**ДЕМОНСТРАЦІЯ РОБОТИ**

Було створено репозиторій

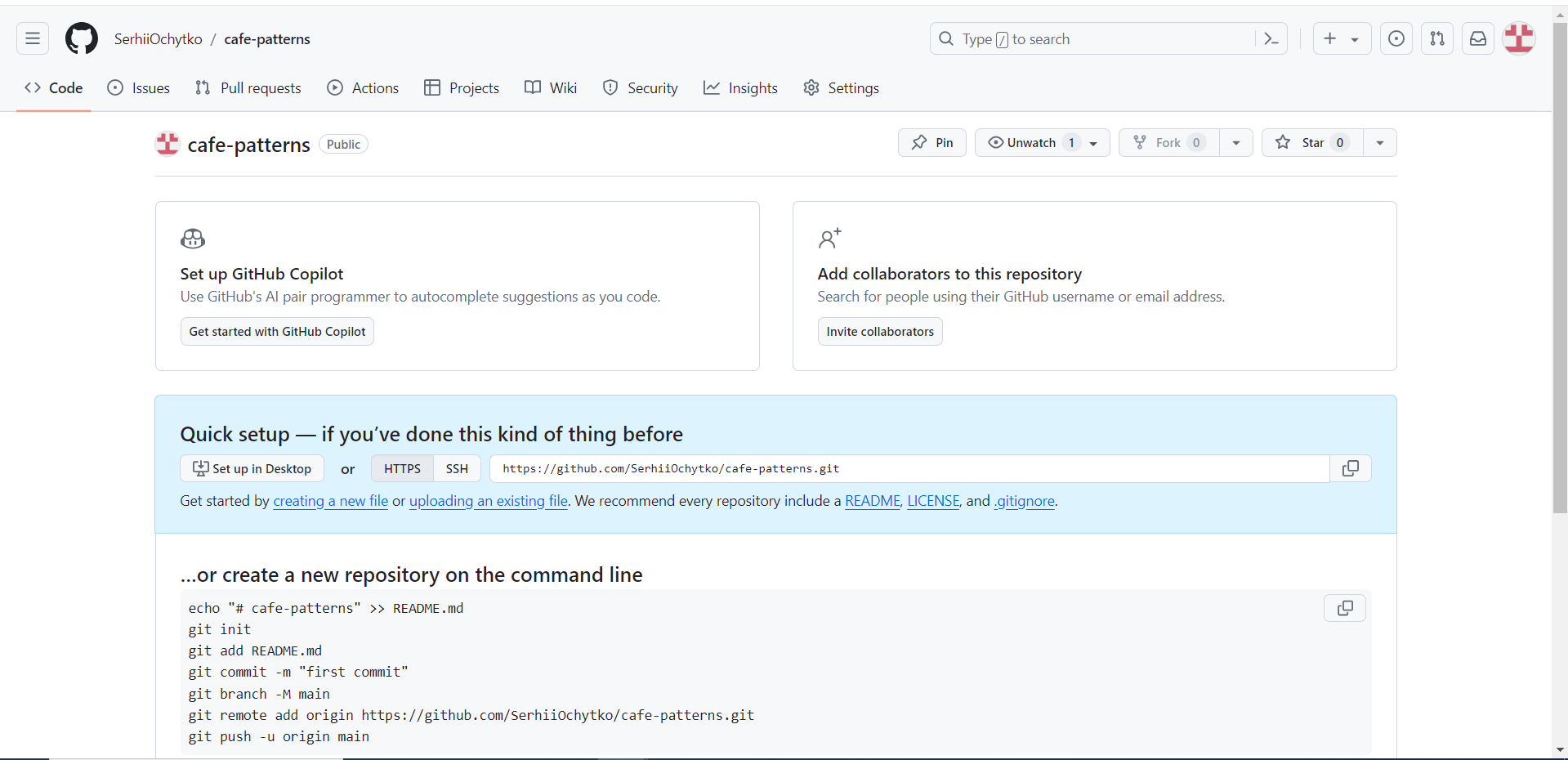


Рис. 1 Репозиторій café-patterns

Клонував репозиторій на свій локальний ПК

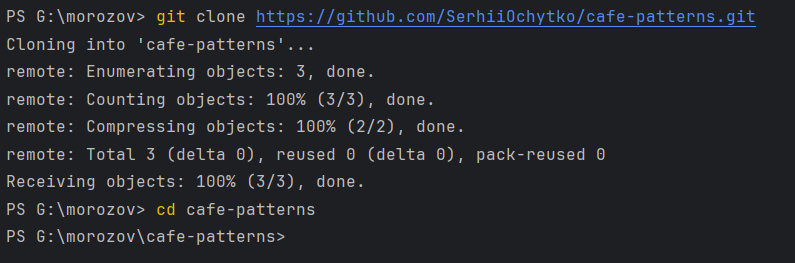


Рис. 2 Клонування

Створив гілку feature-1

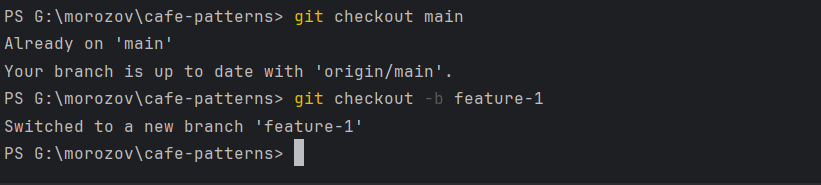


Рис. 3 Створення гілки feature-1

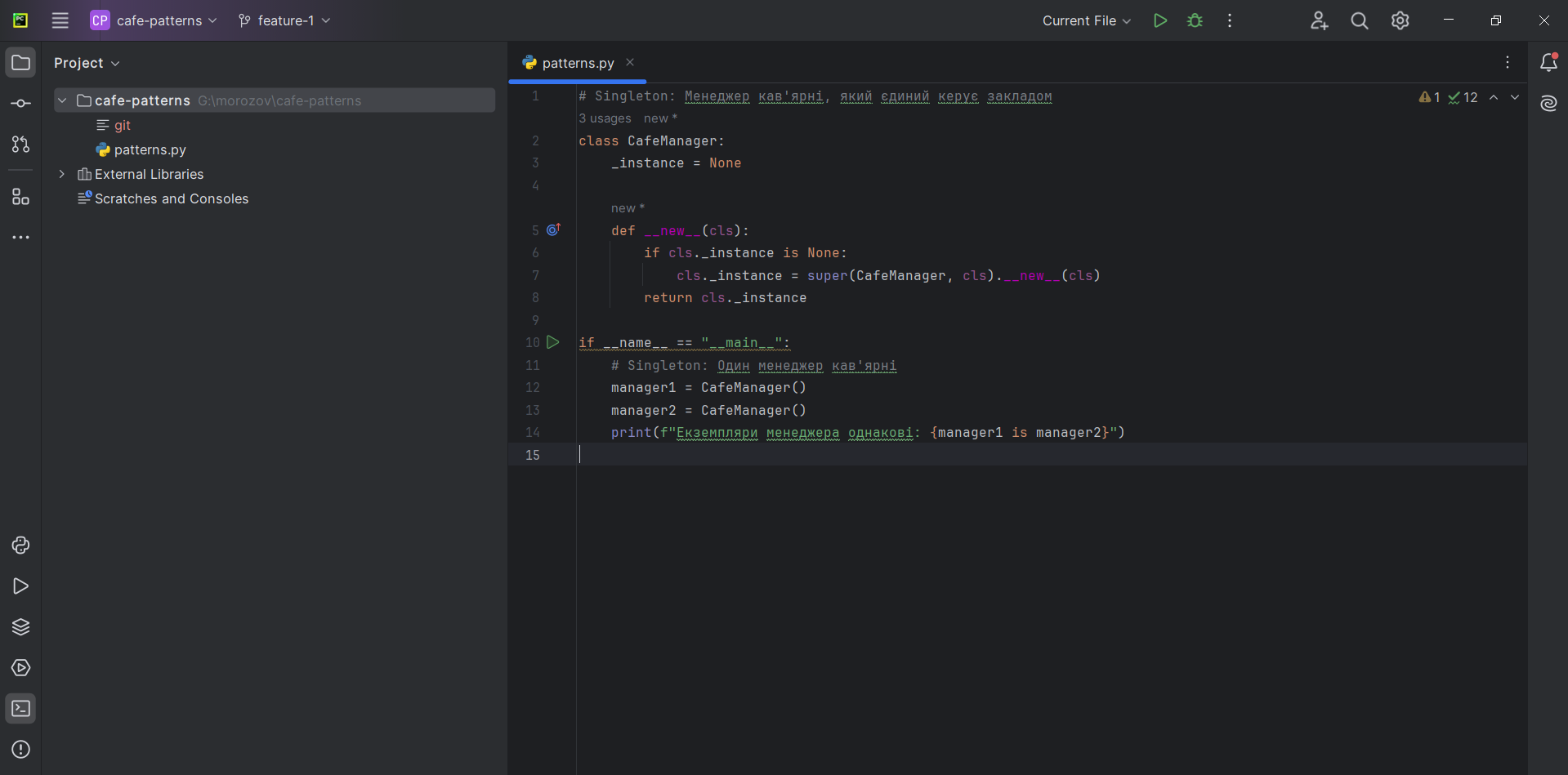


Рис. 4 Реалiзацiя патерна Singleton

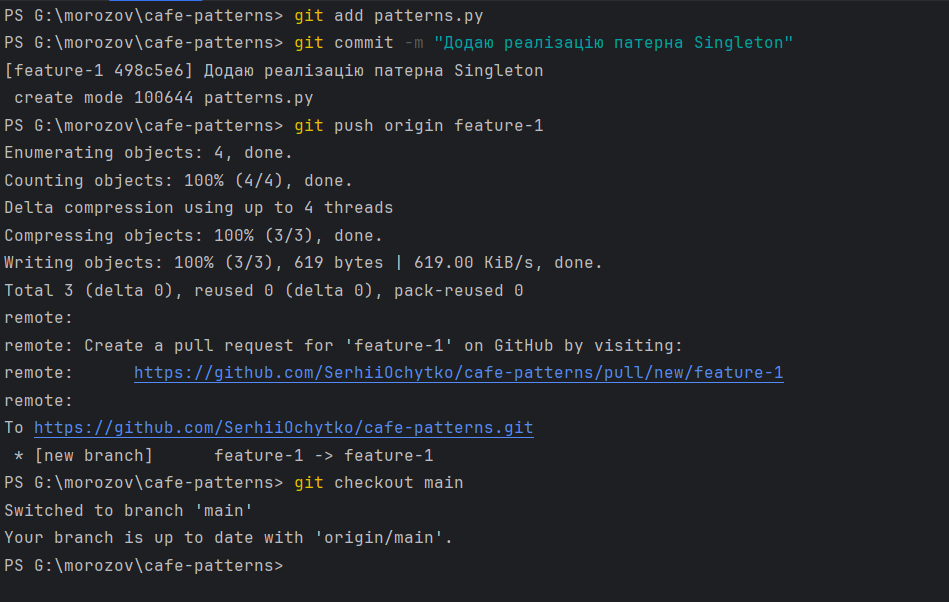


Рис. 5 Закомітив та запушив зміни

Далі виконую аналогічні дії для патернів Factory Method, Observer, Strategy та Decorator.Відповідно до патерна додавався новий код, комітився та пушився у нову гілку.

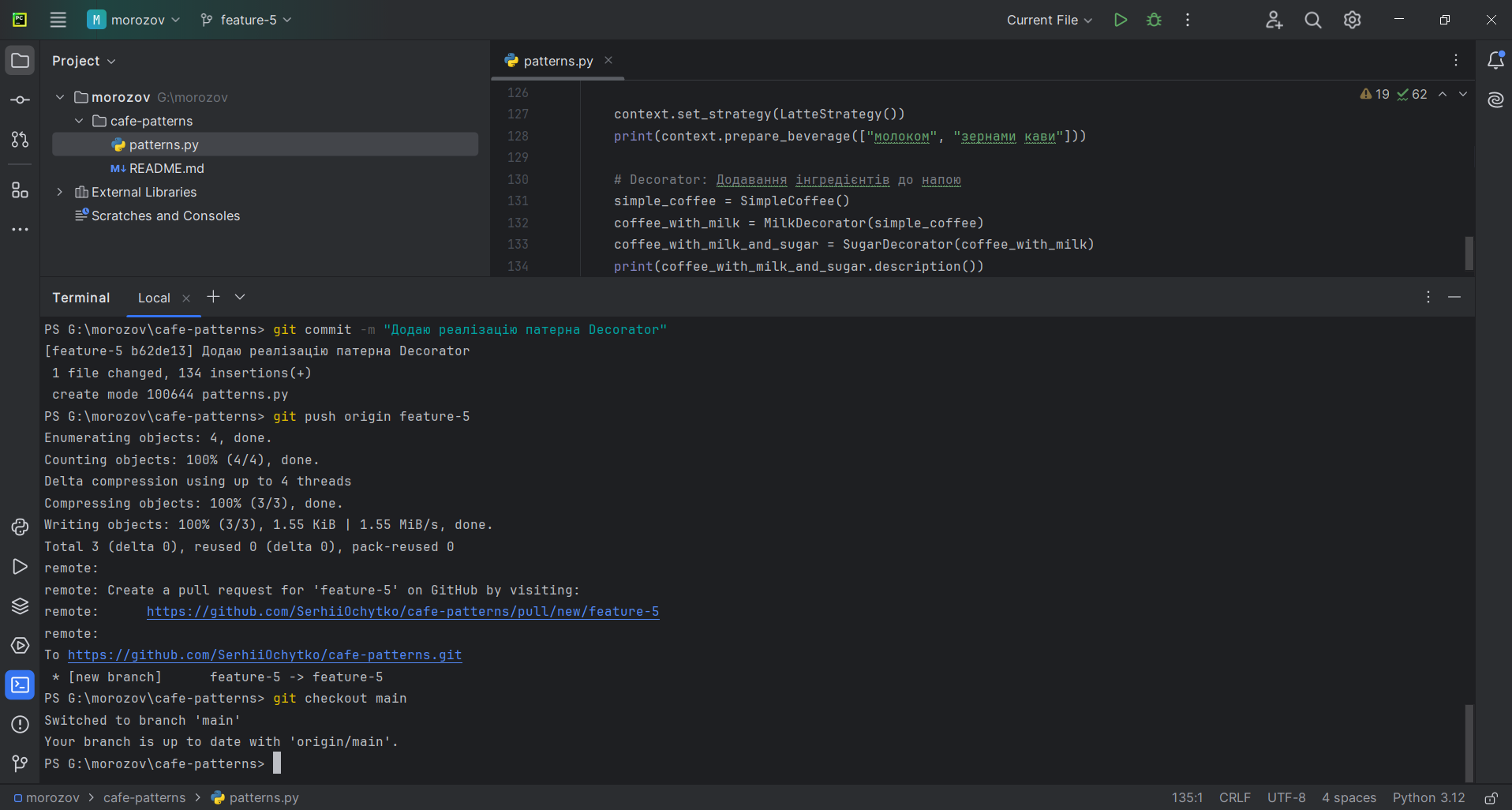


Рис. 6 Коміт і пуш гілки feature-5

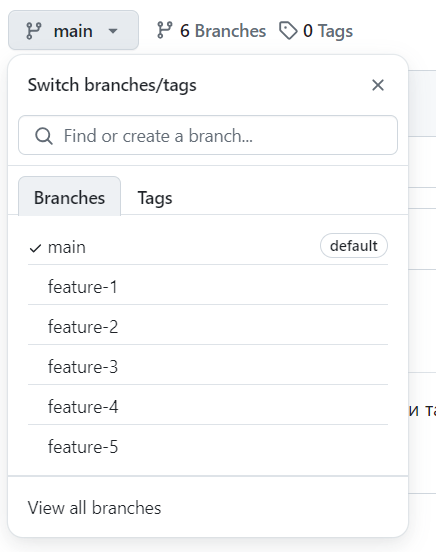


Рис. 7 Перегляд гілок

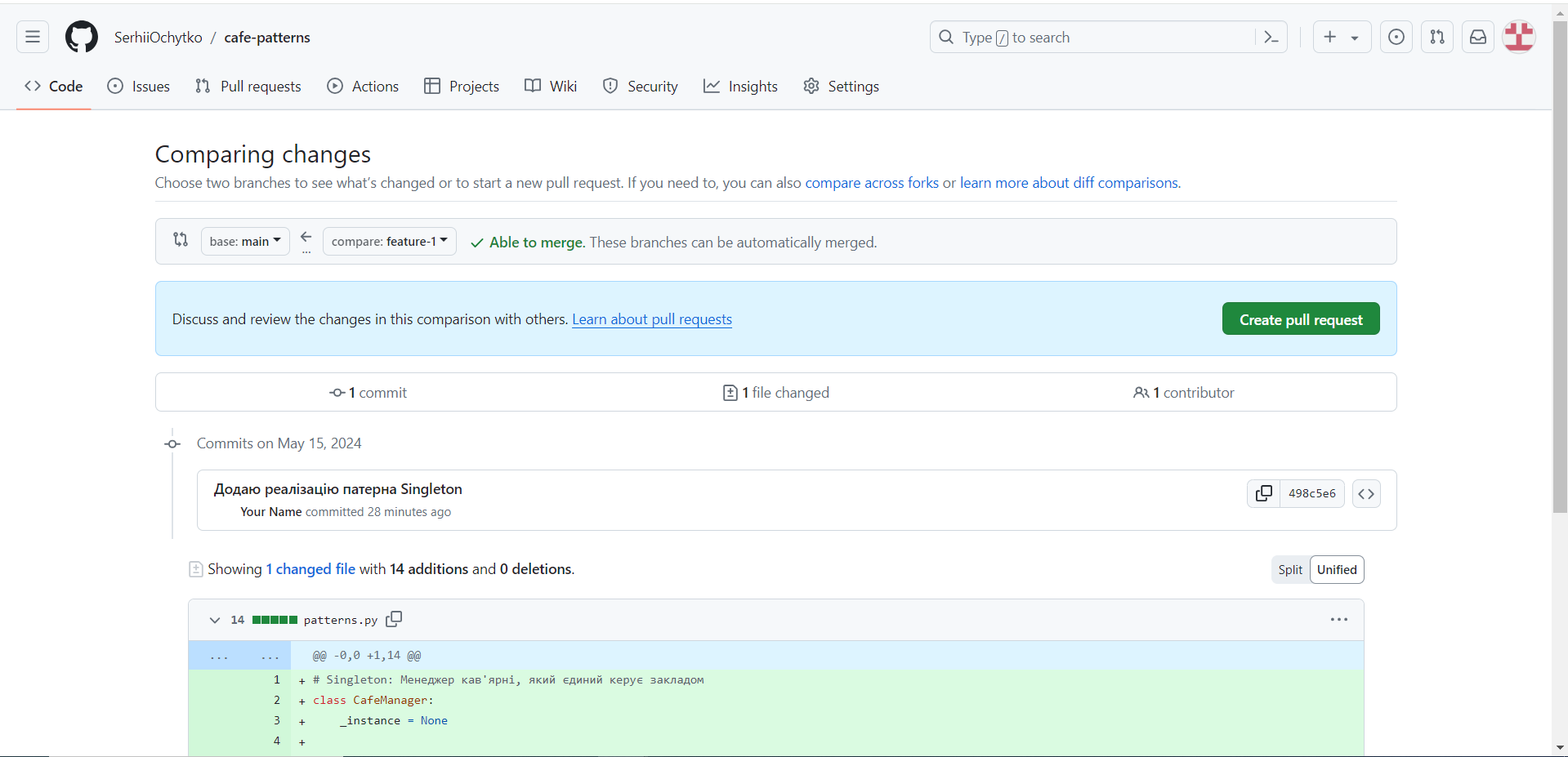


Рис. 8 Виконав pull request

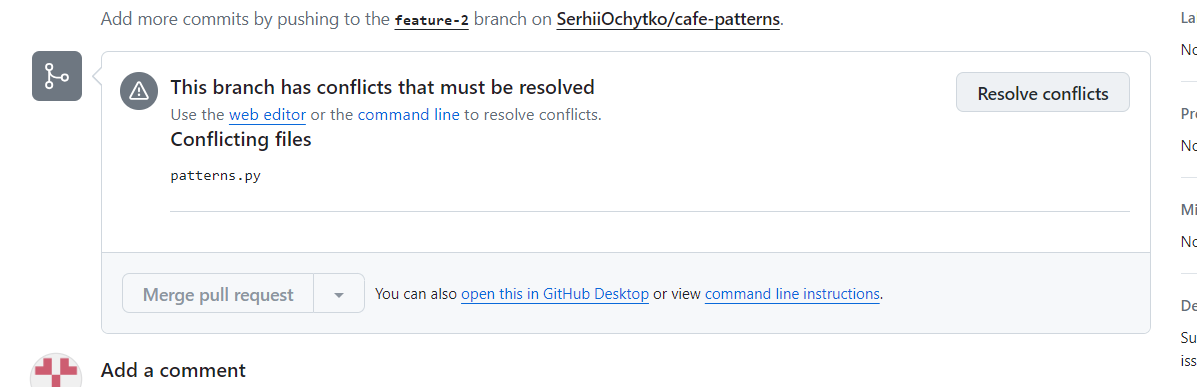


Рис. 9 Відбувся конфлікт під час об’єднання

Під час конфлікта вручну було стерто рядки, які заважали злиттю.

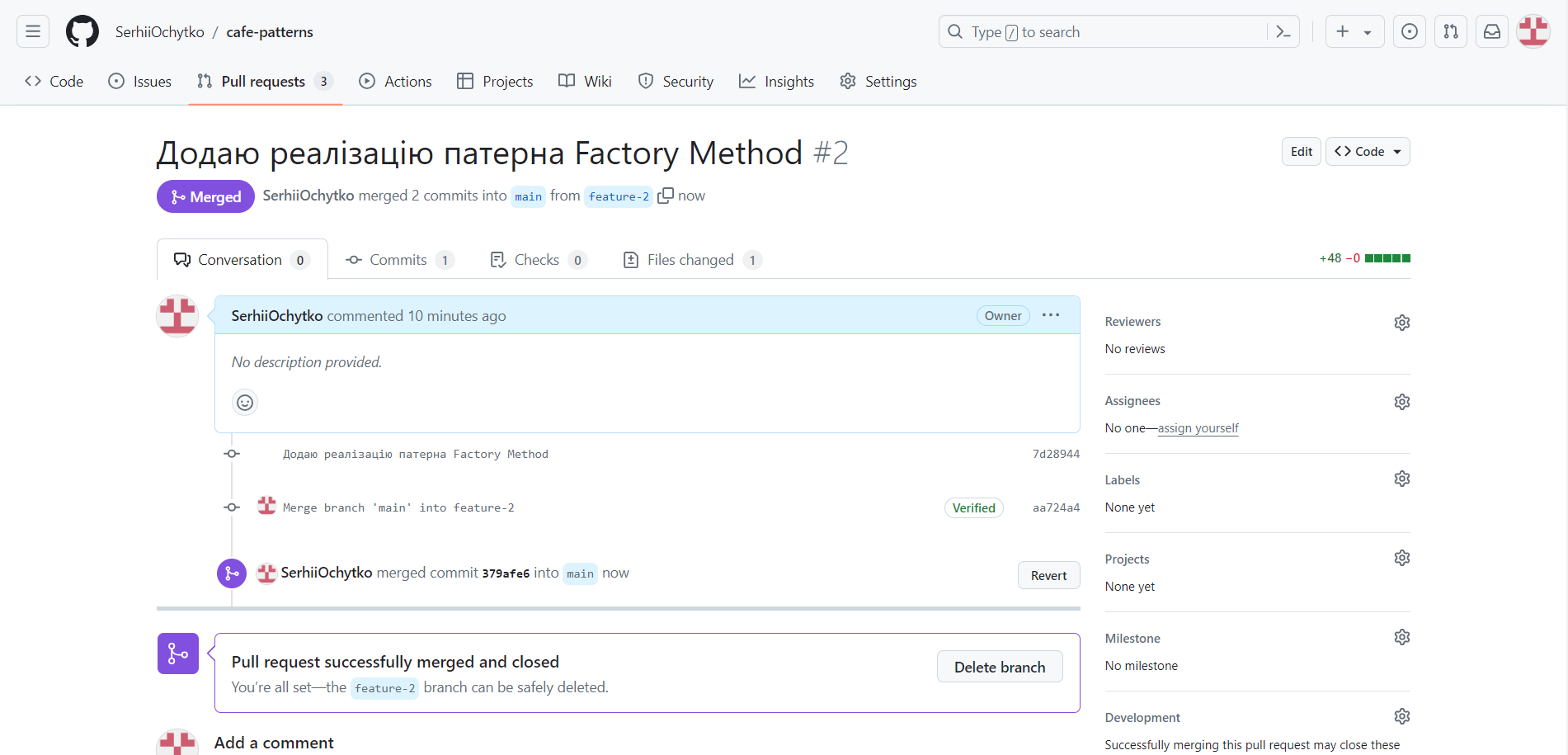


Рис. 10 Успішне злиття

Не потрібні гілки після злиття було видалено.

Після вирішення конфліктів і злиття, ми отримали насутпний код:

# Singleton: Менеджер кав'ярні, який єдиний керує закладом  
class CafeManager:  
 \_instance = None  
  
 def \_\_new\_\_(cls):  
 if cls.\_instance is None:  
 cls.\_instance = super(CafeManager, cls).\_\_new\_\_(cls)  
 return cls.\_instance  
  
# Factory Method: Виробництво різних типів напоїв  
class Beverage:  
 def prepare(self):  
 raise NotImplementedError  
  
class Coffee(Beverage):  
 def prepare(self):  
 return "Приготування кави"  
  
class Tea(Beverage):  
 def prepare(self):  
 return "Приготування чаю"  
  
class BeverageCreator:  
 def create\_beverage(self):  
 raise NotImplementedError  
  
class CoffeeCreator(BeverageCreator):  
 def create\_beverage(self):  
 return Coffee()  
  
class TeaCreator(BeverageCreator):  
 def create\_beverage(self):  
 return Tea()  
  
# Observer: Оповіщення баристи про нові замовлення  
class OrderSubject:  
 def \_\_init\_\_(self):  
 self.\_observers = []  
  
 def attach(self, observer):  
 self.\_observers.append(observer)  
  
 def notify(self, message):  
 for observer in self.\_observers:  
 observer.update(message)  
  
class Barista:  
 def update(self, message):  
 raise NotImplementedError  
  
class ConcreteBarista(Barista):  
 def update(self, message):  
 print(f"Бариста отримав замовлення: {message}")  
  
# Strategy: Різні стратегії приготування напоїв  
class PreparationStrategy:  
 def prepare(self, ingredients):  
 raise NotImplementedError  
  
class EspressoStrategy(PreparationStrategy):  
 def prepare(self, ingredients):  
 return f"Приготування еспресо з {', '.join(ingredients)}"  
  
class LatteStrategy(PreparationStrategy):  
 def prepare(self, ingredients):  
 return f"Приготування латте з {', '.join(ingredients)}"  
  
class PreparationContext:  
 def \_\_init\_\_(self, strategy: PreparationStrategy):  
 self.\_strategy = strategy  
  
 def set\_strategy(self, strategy: PreparationStrategy):  
 self.\_strategy = strategy  
  
 def prepare\_beverage(self, ingredients):  
 return self.\_strategy.prepare(ingredients)  
  
# Decorator: Додавання інгредієнтів до напою  
class BeverageComponent:  
 def description(self):  
 raise NotImplementedError  
  
class SimpleCoffee(BeverageComponent):  
 def description(self):  
 return "Проста кава"  
  
class BeverageDecorator(BeverageComponent):  
 def \_\_init\_\_(self, component: BeverageComponent):  
 self.\_component = component  
  
 def description(self):  
 return self.\_component.description()  
  
class MilkDecorator(BeverageDecorator):  
 def description(self):  
 return f"{self.\_component.description()} + Молоко"  
  
class SugarDecorator(BeverageDecorator):  
 def description(self):  
 return f"{self.\_component.description()} + Цукор"  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Singleton: Один менеджер кав'ярні  
 manager1 = CafeManager()  
 manager2 = CafeManager()  
 print(f"Екземпляри менеджера однакові: {manager1 is manager2}")  
  
 # Factory Method: Створення напоїв  
 coffee\_creator = CoffeeCreator()  
 coffee = coffee\_creator.create\_beverage()  
 print(coffee.prepare())  
  
 tea\_creator = TeaCreator()  
 tea = tea\_creator.create\_beverage()  
 print(tea.prepare())  
  
 # Observer: Оповіщення баристи про замовлення  
 order\_subject = OrderSubject()  
 barista = ConcreteBarista()  
 order\_subject.attach(barista)  
 order\_subject.notify("Нове замовлення: Капучино")  
  
 # Strategy: Різні стратегії приготування напоїв  
 context = PreparationContext(EspressoStrategy())  
 print(context.prepare\_beverage(["водою", "зернами кави"]))  
  
 context.set\_strategy(LatteStrategy())  
 print(context.prepare\_beverage(["молоком", "зернами кави"]))  
  
 # Decorator: Додавання інгредієнтів до напою  
 simple\_coffee = SimpleCoffee()  
 coffee\_with\_milk = MilkDecorator(simple\_coffee)  
 coffee\_with\_milk\_and\_sugar = SugarDecorator(coffee\_with\_milk)  
 print(coffee\_with\_milk\_and\_sugar.description())

Під час роботи було продемонстровано створення репозиторію, створення гілок, перехід між ними, вирішення конфліктів. В фінальній версії patterns.py, мии отримали код з використанням 5 патернів.