Лабораторная работа № 1

по дисциплине “Объектно ориентированное программирование”

Выполнил студент

Группы БФИ1901

Кумма К. С.

Москва 2020

**Цель работы**: изучение базовых функций системы контроля версий на примере 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. Создание веток и выполнение ряда изменений в проекте в созданных ветках

4. Слияние ветки с веткой мастер без конфликтов

5. Слияние ветки с веткой мастер с решением возникающих конфликтов

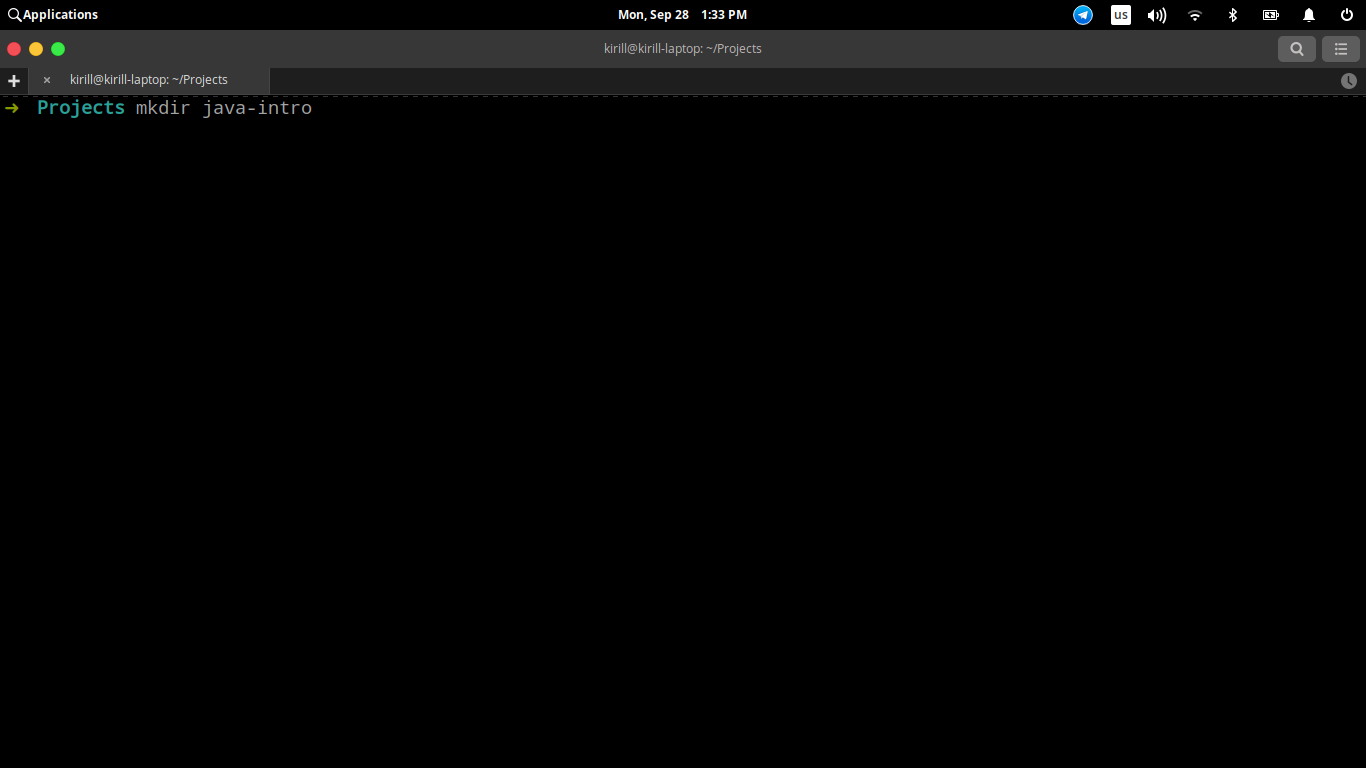
6. Откат изменений до одного из предыдущих коммитов

Возможна совместная работа студентов над одним проектом для изучения взаимодействия группы разработчиков и возможностей работы с СКВ.

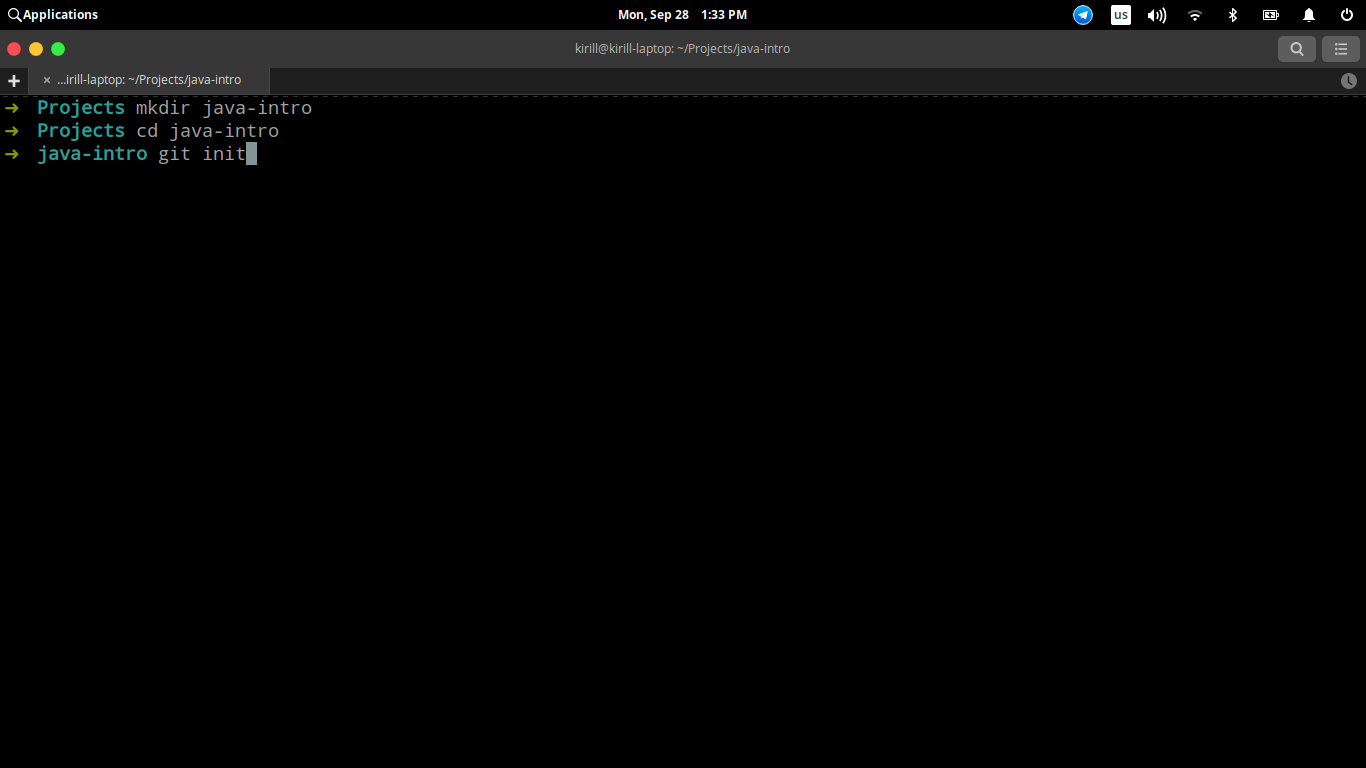
**Ход работы**:

Для того чтобы понять основы работы с Git, напишем простую программу, состоящую из двух классов. Первый класс Main будет входной точкой нашей программы, а второй класс Box будет вспомогательным классом, создающим простую абстракцию данных.

Создаем папку нашего проекта.



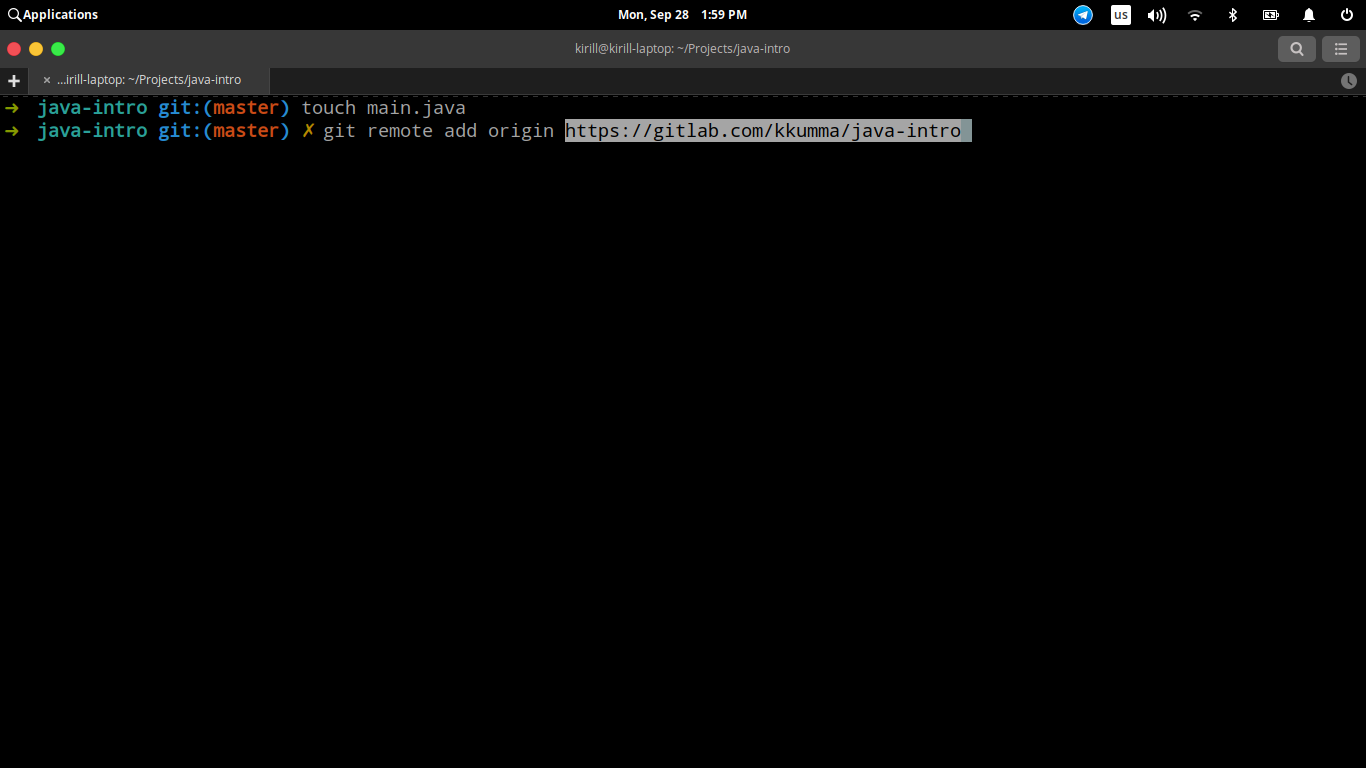
Переходим в нее и инициализируем Git командой “git init”.



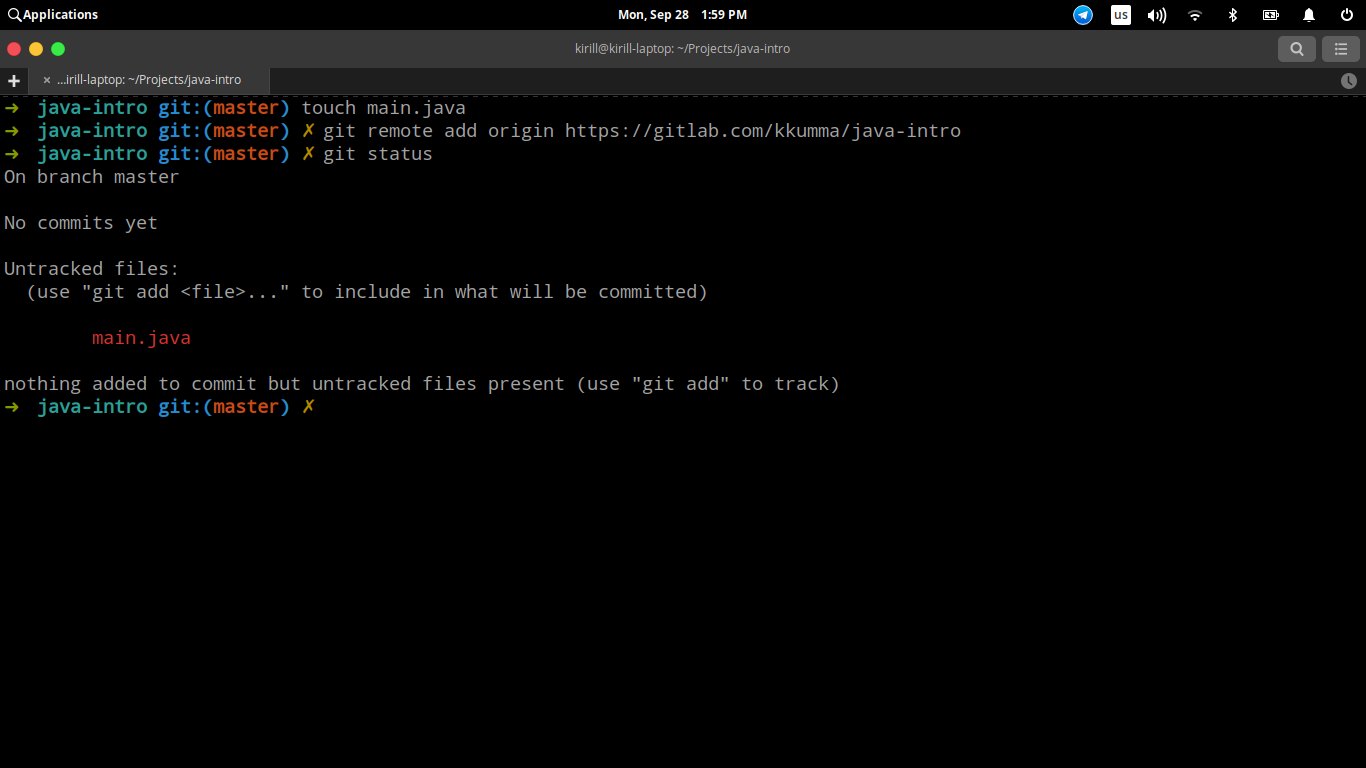
Создаем входную точку нашей программы.



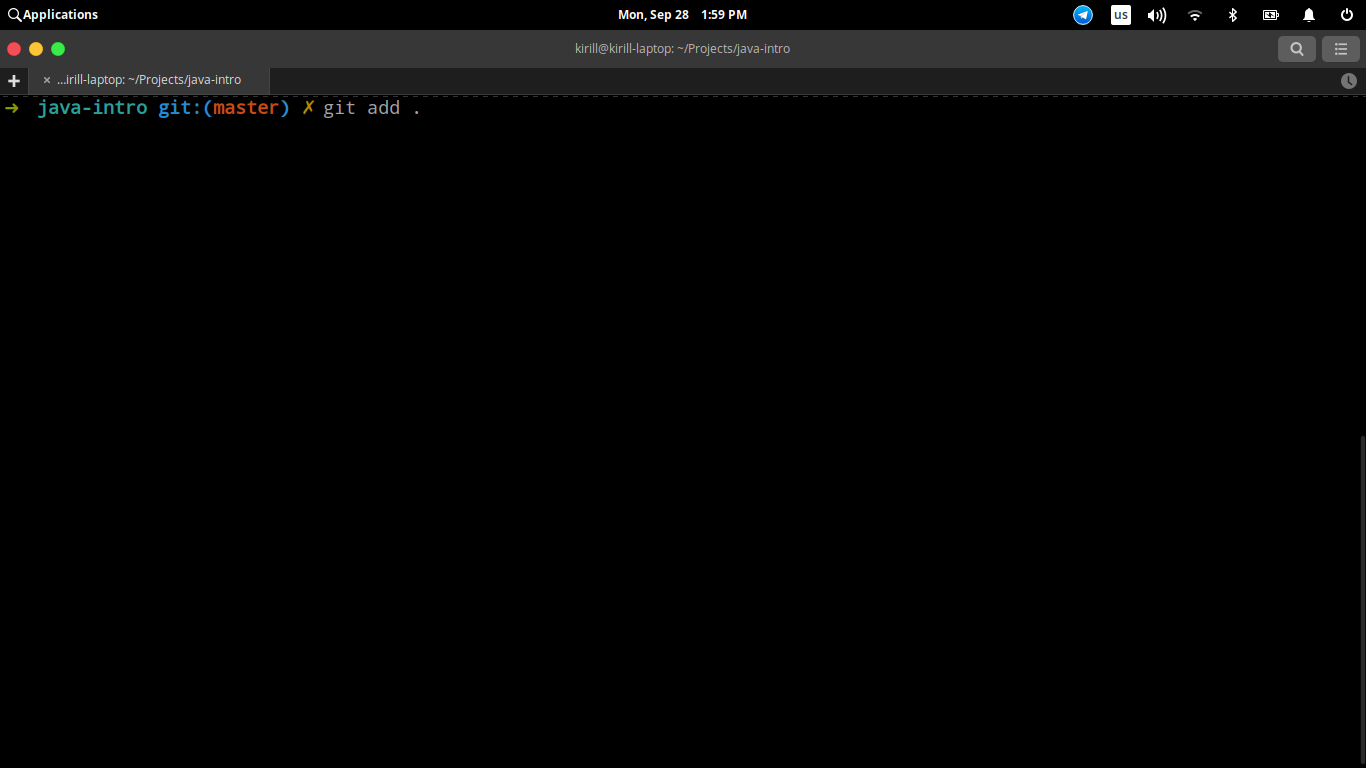
Подключаемся к удаленному репозиторию с помощью команды “git remote add origin”.



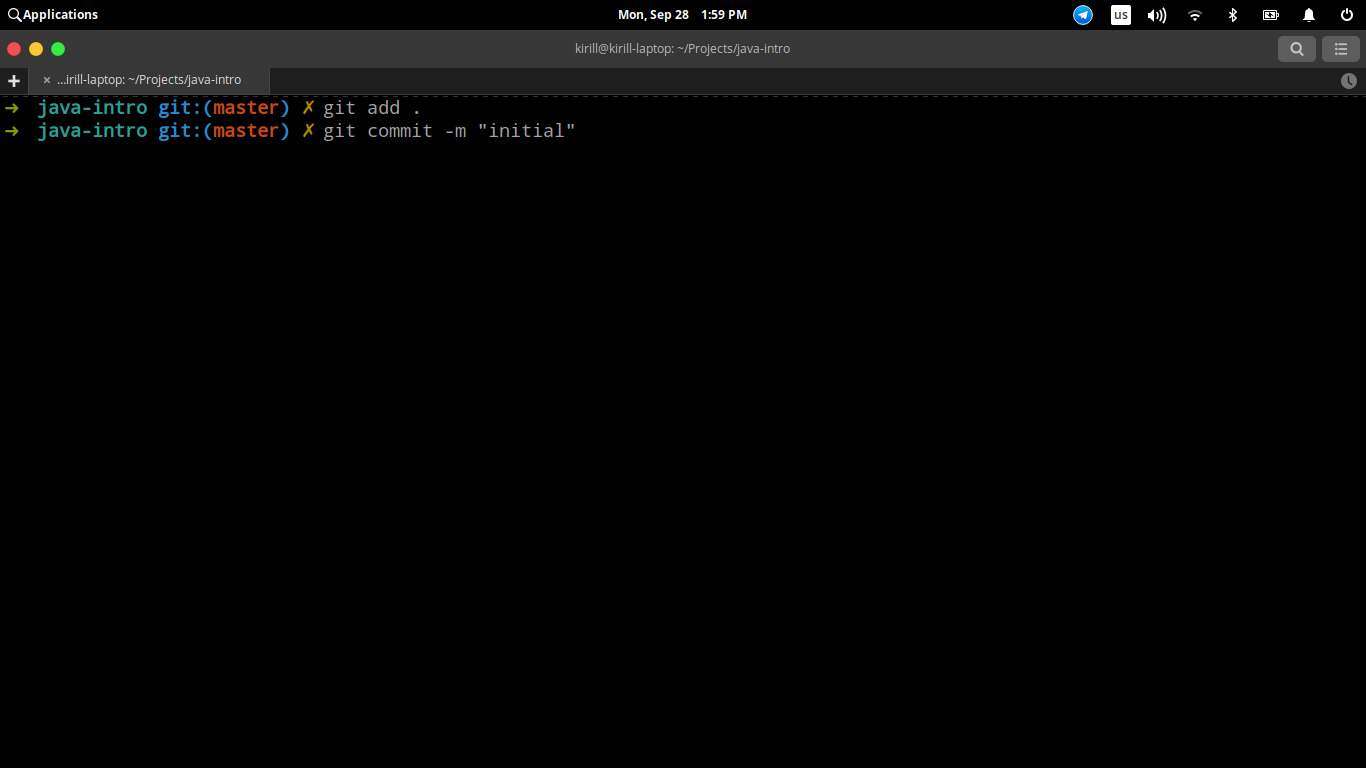
С помощью команды “git status” проверяем состояние нашего локального репозитория. Как мы видим, у нас есть один созданный файл, который еще не добавлен в Git.



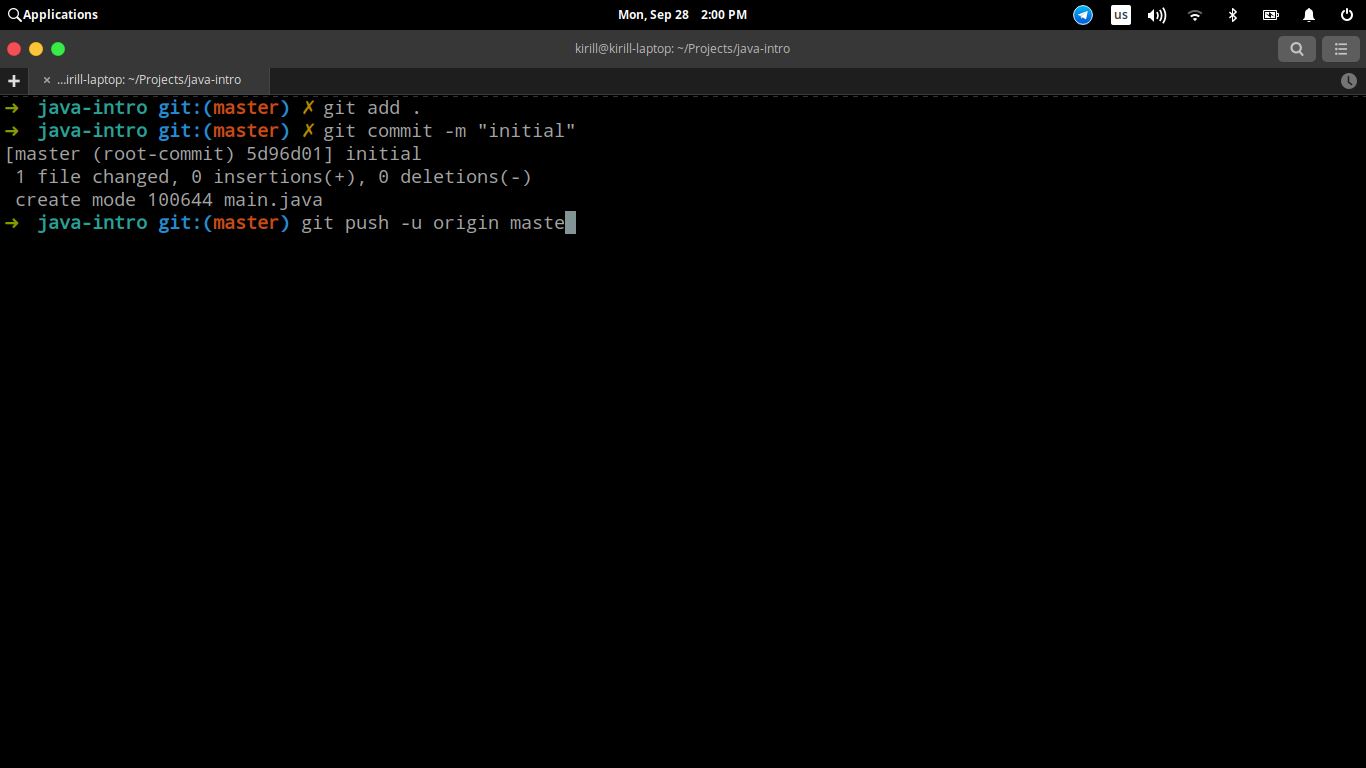
С помощью команды “git add .” добавляем этот файл в Git (аргумент точка обозначает, что добавить нужно все файлы, т.к. у нас он только один, данный вариант нам подходит).



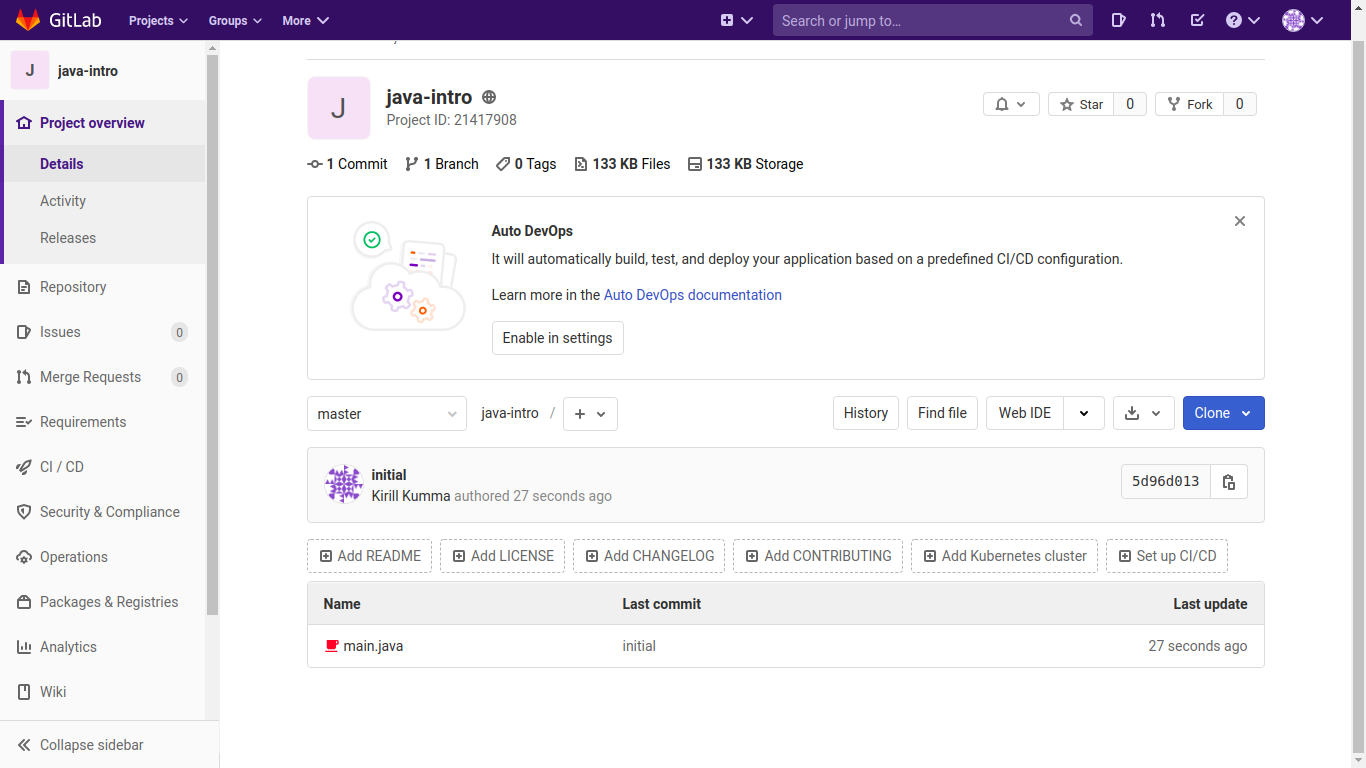
Делаем первый коммит с помощью команды “git commit -m” (флаг -m дает нам возможность сразу же написать пояснительное сообщение, если написать данную команду без флага, то сообщение нужно будет написать в открывшемся редакторе кода).



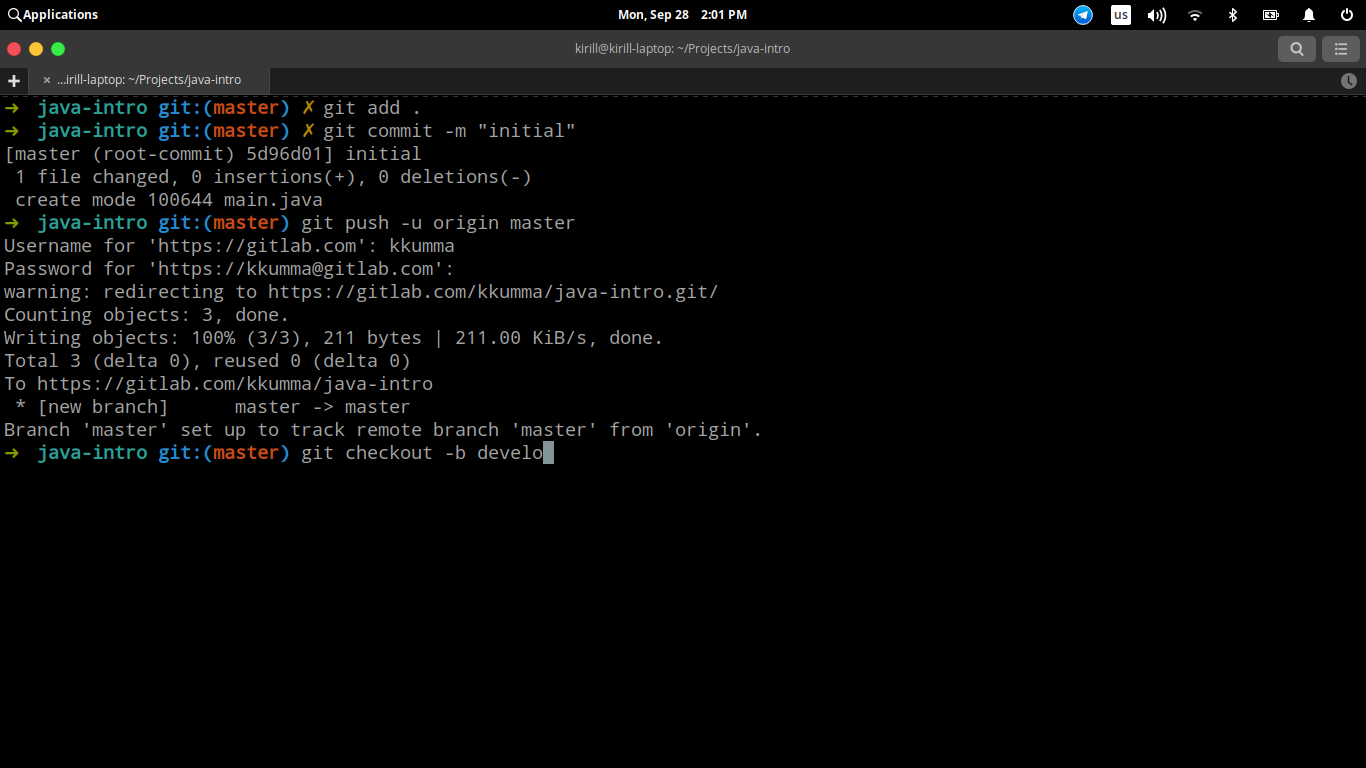
С помощью команды “git push” отправляем наши изменения в удаленный репозиторий.



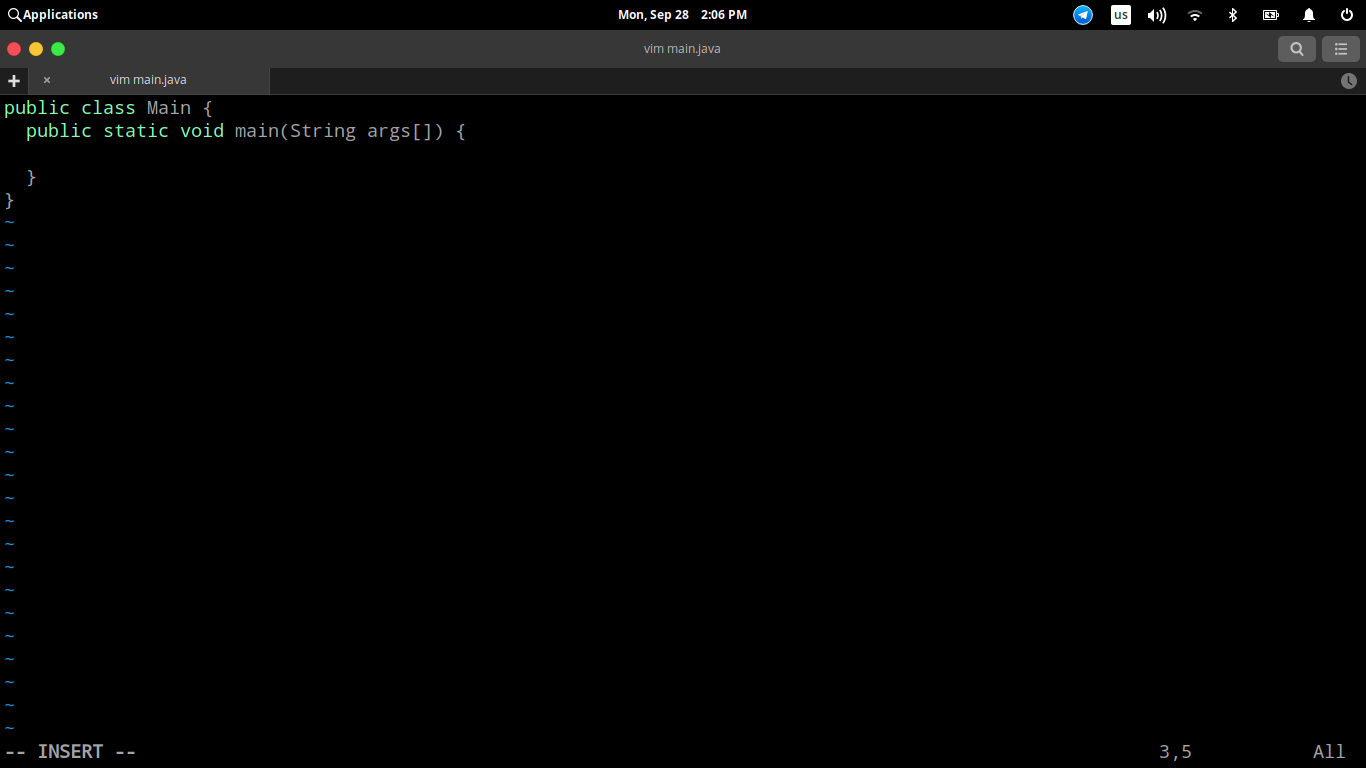
Как мы видим, теперь в нашем репозитории появился файл main.java.



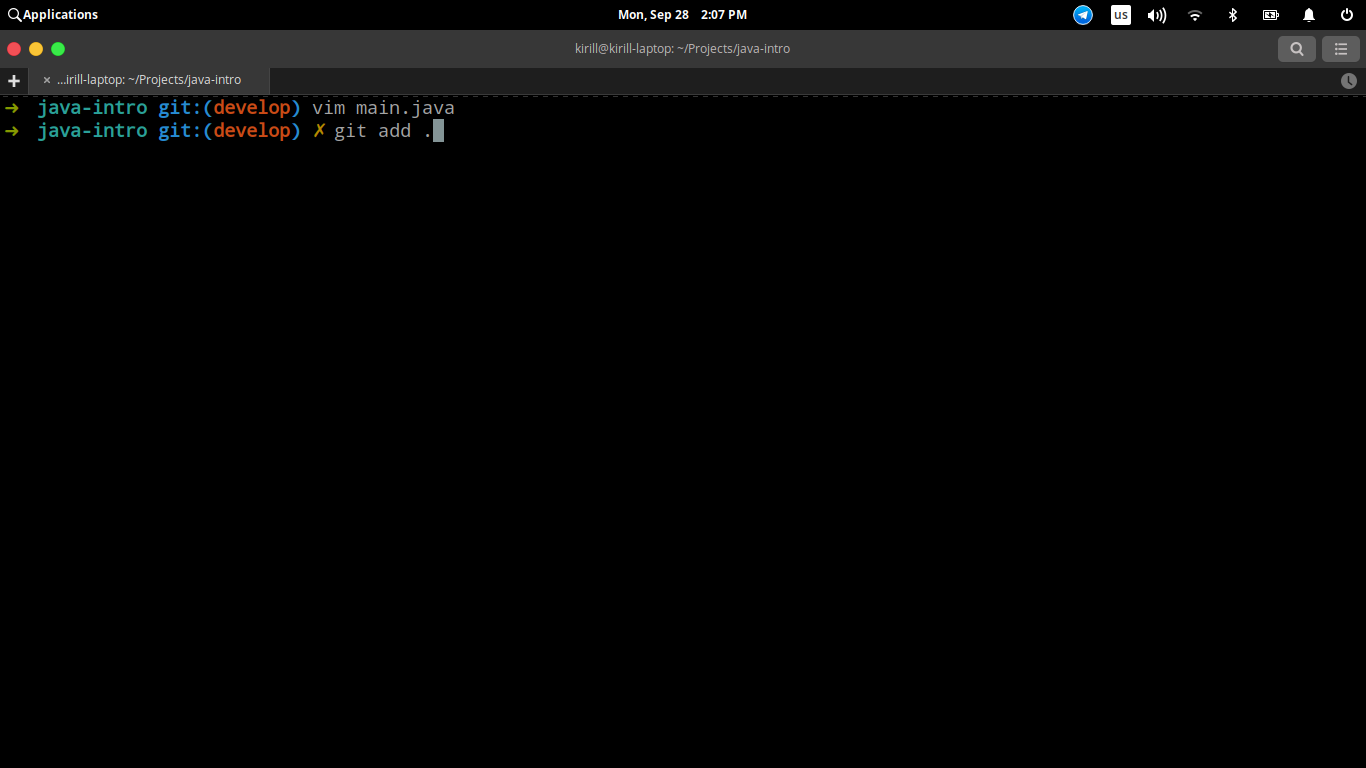
Git позволяет работать нам в разных ветках, поэтому на время разработки программы мы перейдем в ветку develop с помощью команды “git checkout –b develop”.



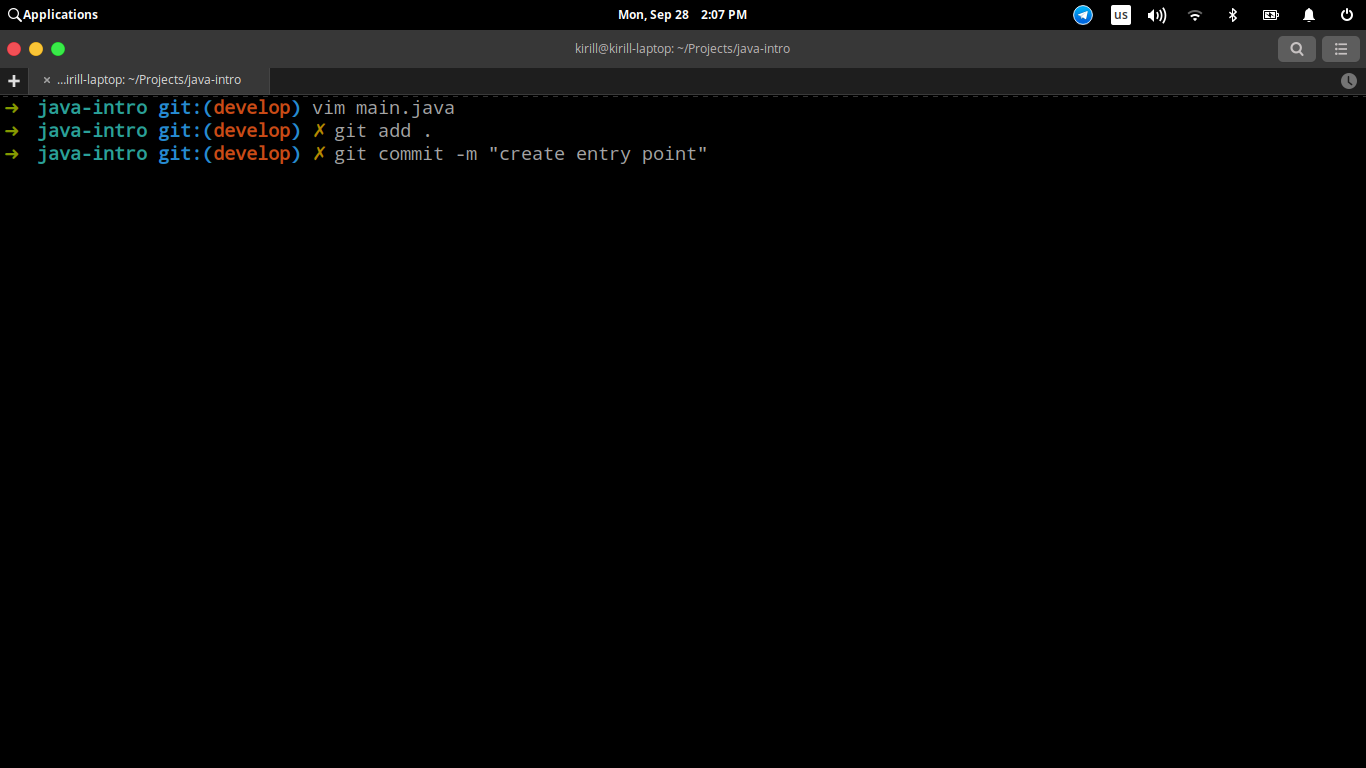
Открываем файл main.java и создаем в нем класс Main с одноименном методом, который будет нашей входной точкой.



Снова добавляем наш файл в Git.



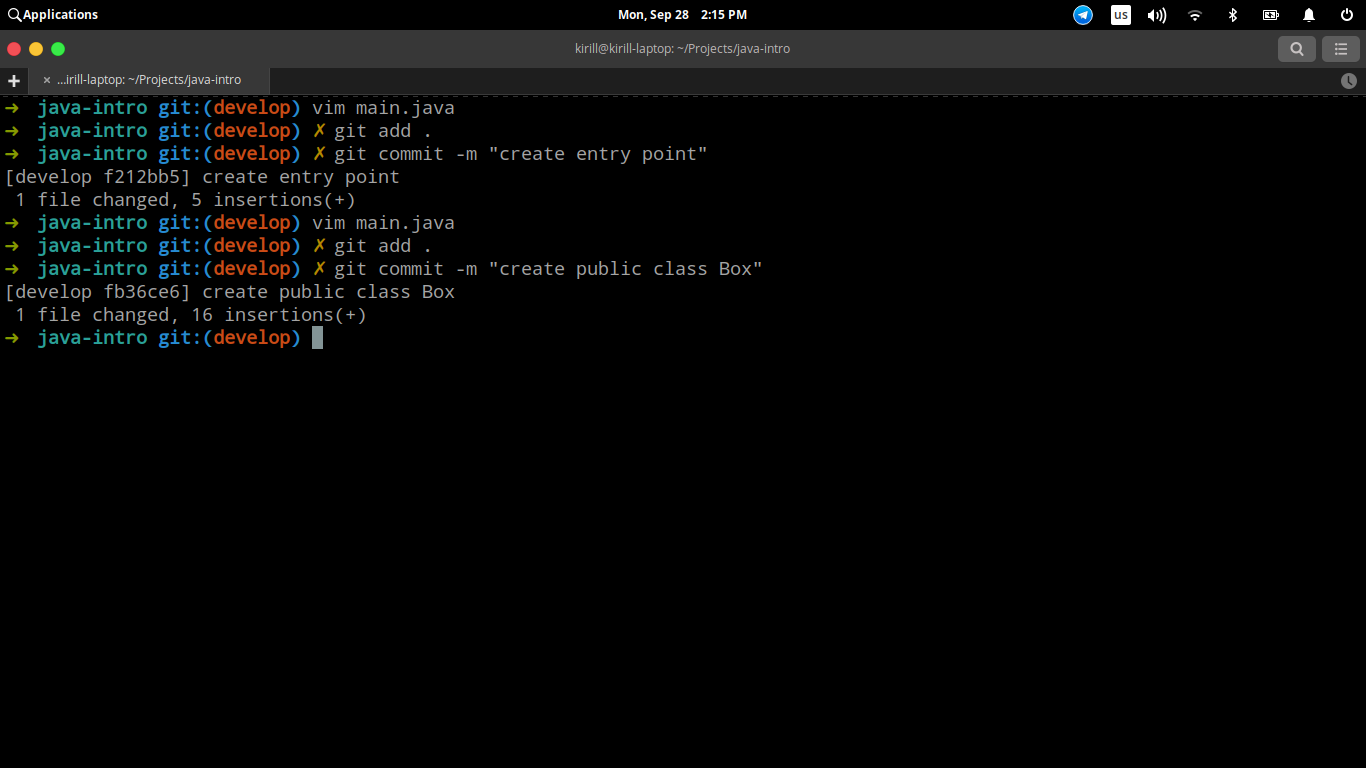
И делаем первый коммит в ветке develop.



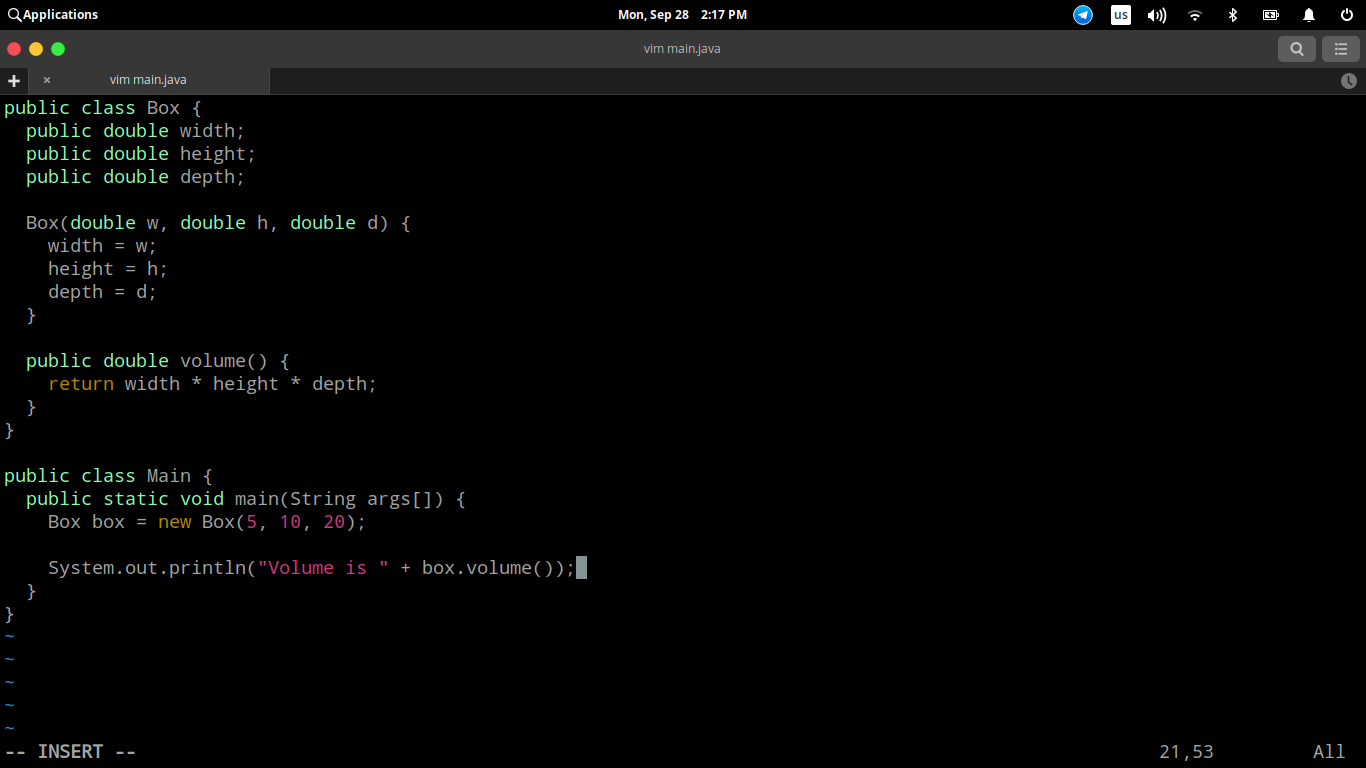
Создаем класс Box с конструктором и методом volume, который возвращает объем.



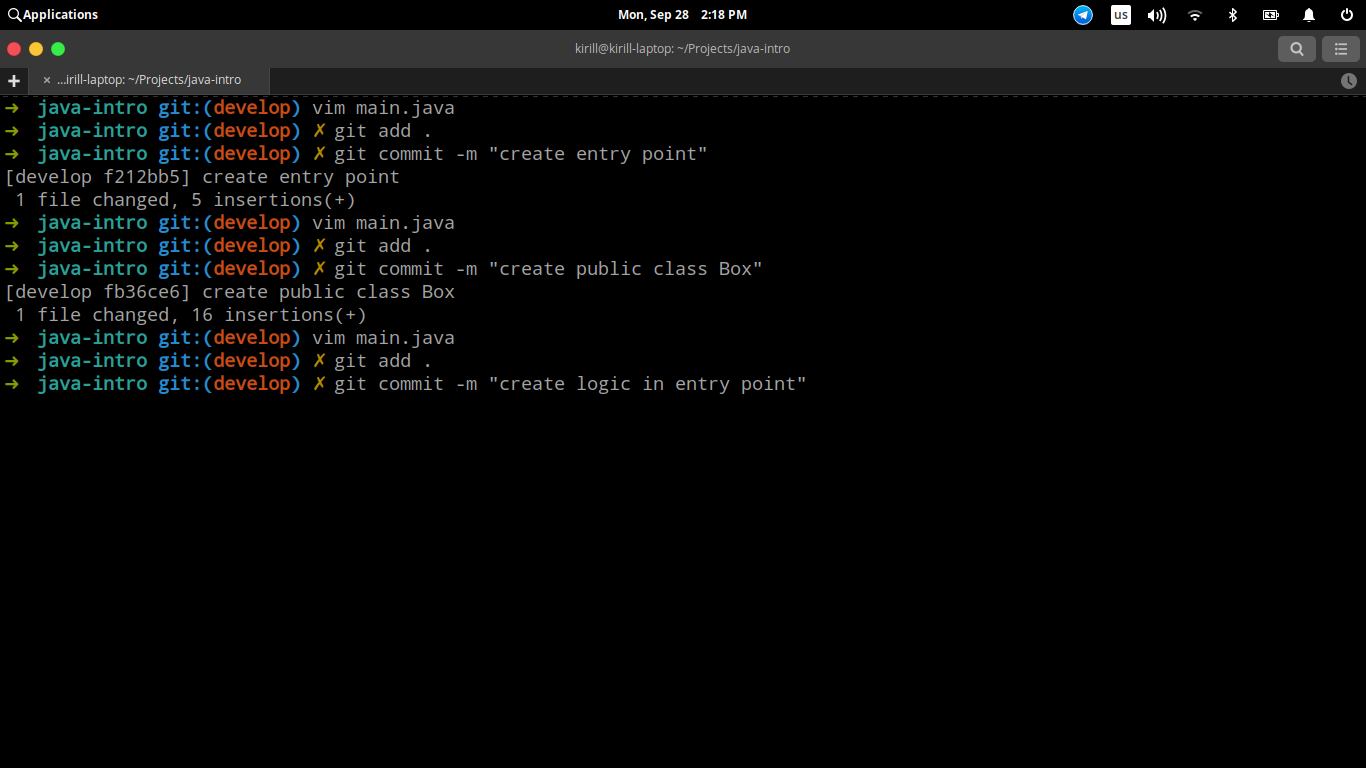
Делаем еще один коммит с описанием того, что мы сделали.



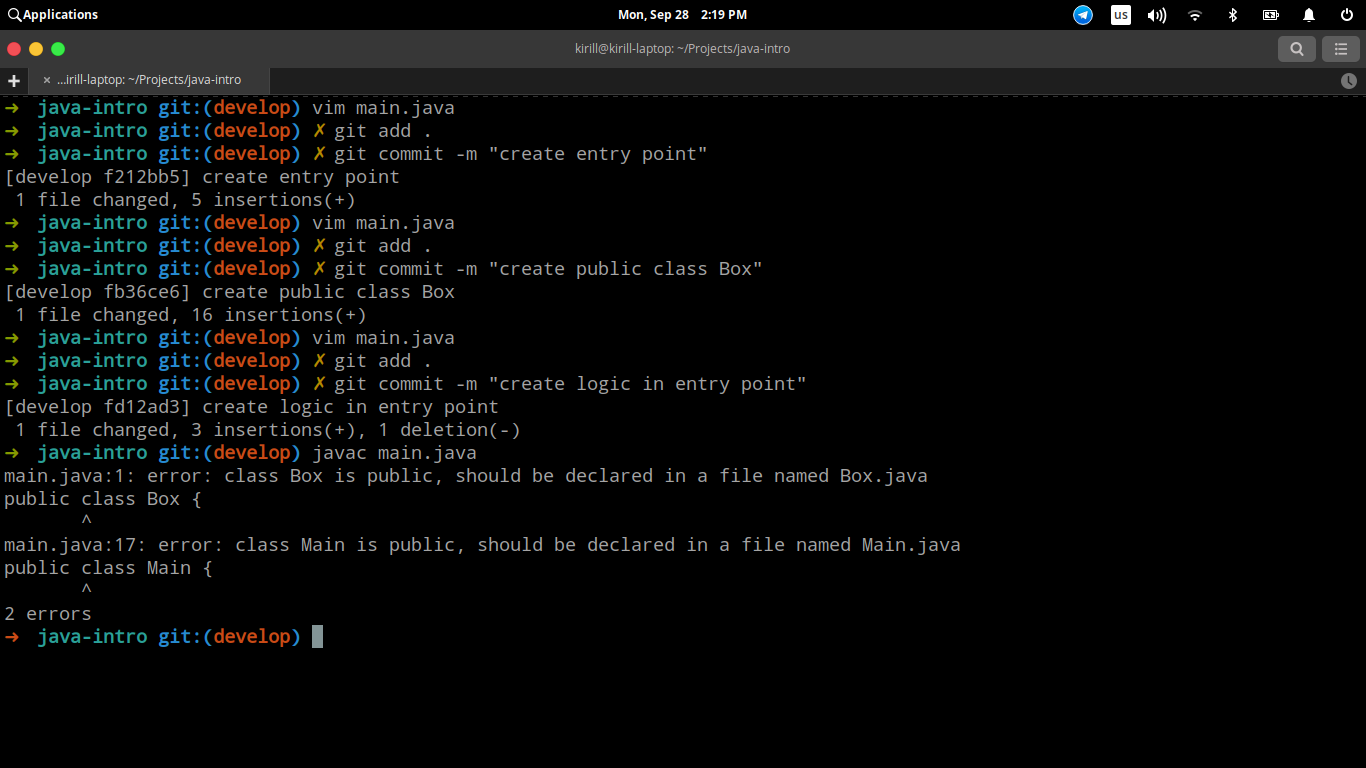
Реализуем метод main. В нем мы создаем экземпляр класса Box и выводим в консоль его объем.



Делаем еще один коммит.



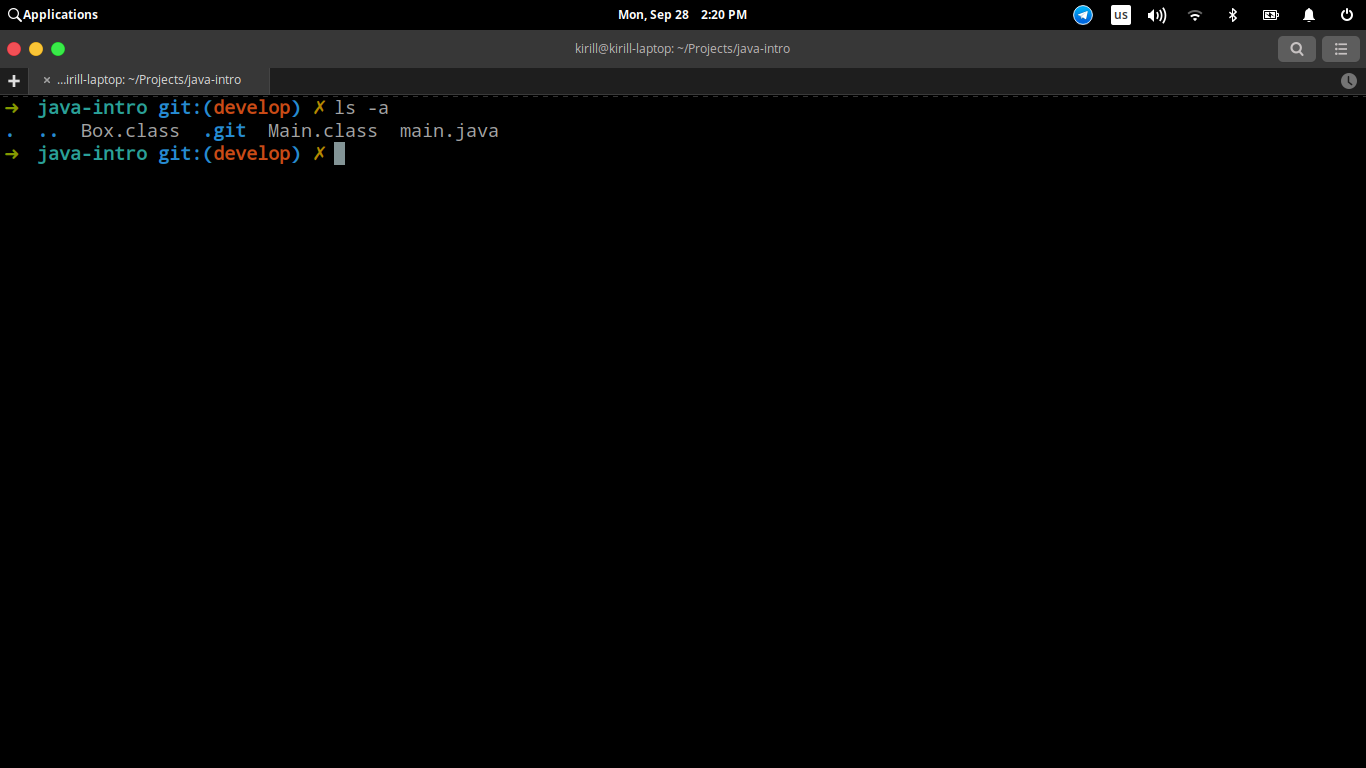
С помощью команды “javac” пробуем скомплировать нашу программу. Как мы видим, компилятор выдает нам ошибку, в связи с тем, что мы поставили модификаторы доступа у двух классов в одном файле, данные модификаторы нужно убрать.



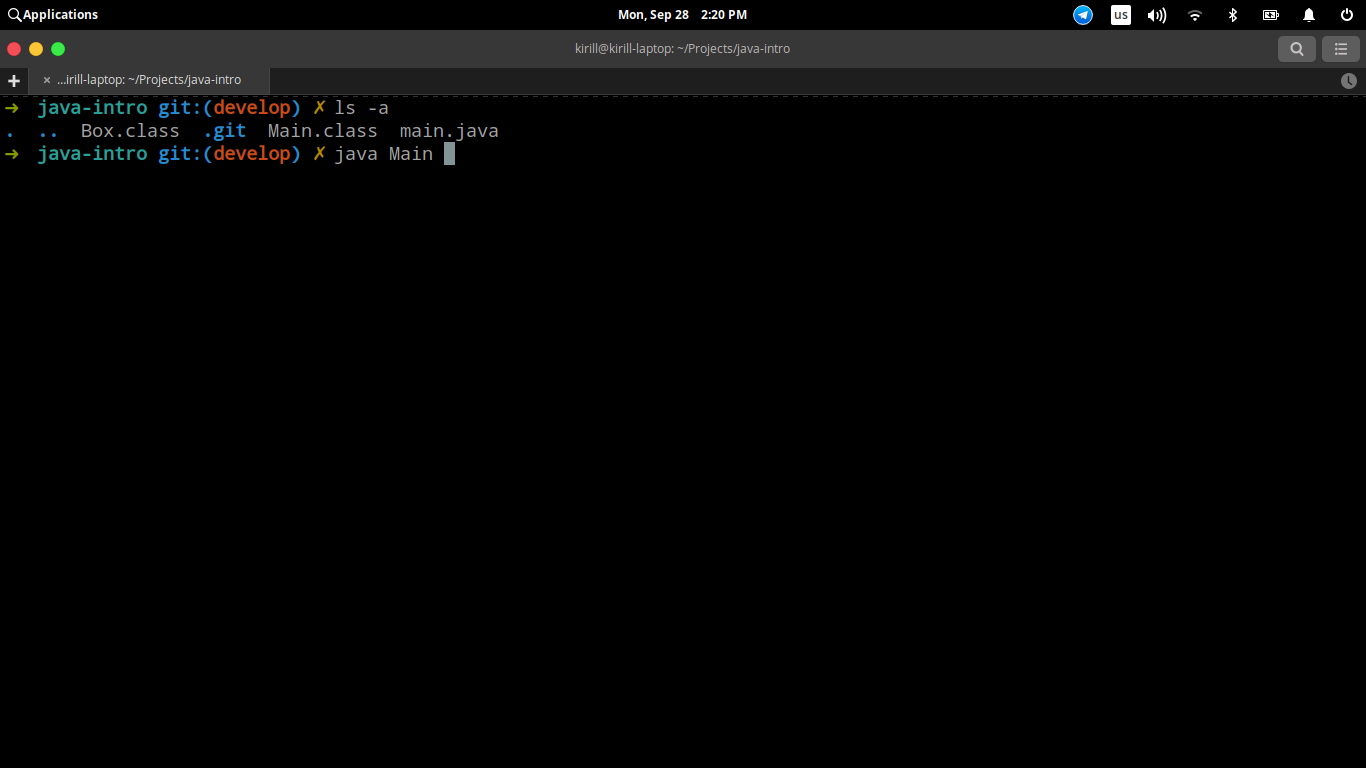
Исправляем ошибку и успешно компилируем программу.



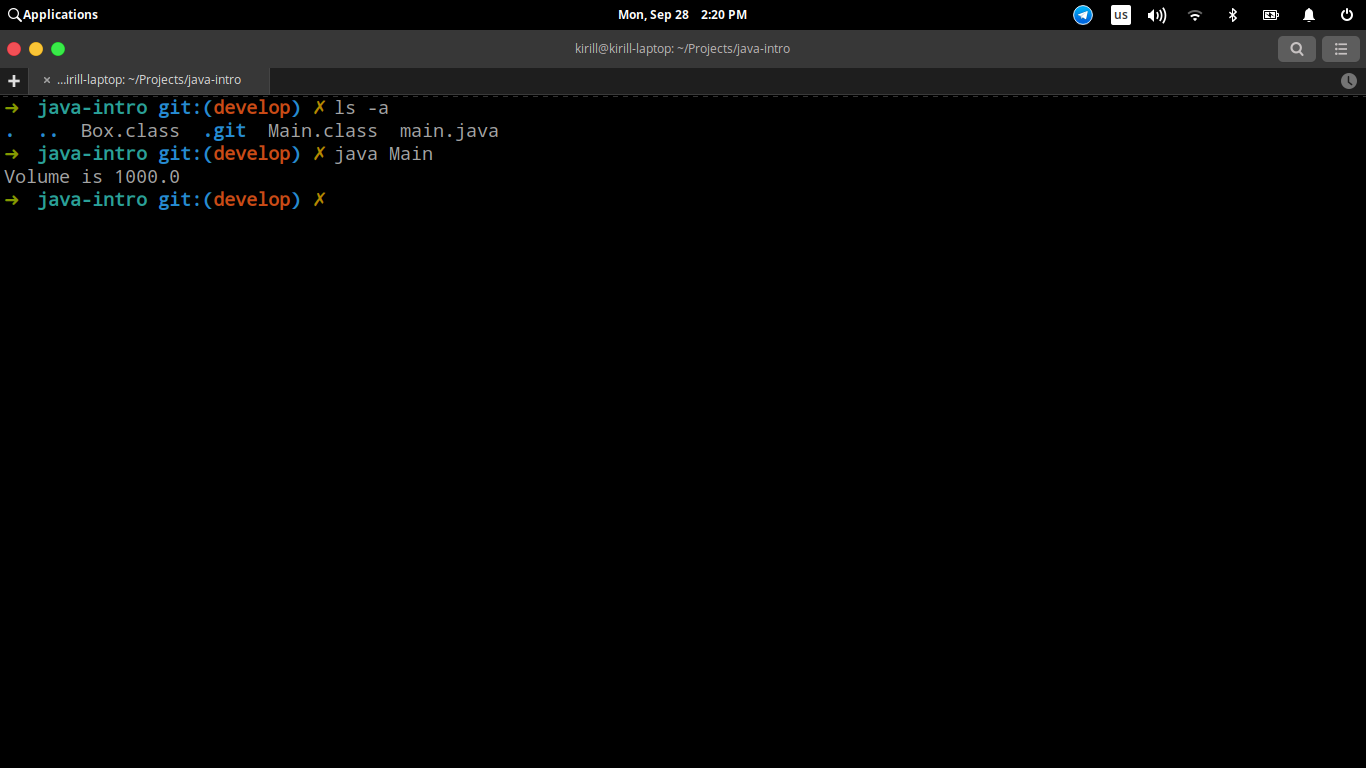
Теперь в нашей папке появились два файла. Один - скомпилированный класс Box, а другой - наша входная точка.



Вызываем команду “java” и посылаем первым аргументом файл с нашей входной точкой.



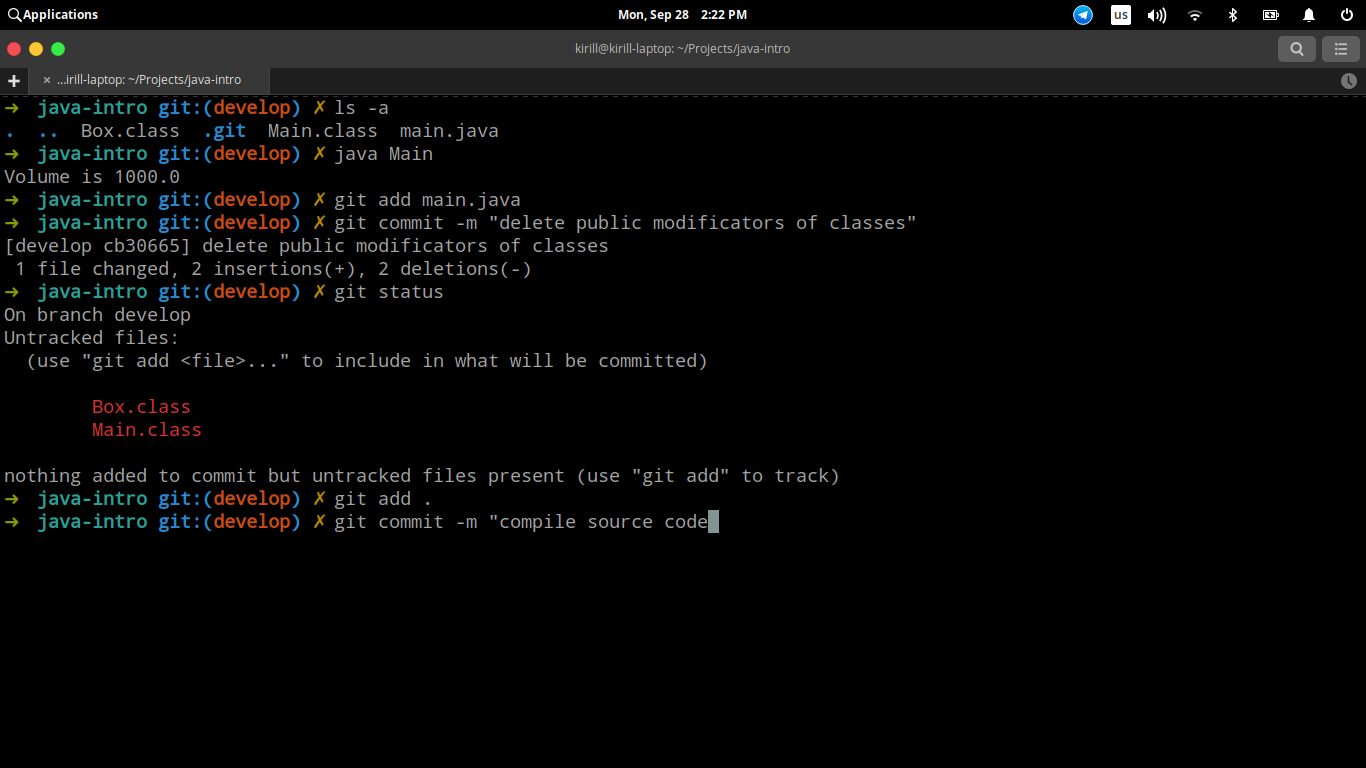
Программа успешно запустилась. Результат работы мы видим в консоли.



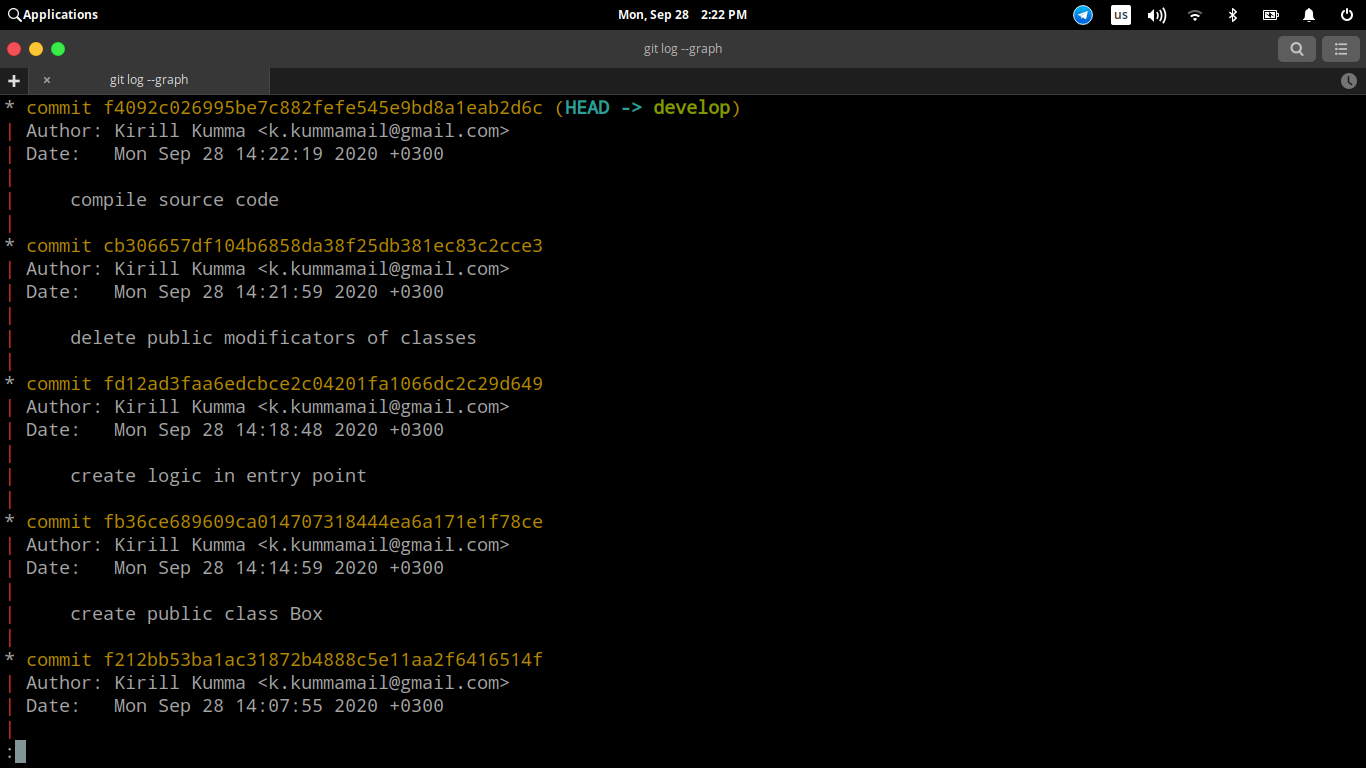
Делаем коммит об исправлении ошибки.



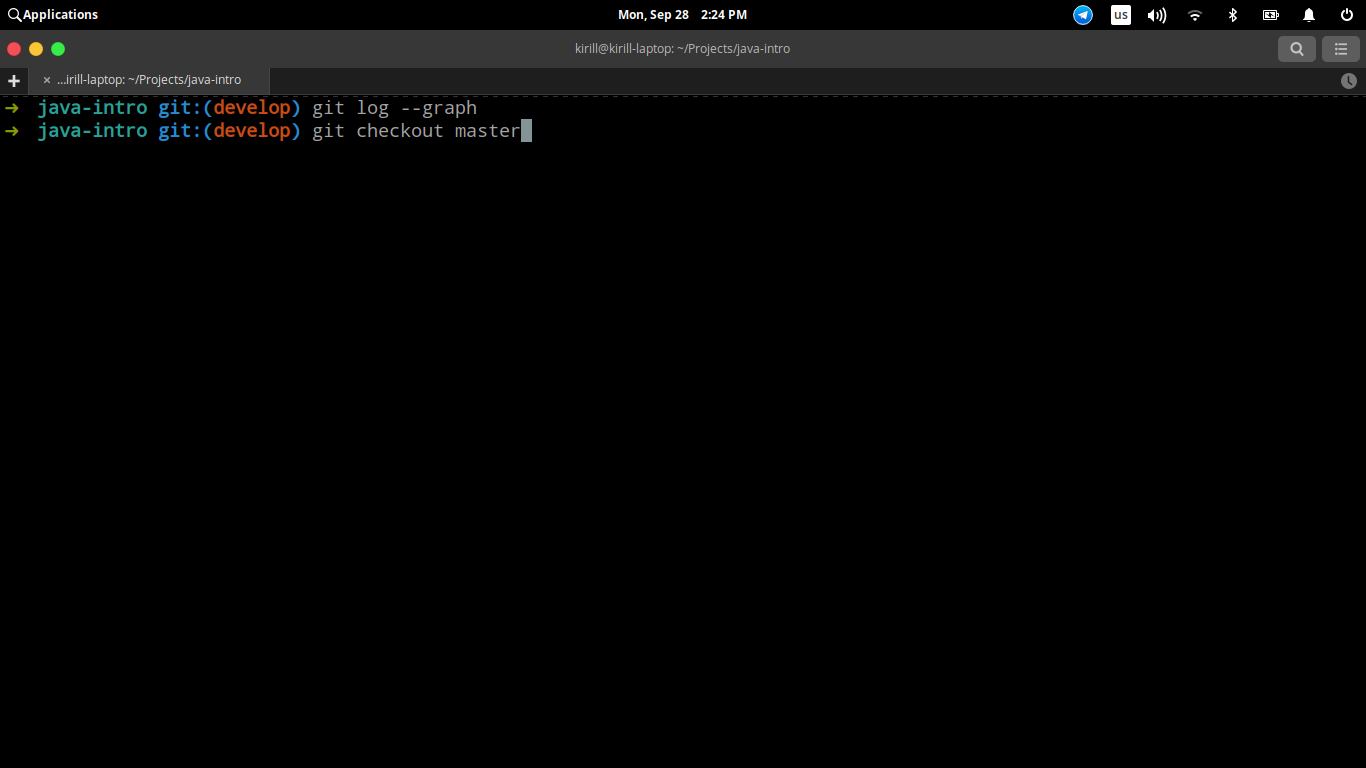
Обычно в репозитории хранятся только исходные файлы программы. Но так как в данной лабораторной работе не предусмотрено использование .gitignore файлов, мы добавляем скомпилированные файлы в Git.



С помощью команды “git log” или “git log --graph” мы можем посмотреть историю разработки программы. В коммите содержится информация о том, кто его сделал. Дата и время, а так же специальный код, с помощью которого можно откатиться к конкретному коммиту.



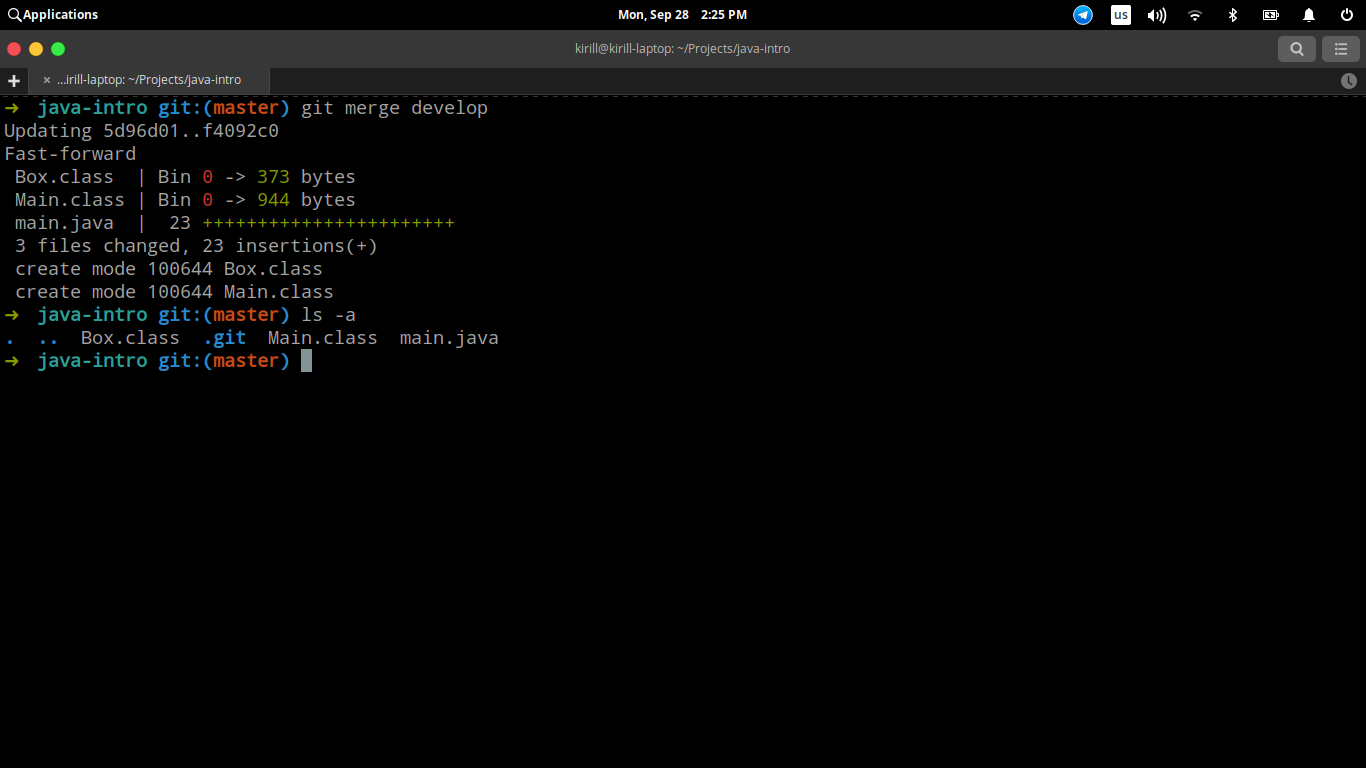
Теперь нам нужно перенести нашу программу на мастер ветку, для этого переходим на нее.



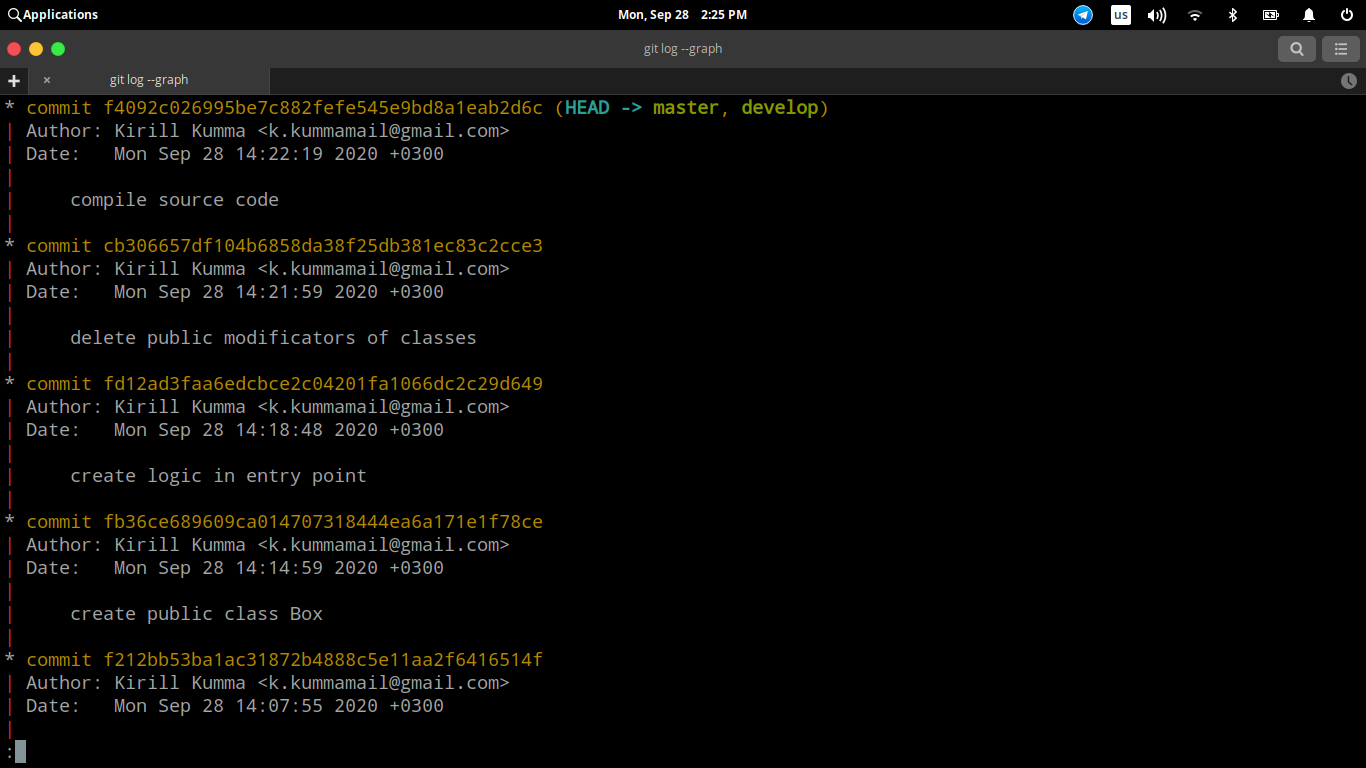
И делаем слияние веток с помощью команды “git merge”.



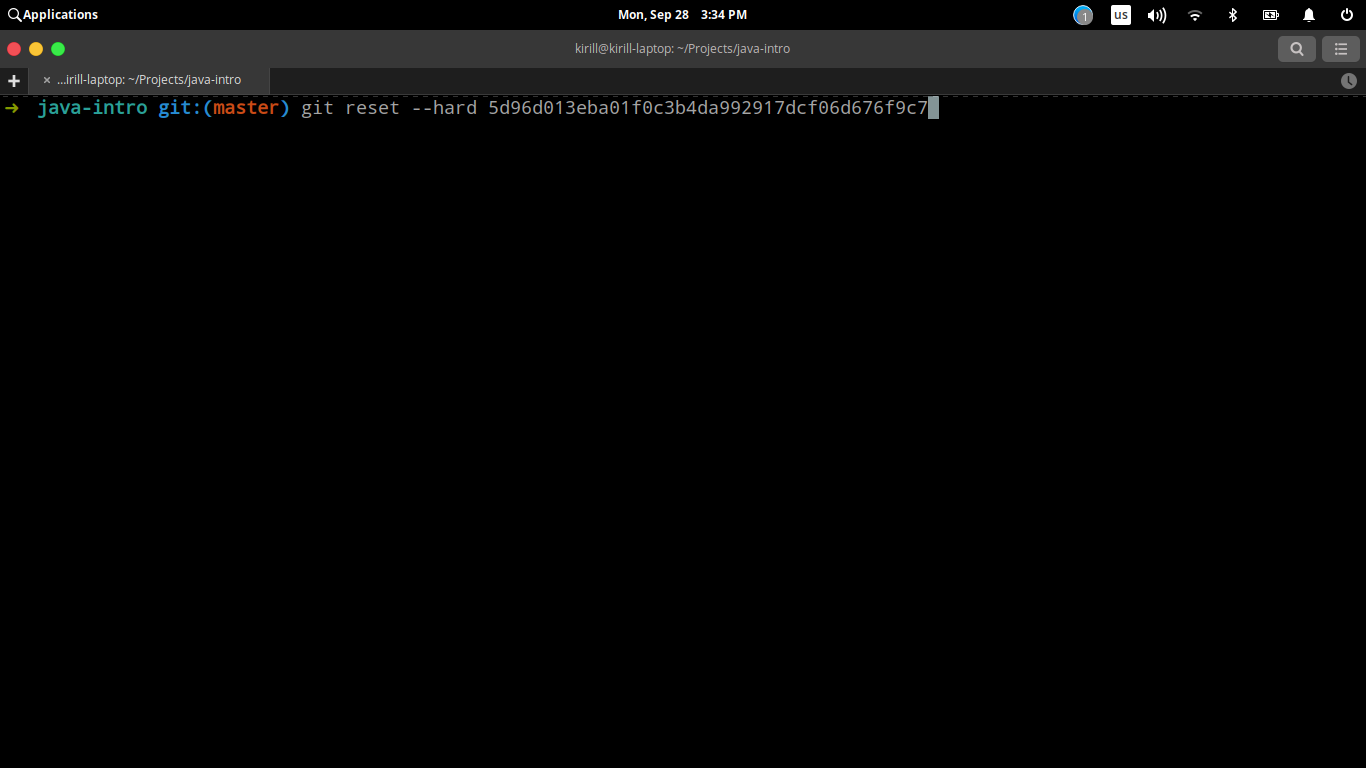
Теперь ветка master содержит все файлы нашей программы.



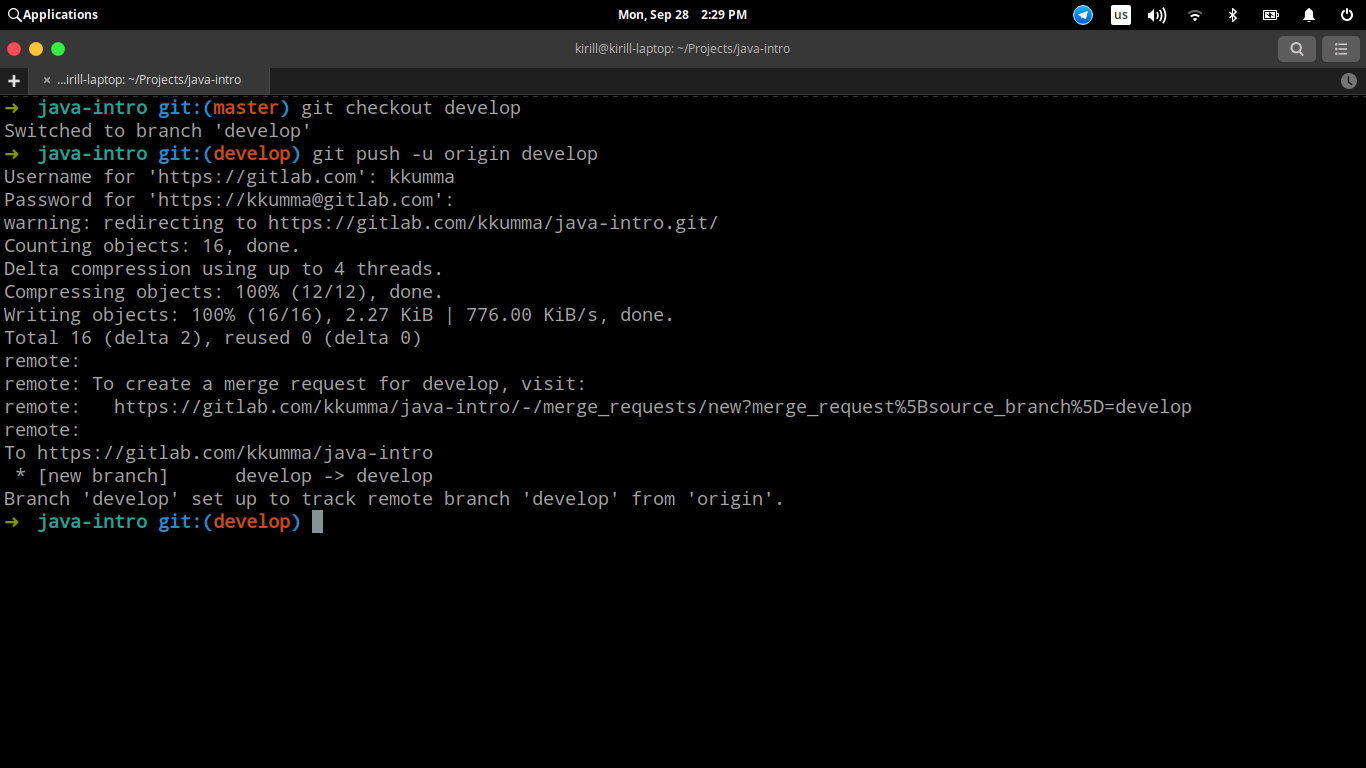
Но данный способ слияния веток является менее предпочтительным. По многим конвенциям работы с Git, на master ветке находятся только коммиты с готовыми версиями программы для удобного перехода между ними. А как мы видим, команда “git merge” перенесла все коммиты с нашей develop ветки на master ветку.



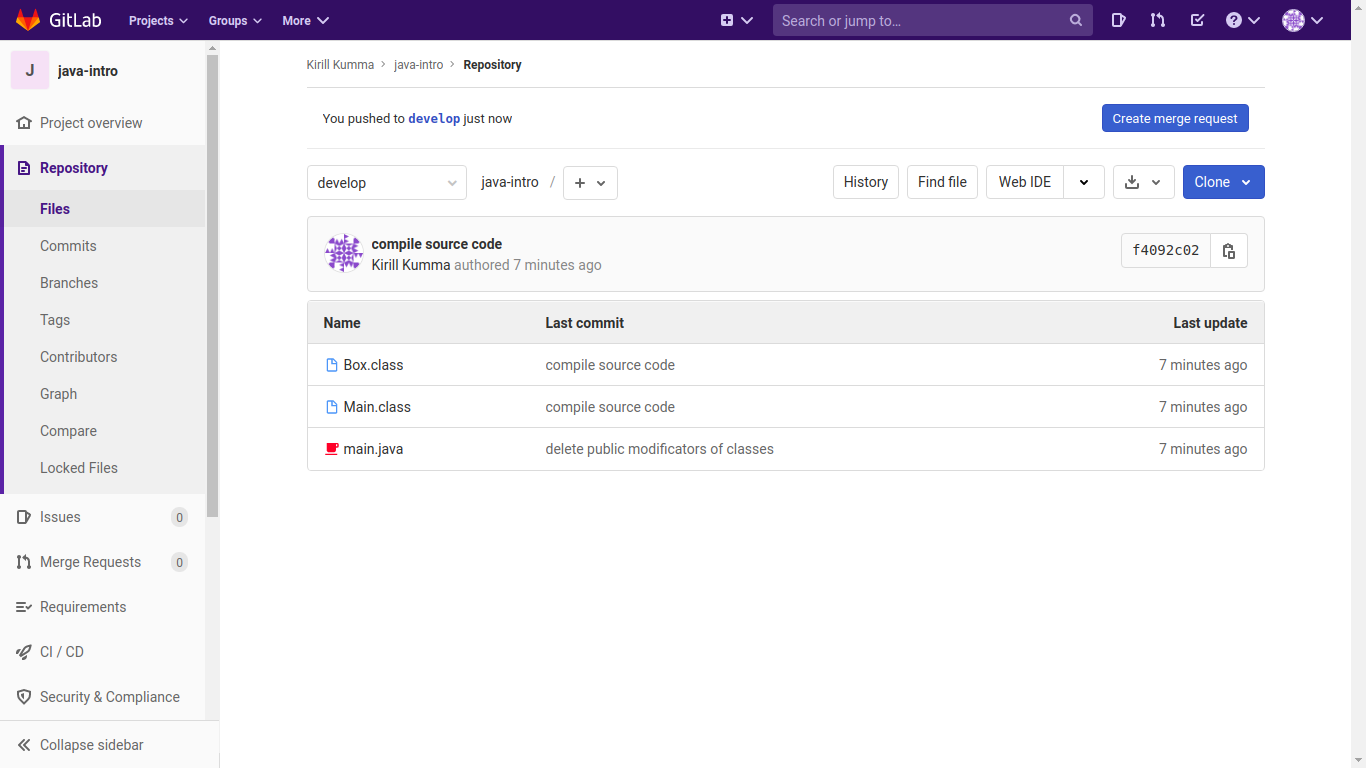
Для того чтобы откатить изменения, мы используем команду “git reset” и аргументом передаем наш самый первый коммит (флаг --hard, означает, что откатить нужно не только историю коммитов, но и состояние файлов).



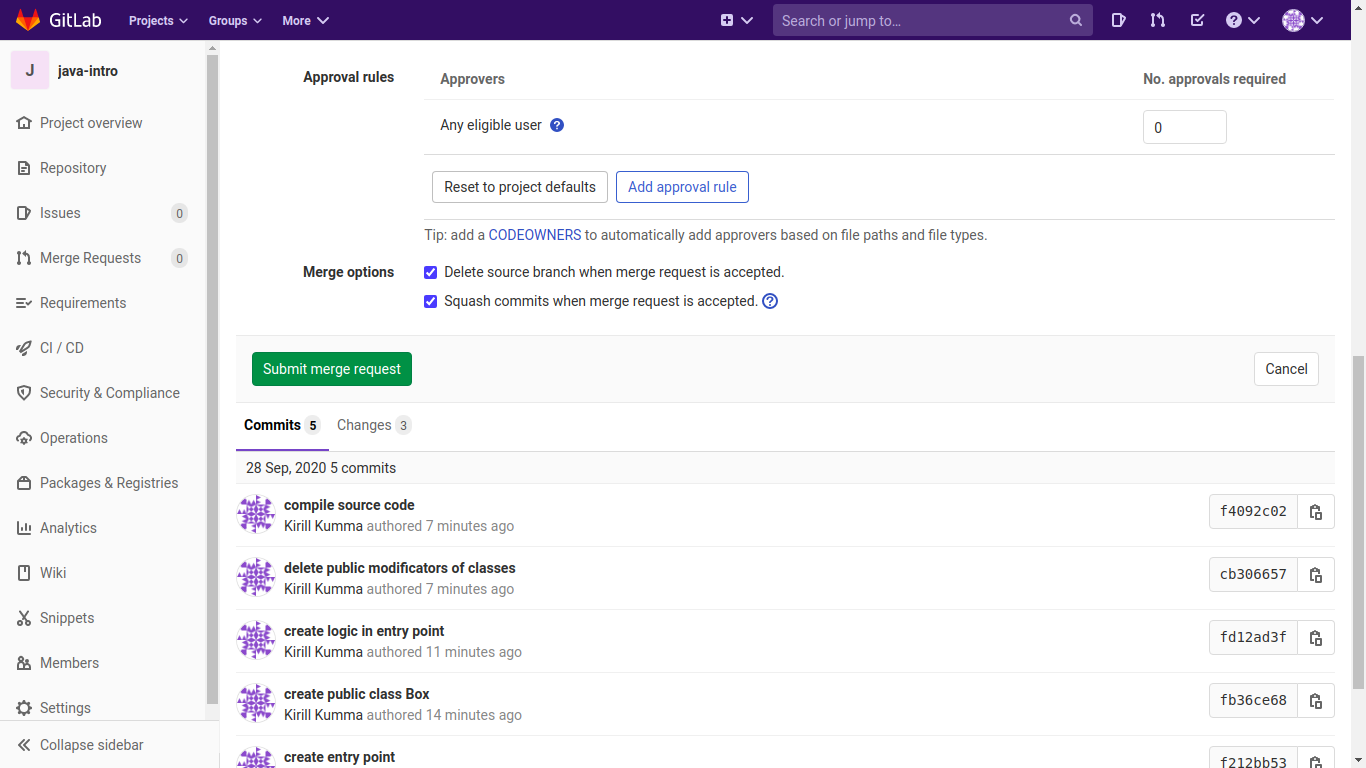
Чтобы перенести результаты нашей работы на ветку master без добавления всех коммитов, мы отправляем изменения develop ветки в удаленный репозиторий.



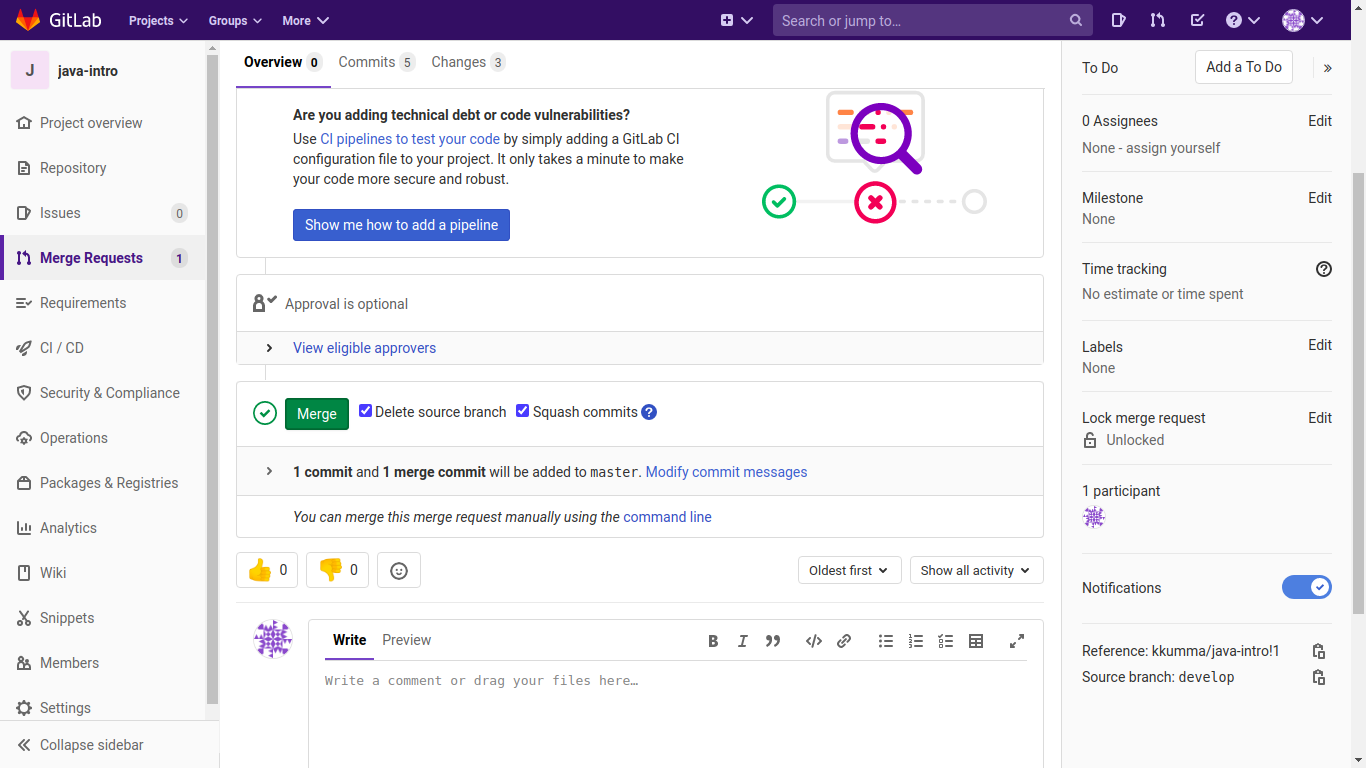
Теперь мы можем видеть эти изменения в удаленном репозитории и создать Merge Request (аналог Pull Request в GitHub), который позволит нам слить ветки в режиме Squash, данный режим объединяет все коммиты ветки в один и уже потом производит слияние.



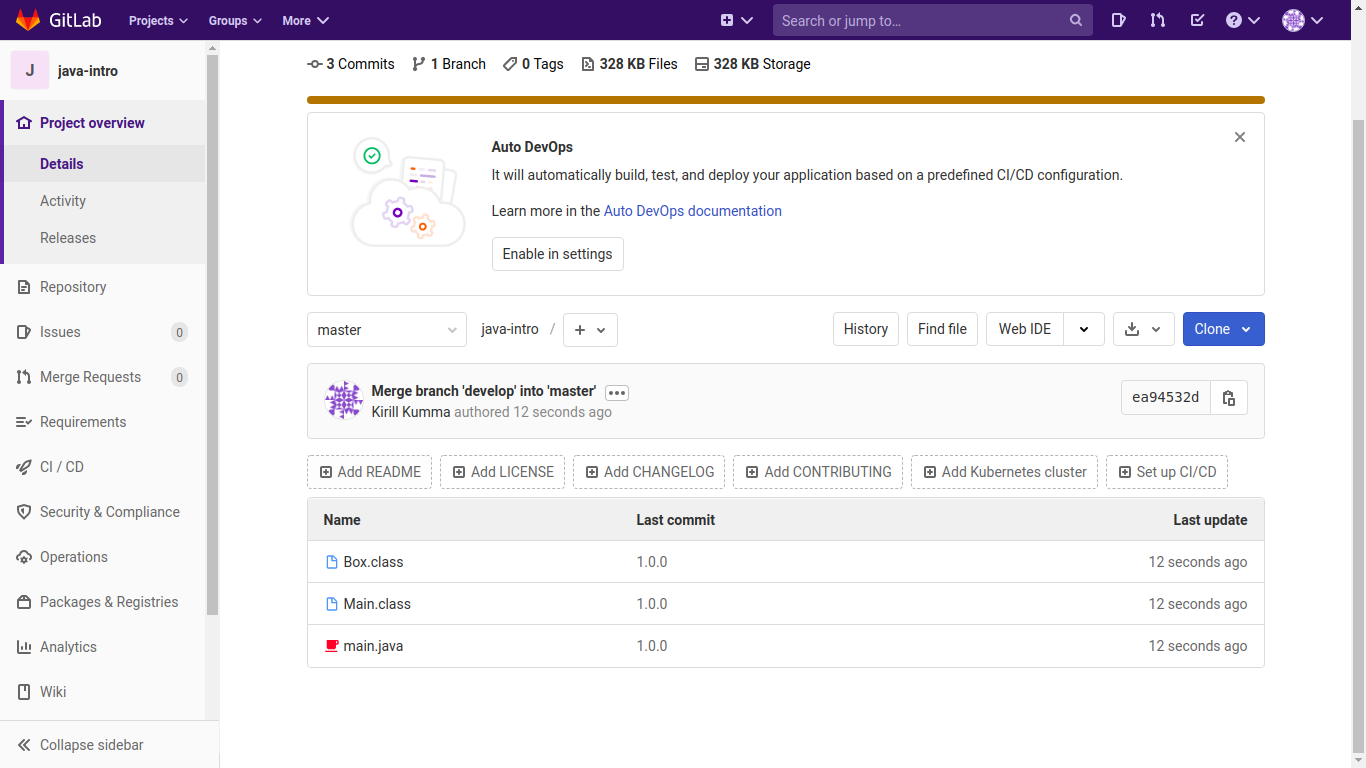
При создании Merge Request’а выбираем режим Squash, а также удаляем нашу develop ветку, т.к. она нам пока что не понадобится.



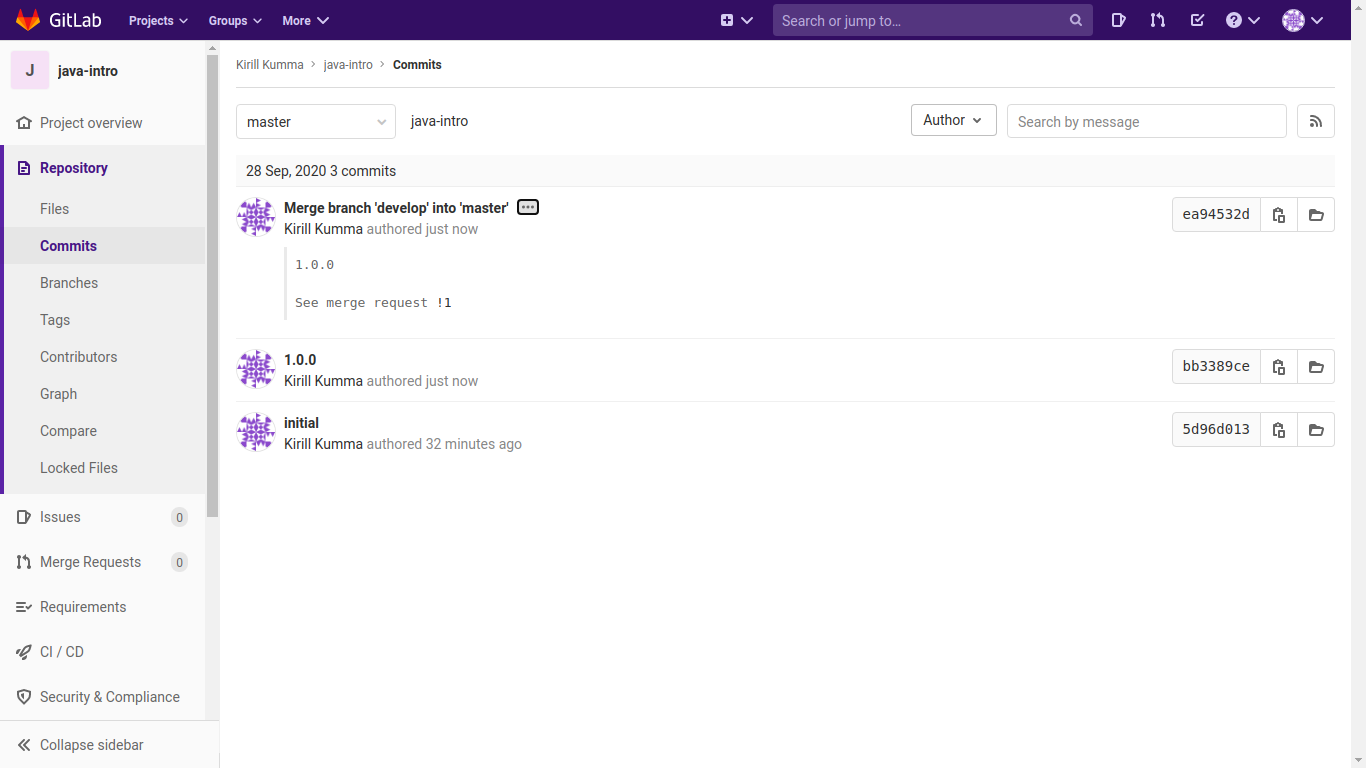
Мы создали Merge Request и теперь можем произвести слияние, нажав на кнопку “Merge”.



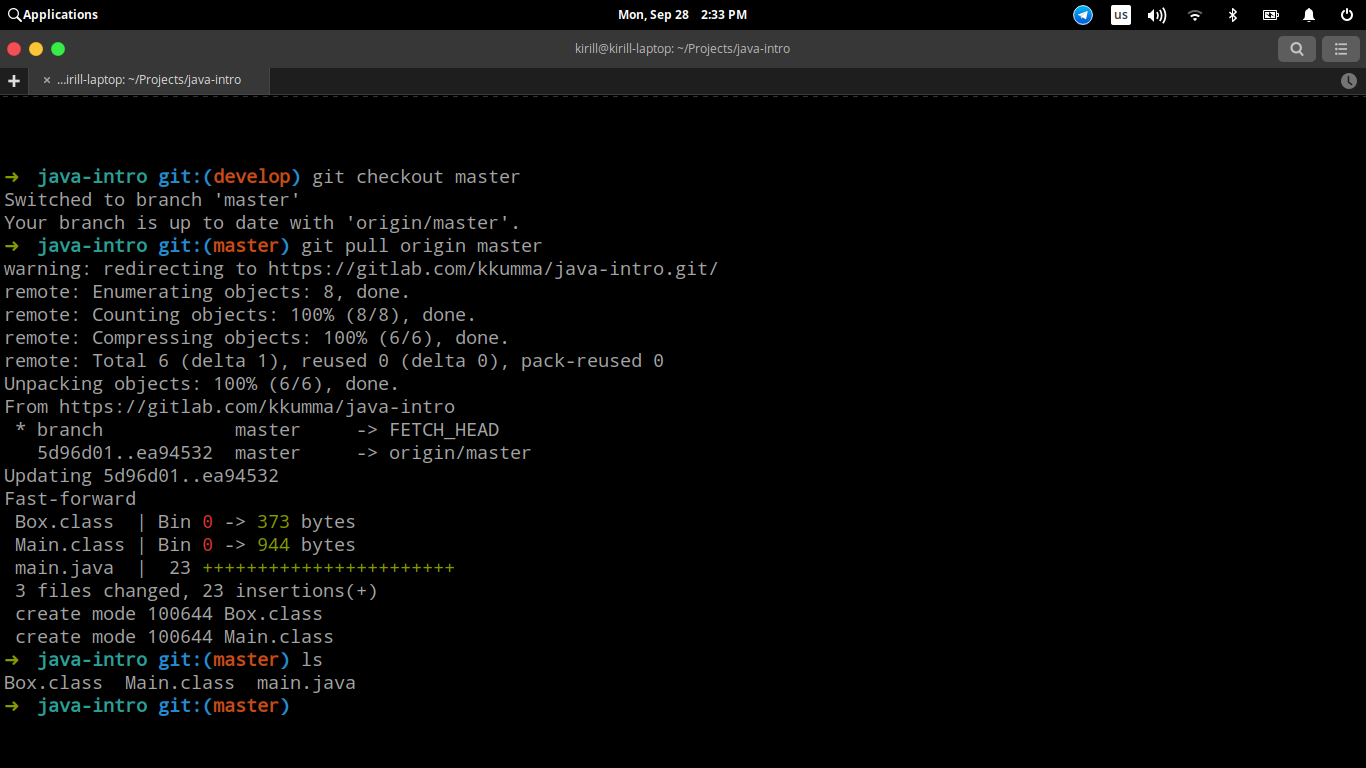
Теперь ветка master имеет все файлы нашей программы.



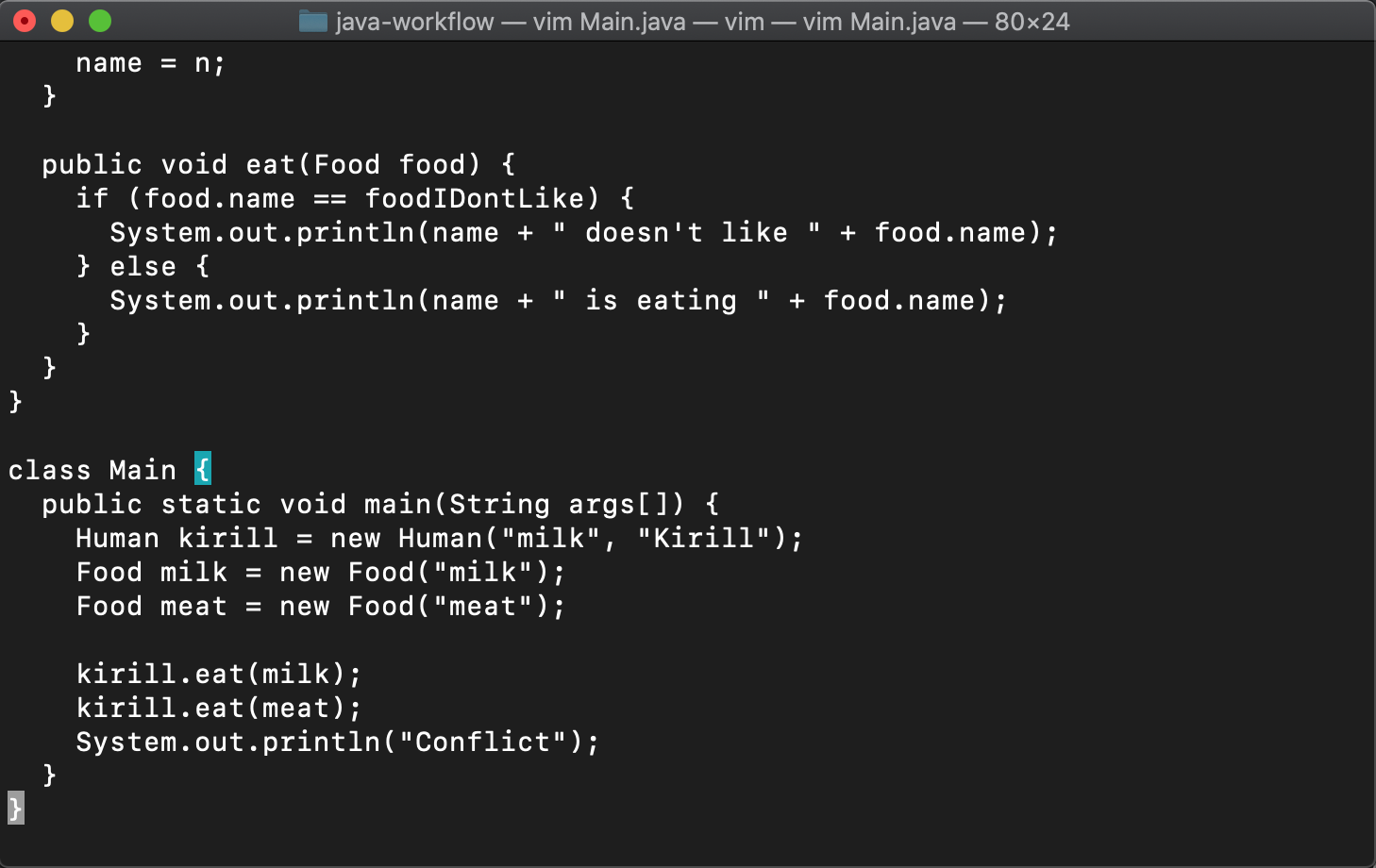
Но история коммитов не загрязнена коммитами ветки develop.

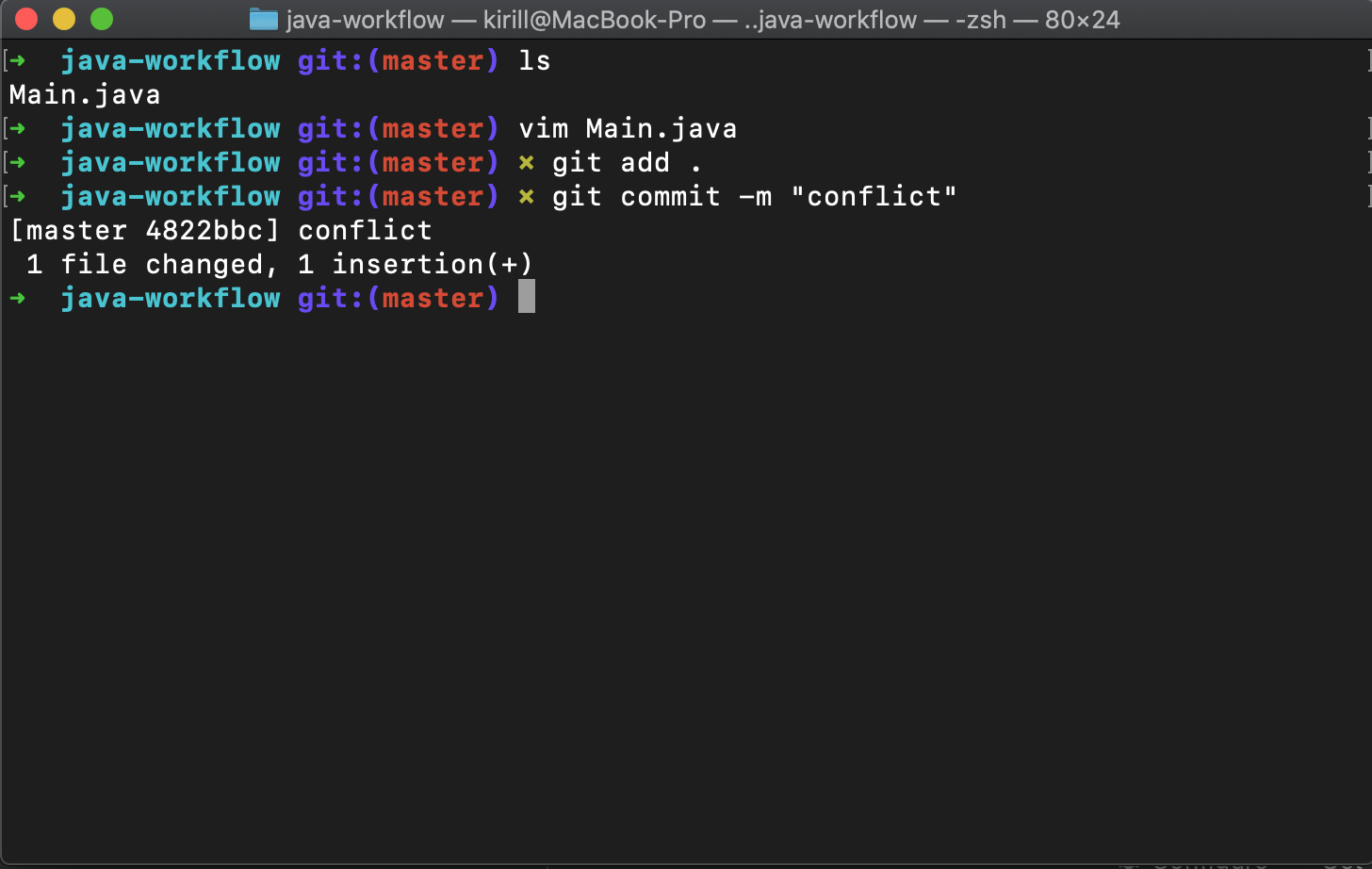


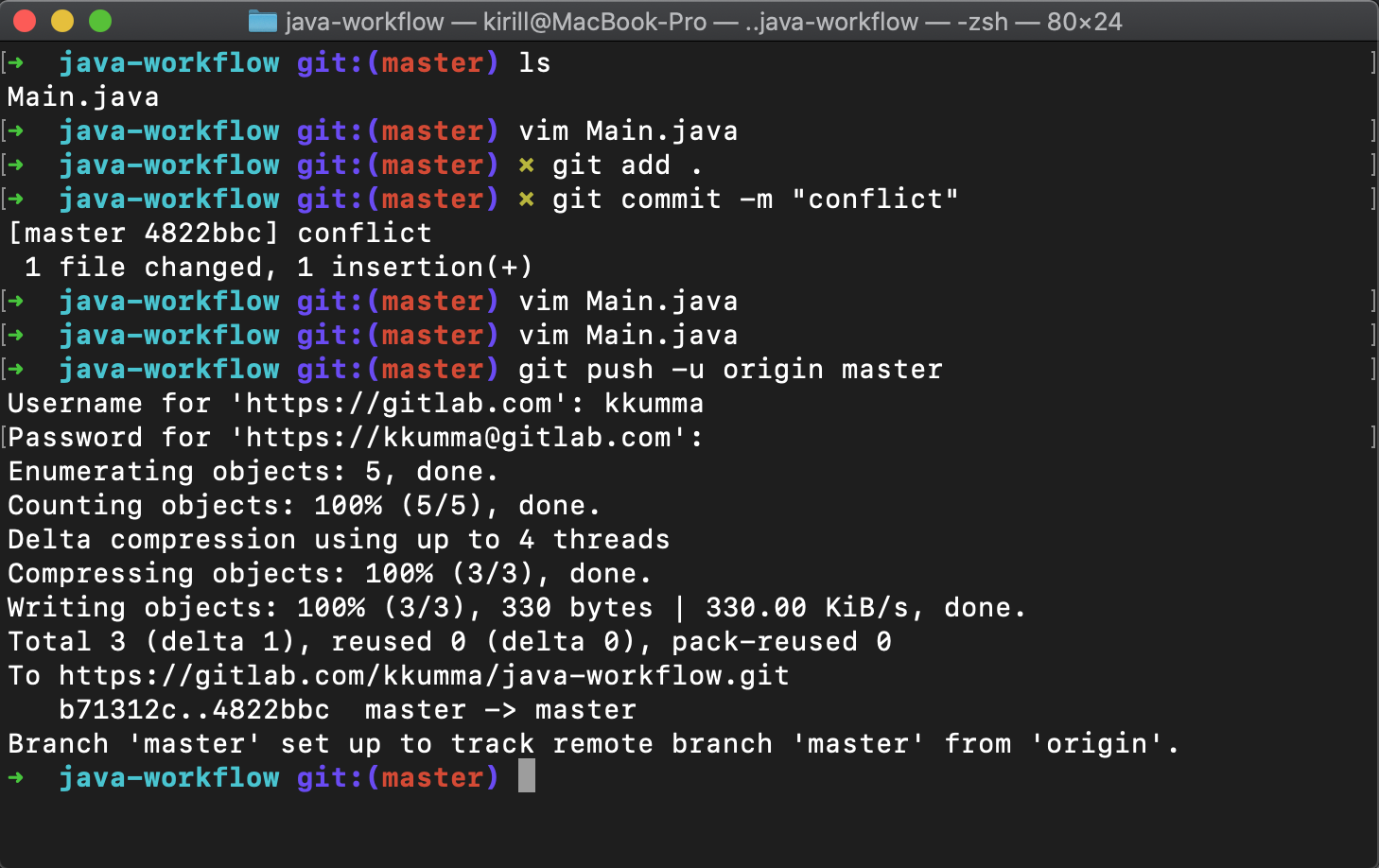
Теперь с помощью команды “git pull” мы можем получить изменения master ветки. Как мы видим все файлы нашей программы присутствуют.

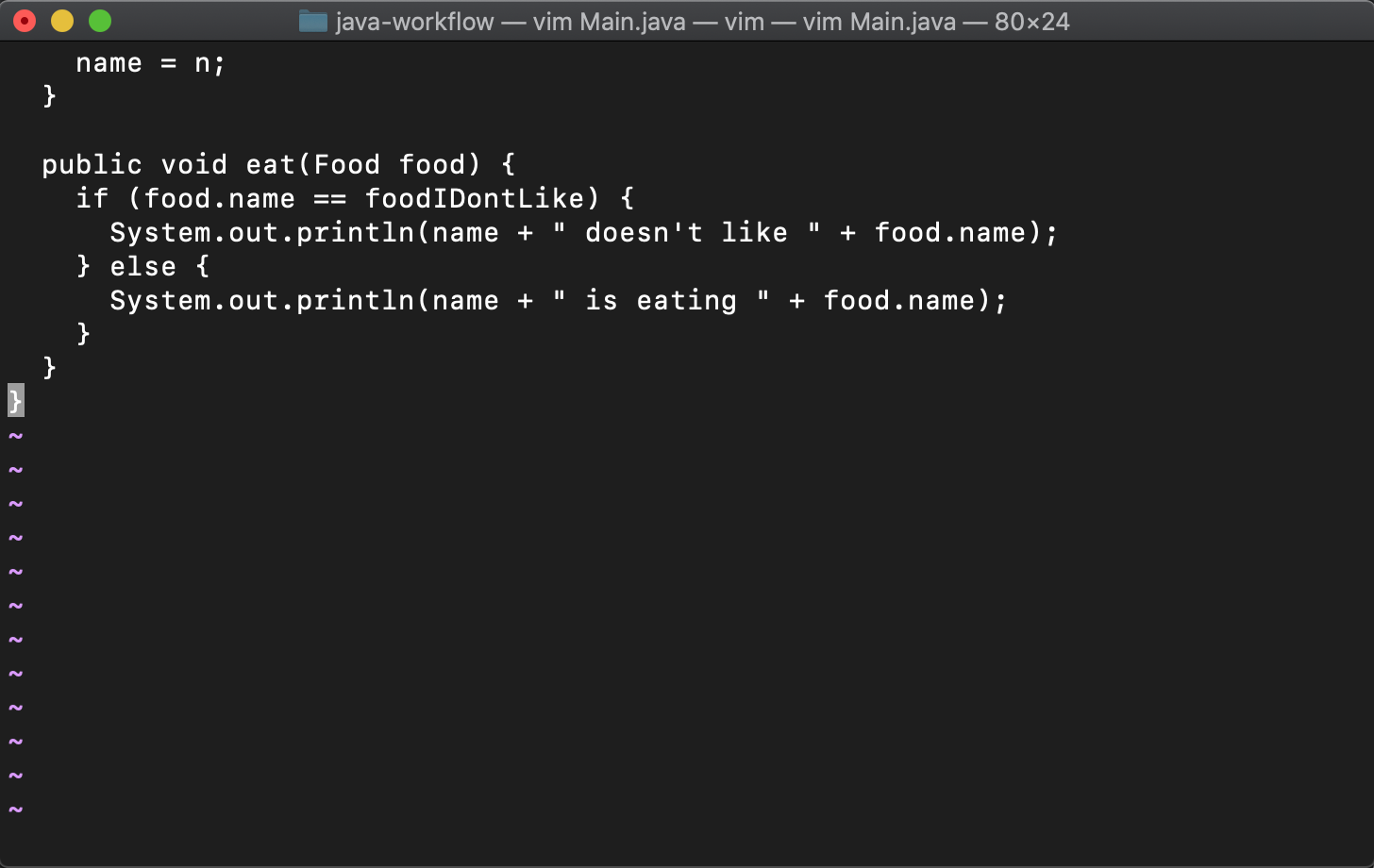


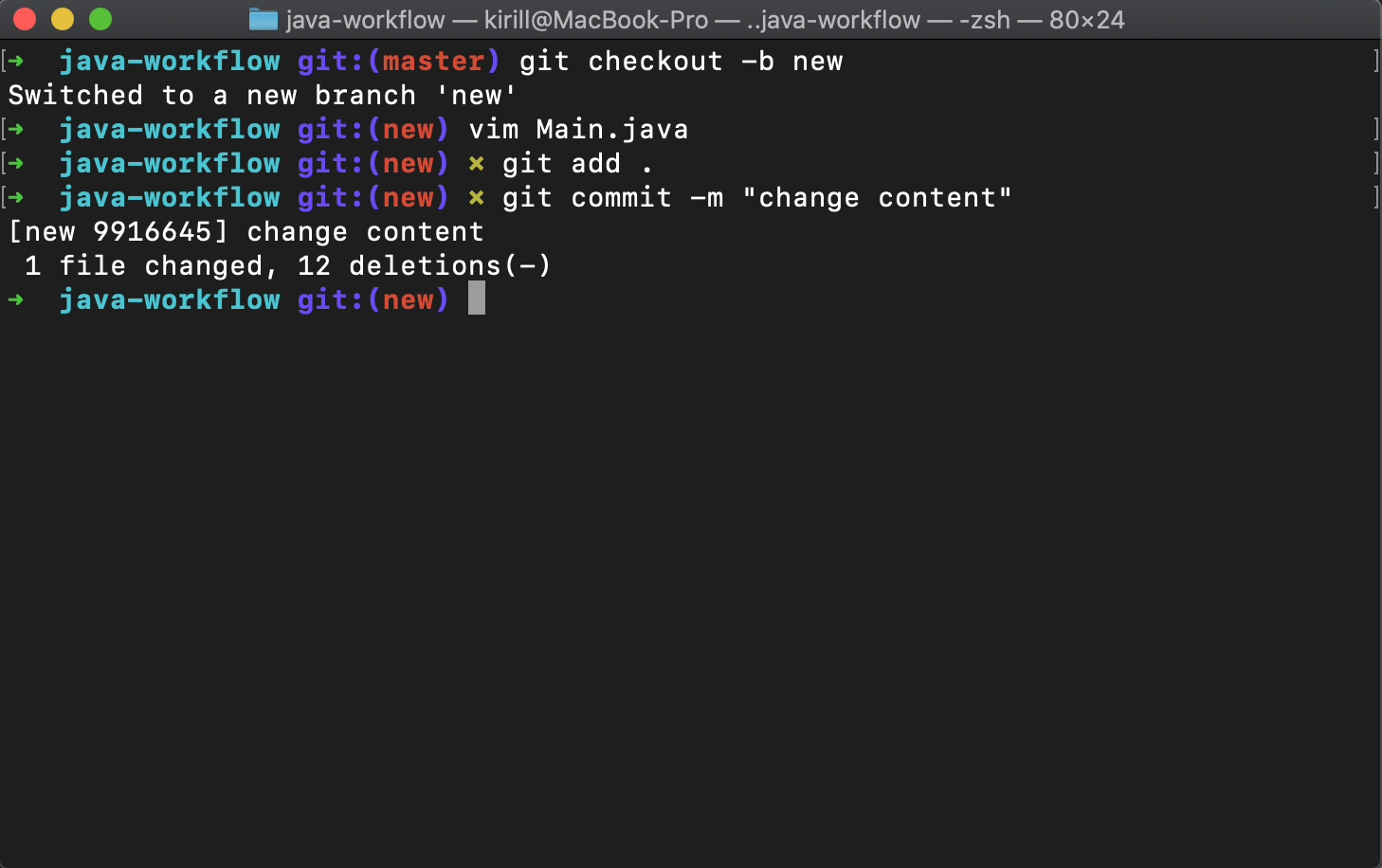
Merge с конфликтом

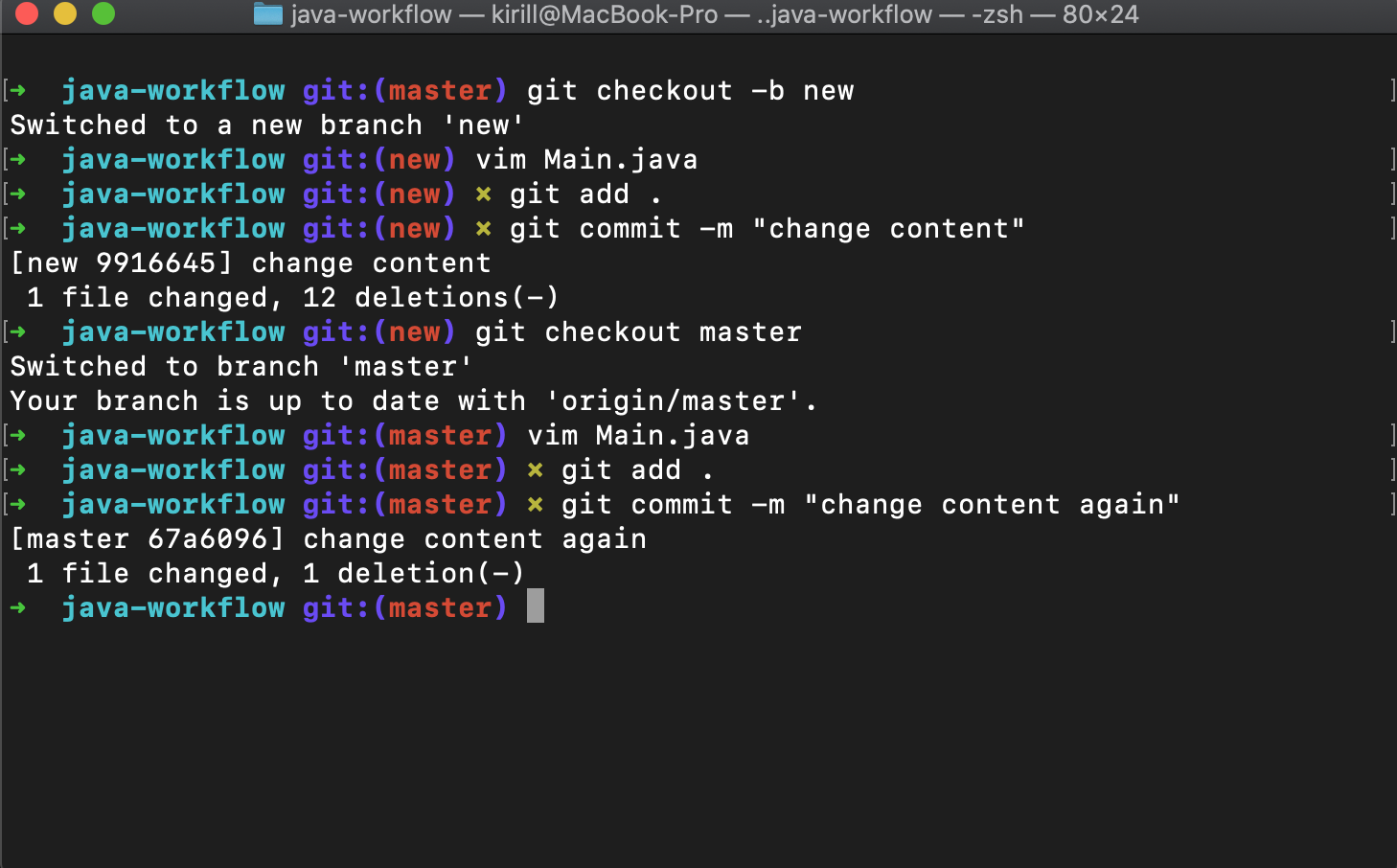


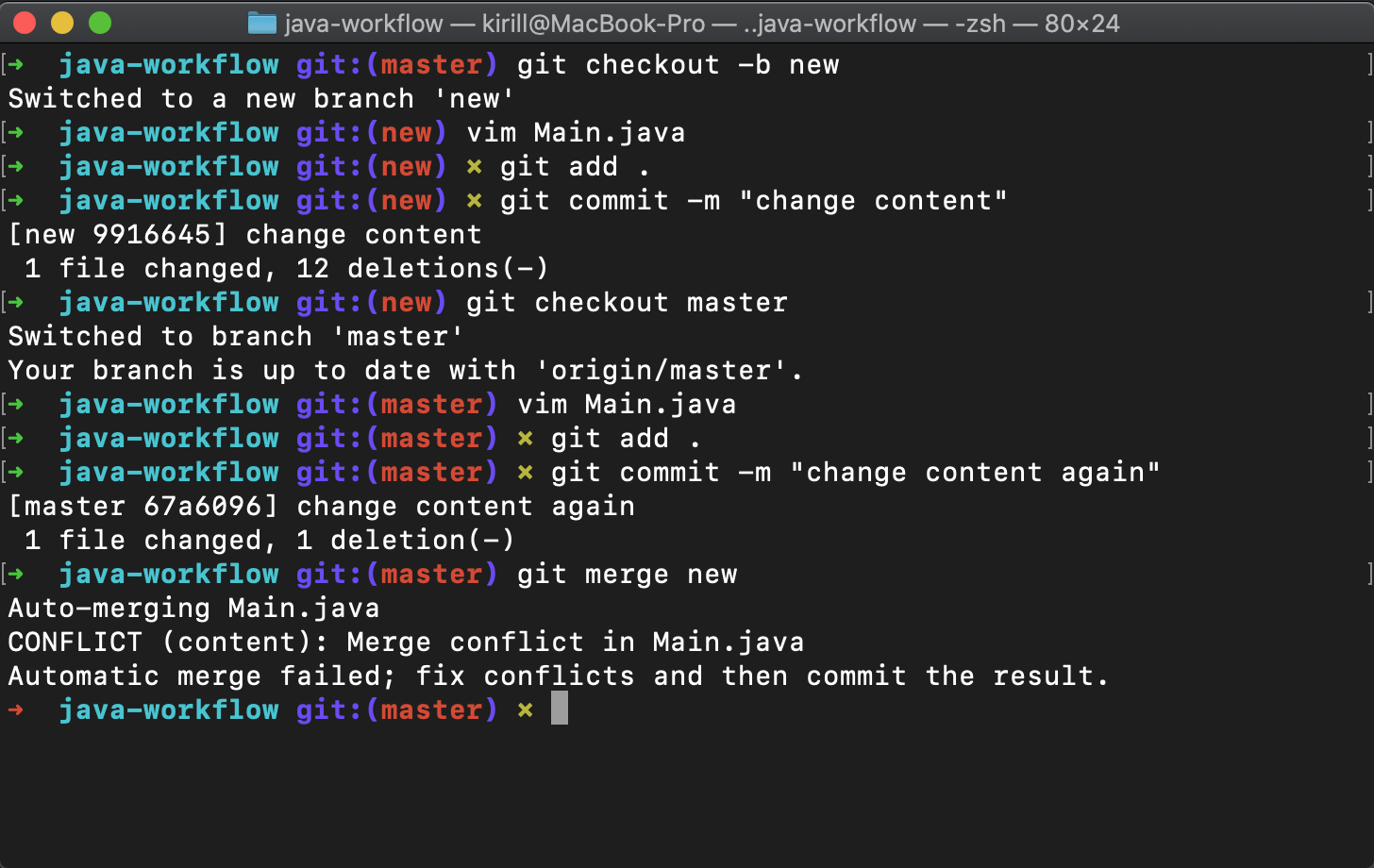


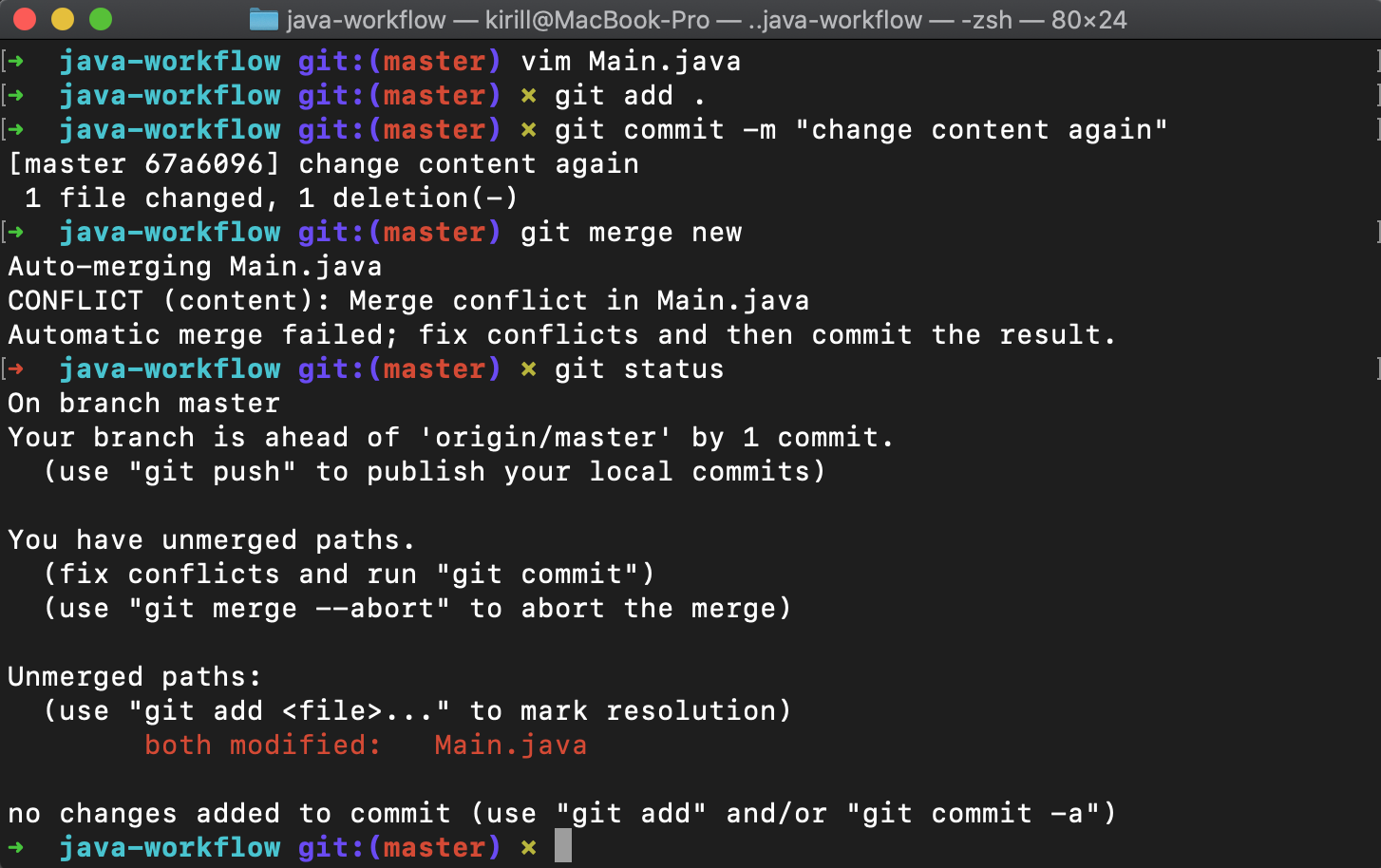
















**Вывод**: выполнив данную лабораторную работу, мы написали простую программу, на примере которой изучили базовые функции системы контроля версии Git.

**Ссылка на репозиторий**: <https://gitlab.com/kkumma/java-intro>

**Список использованных источников**

1) Документация Git: <https://git-scm.com/doc>

2) Документация GitLab: <https://docs.gitlab.com/>

3) Java. Полное руководство. Герберт Шилдт