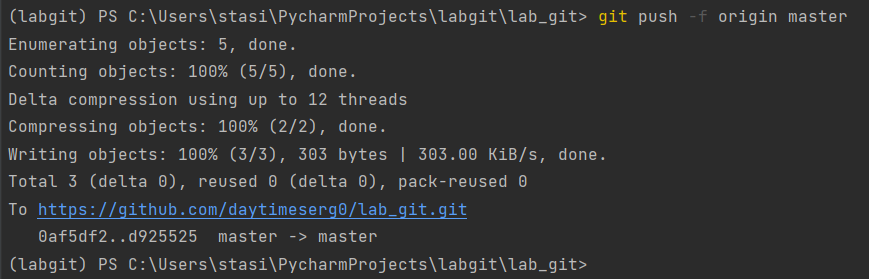


Рис.18 – пушим изменения в репозиторий друга

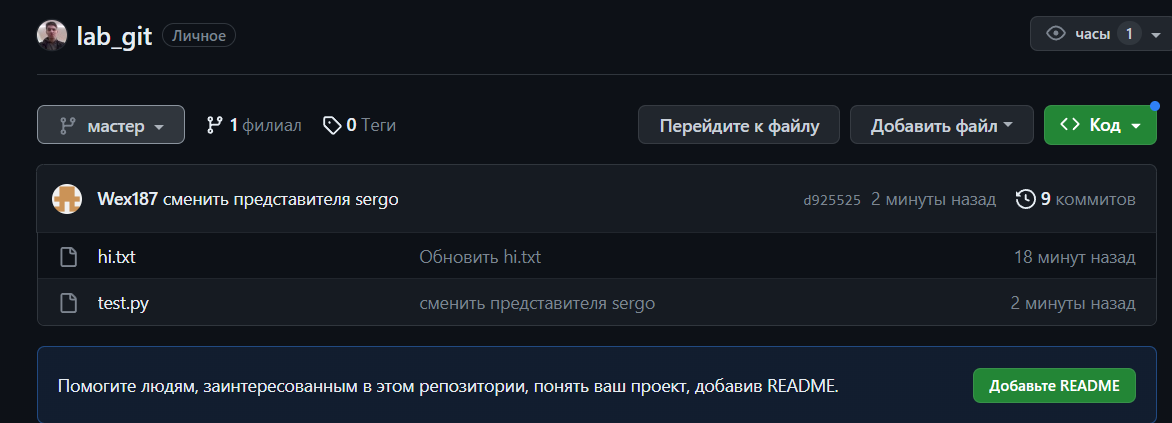


Рис.19 – проверяем изменения

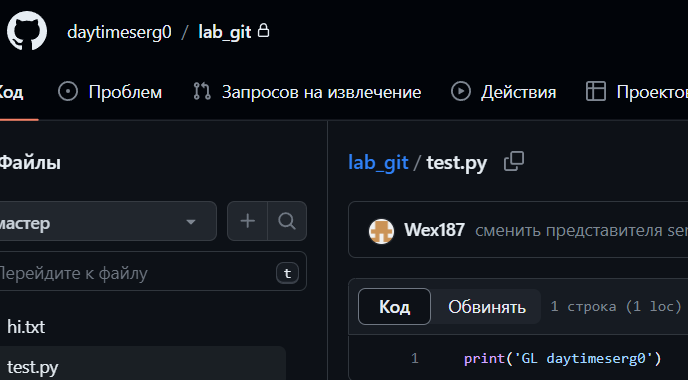


Рис.20 – убеждаемся в том, что всё сделано верно

**Вывод:** я изучил базовые функции и подходы к командной разработке в системе контроля версий Git

Ссылка на Git - <https://github.com/Wex187/-.git>

**Источники:**

1. <https://github.com/ValeryLVS/Git>
2. <https://www.atlassian.com/ru/git/tutorials/using-branches/merge-conflicts>
3. <https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/resolving-a-merge-conflict-using-the-command-line>