**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное

учреждение высшего образования

**Национальный исследовательский**

**Томский политехнический университет**

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

Отчет по лабораторной работе №5 по дисциплине

**«Управление разработкой промышленного программного обеспечения»**

Автопроверка PEP-8 c GitHub Actions

Выполнил:

Студент группы 8ПМ4Л \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.А. Валетов

Проверил:

Старший преподаватель ОИТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Г.Небаба

Томск 2025

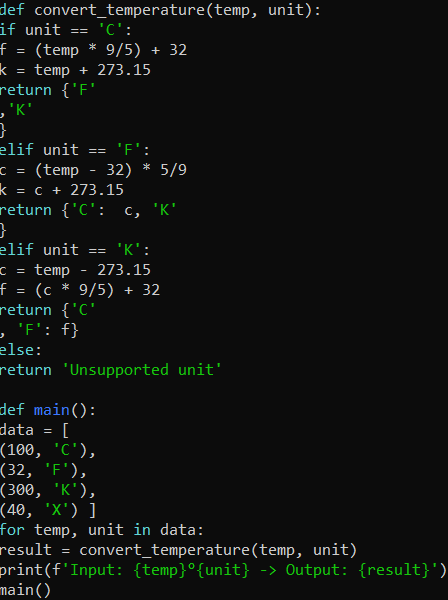
**Цель**: научиться делать автопроверку согласно указанному соглашению с использованием GitHub Actions.

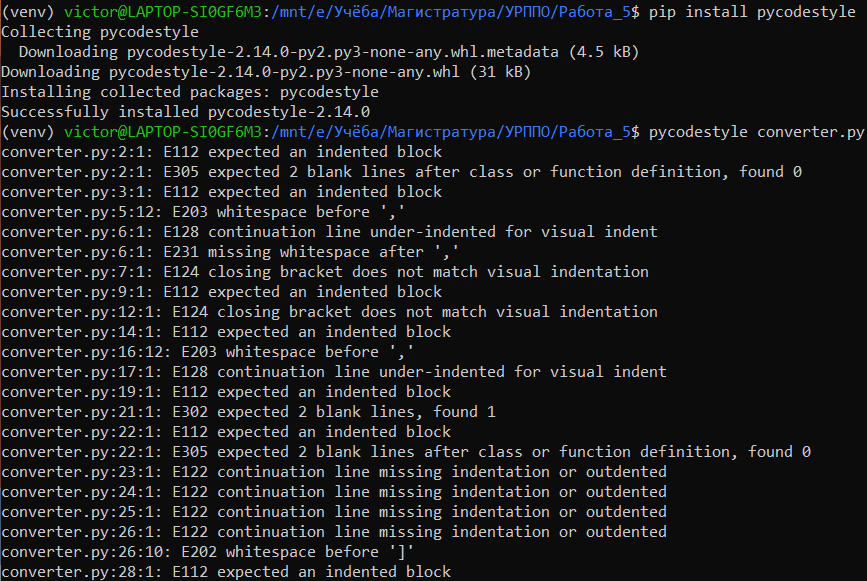
**Задания**:

1. Реализовать код на Python, решающий небольшую задачку (например, парсер или конвертер). Минимум 30 строк кода.  
2. Установить инструмент pycodestyle и провести проверку с помощью данного инструмента вручную (через CLI). Какие [ошибки](https://pycodestyle.pycqa.org/en/latest/intro.html#error-codes) были обнаружены? Ошибки исправлять не нужно.  
3. Сгенерировать с помощью pip freeze файл *requirements.txt*4. Подключить GitHub Actions к проекту в GitHub. В файл *.github/workflows/check.yml*прописать следующие сценарии:

    a. Установка необходимых файлов из requierements.txt  
    b. Предварительная установка pycodestyle в конфигурационном файле  
    c. Реализовать внутренний скрипт для проверки на наличие ошибок. Если ошибки есть - скрипт завершается с ошибкой. Подсказка для продвинутых: некоторые ошибки можно исключать при должном обосновании с помощью передачи дополнительного специального  параметра.  
5.  Загрузить написанный python-скрипт. Внимание: для первого push'а с  .py-файлом важно, чтобы была хотя бы одна ошибка. Workflow check должен завершиться со статусом failure (это также должно быть отображено в GitHub, в истории коммитов в виде маленьких меток у каждого коммита, и в Actions). Необходимо предоставить детализацию, почему именно возвращена ошибка (для проверки, правильно ли реализовано задание 4).  
6. Внести исправления в .py-файл. Не должно остаться ни одной ошибки от pycodestyle. Снова сделать push. Workflow check должен быть завершен со статусом success: это также необходимо продемонстрировать.

**Ход работы**:





ghp\_ABC123 - токен

Настроим git в Git Bash.

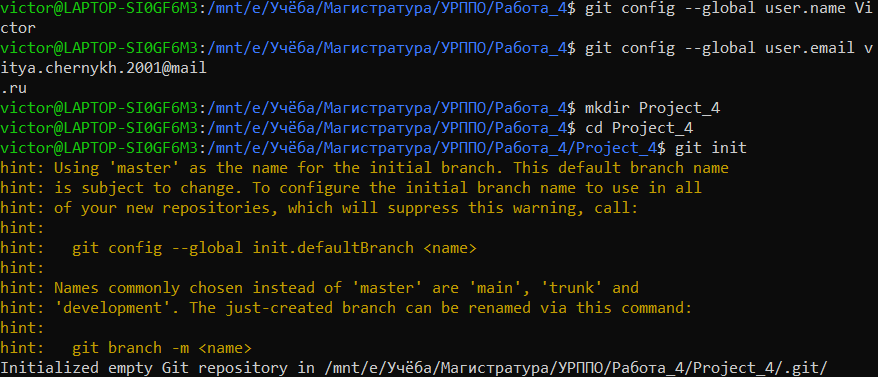


Рисунок 1 – Настройка git

Сделаем изменения (commits). Добавим файл README.md, основной файл main.py, функциональный файл views.py, файл для тестирования test.py и игнорщик .gitignore.

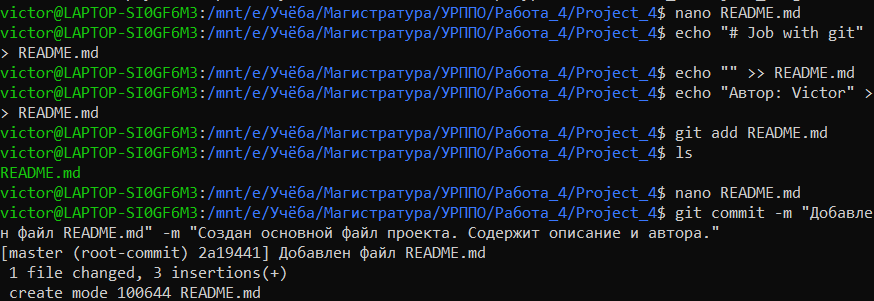


Рисунок 2 – README.md

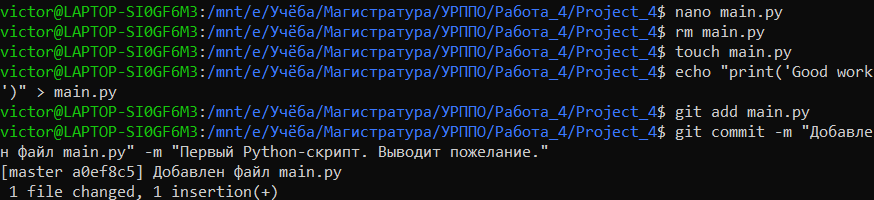


Рисунок 3 – main.py

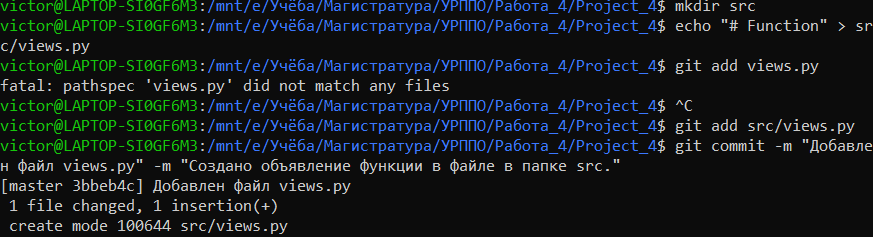


Рисунок 4 – views.py

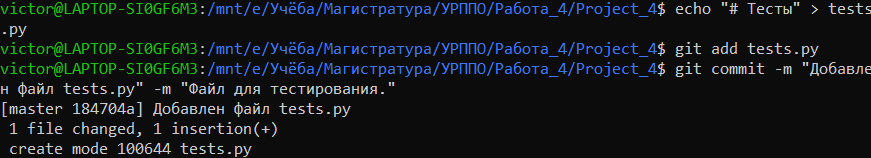


Рисунок 5 - tests.py

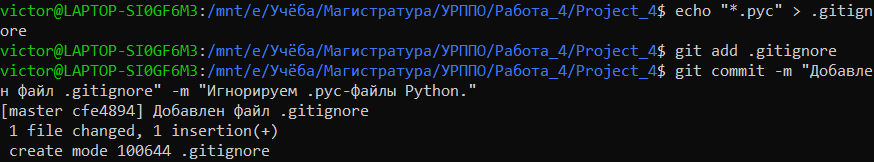


Рисунок 6 - .gitignore

Посмотрим статус проекта.



Рисунок 7 – Статус

Изучим историю коммитов с детализацией.

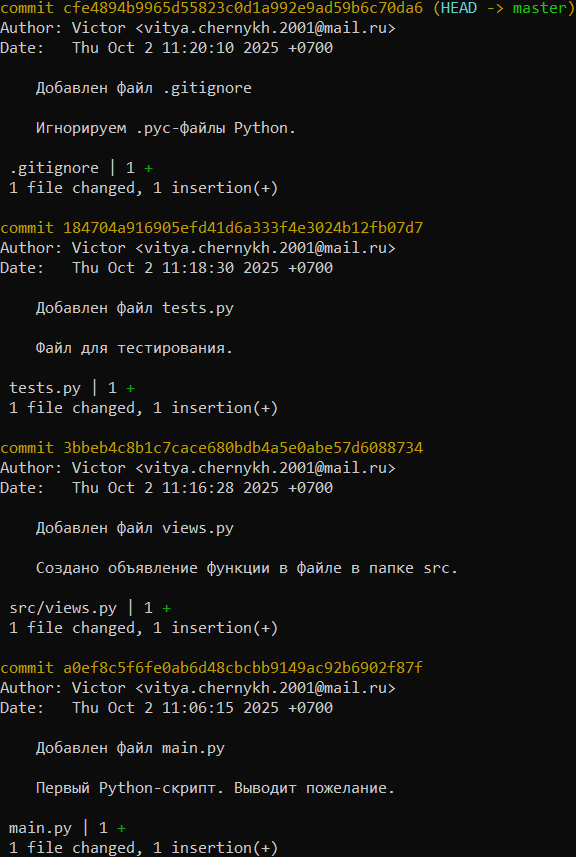


Рисунок 8 – История коммитов.

Грузим репозиторий на github.

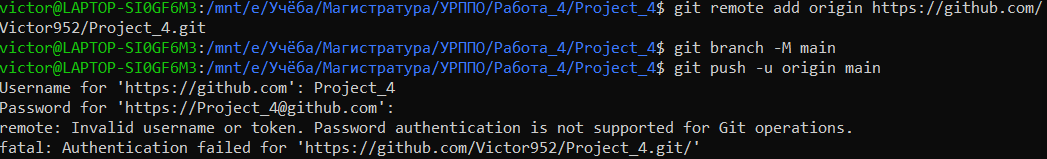


Рисунок 9 – Репозиторий github

Перейдём в другую ветку и сделаем ещё несколько изменений – файл с моделями models.py, файл с утилитами utils.py, админка admin.py, лицензия LICENSE, сборочный файл Makefile и папка в .gitignore.

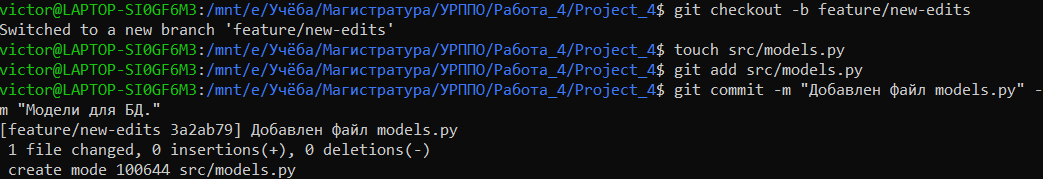


Рисунок 10 – models.py

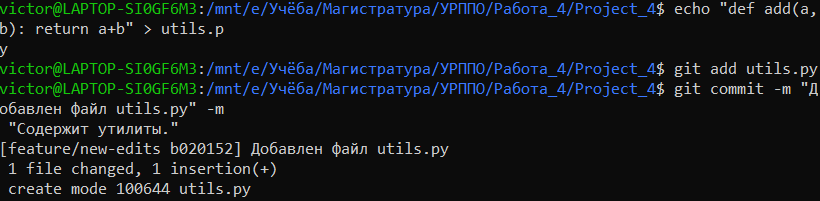


Рисунок 11 - utils.py

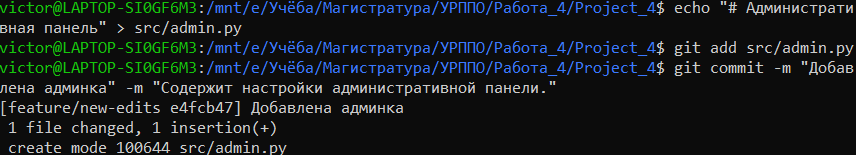


Рисунок 12 – admin.py

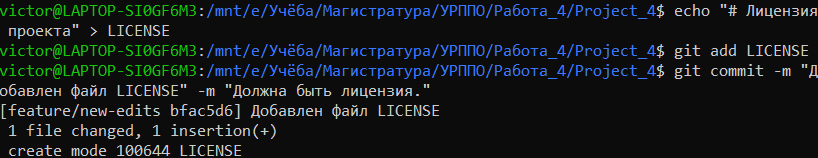


Рисунок 13 - LICENSE

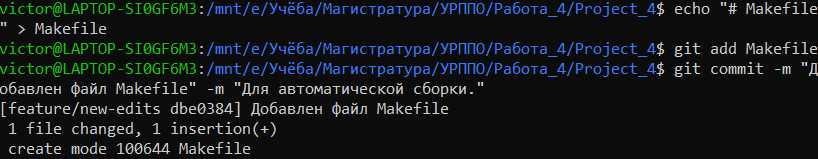


Рисунок 14 – Makefile

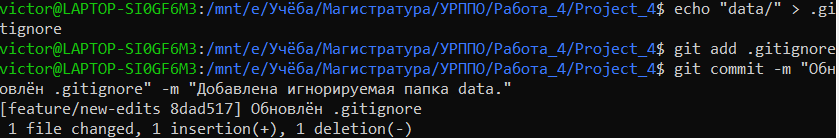


Рисунок 15 – data в .gitignore

Изменим коммиты – один переименуем, другой отредактируем, два объединим.

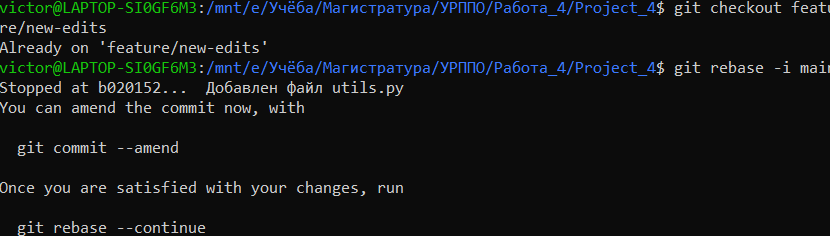


Рисунок 16 – Переход в rebase

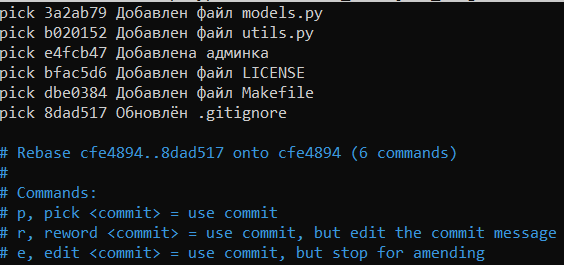


Рисунок 17 – rebase c коммитами

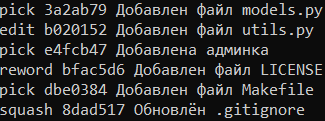


Рисунок 18 – Изменения в коммитах

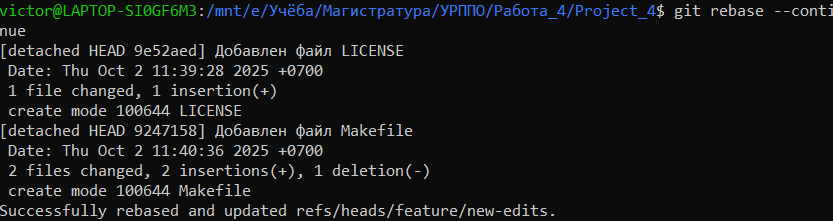


Рисунок 19 – Завершение работы rebase

Объединим ветки.

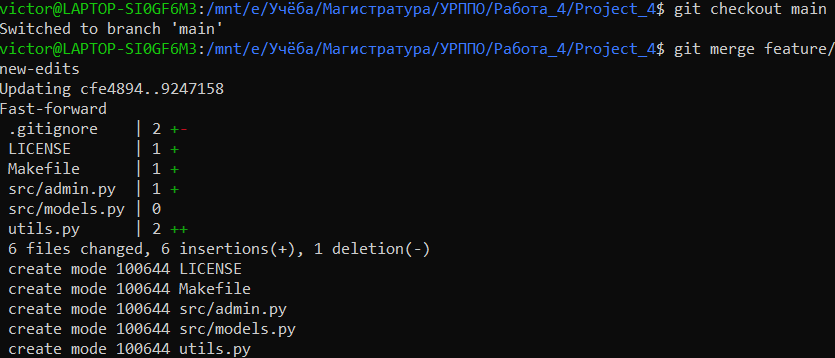


Рисунок 20 - merge

Загрузим изменения на github.

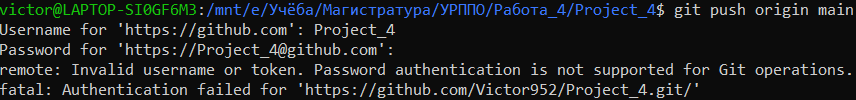


Рисунок 21 – push на github

Вывод: в ходе лабораторной работы был создан репозиторий на github, создан проект в git, выполнены несколько коммитов, созданы две ветки.