Звіт

З лабораторної роботи №1

Студента групи МІТ-31

Добровольського Арсенія Михайловича

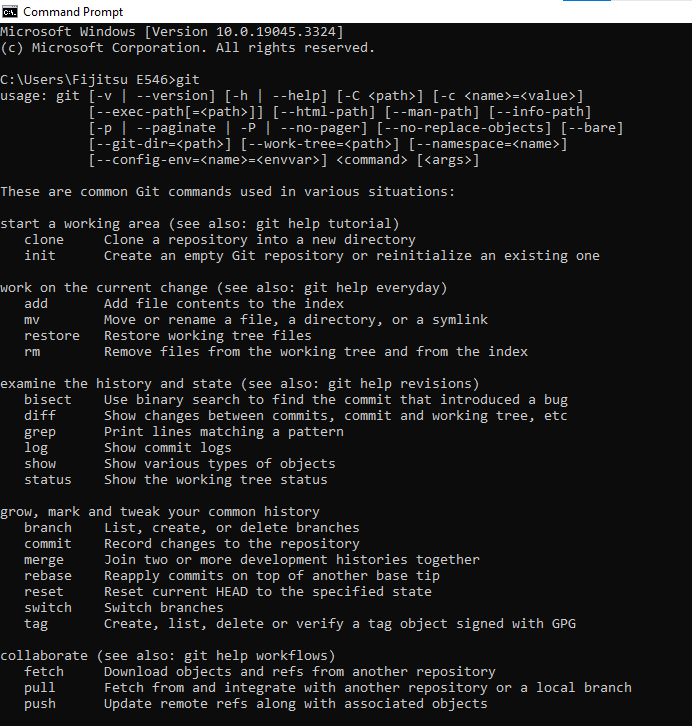
Частина №1

**Тема роботи:** Основи Git

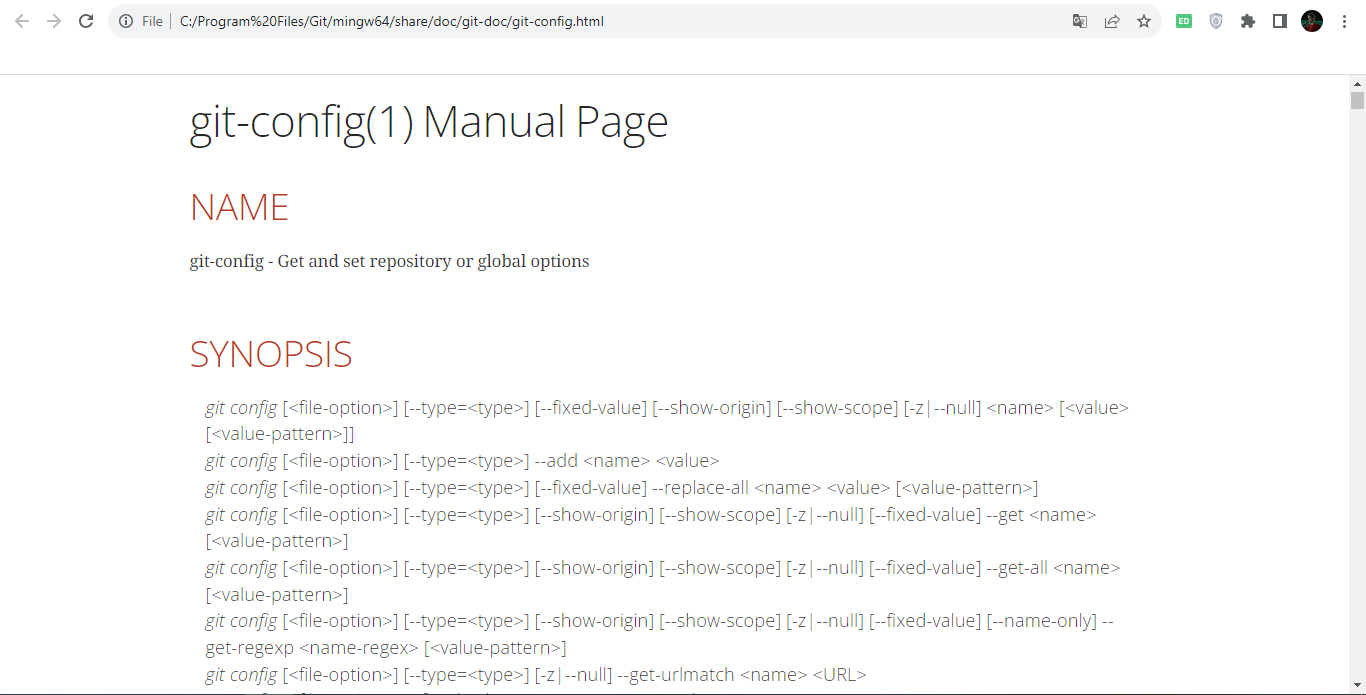
**Мета роботи:** ознайомитися з призначенням та функціональними можливостями системи контролю версій Git та навчитися виконувати базові операції в ній.

**Хід виконання роботи**

Виклик команди git:

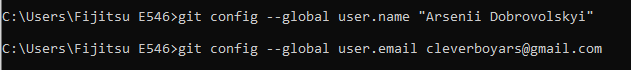


Виклик довідки для команди config (команди git config --help   
та git help config):

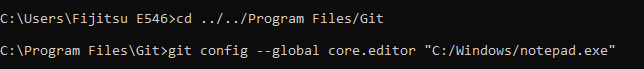


В результаті виконання обох команд у вікні браузера відображається довідкова сторінка по команді config, яка містить детальний опис цієї команди, можливі параметри, приклади її використання тощо.

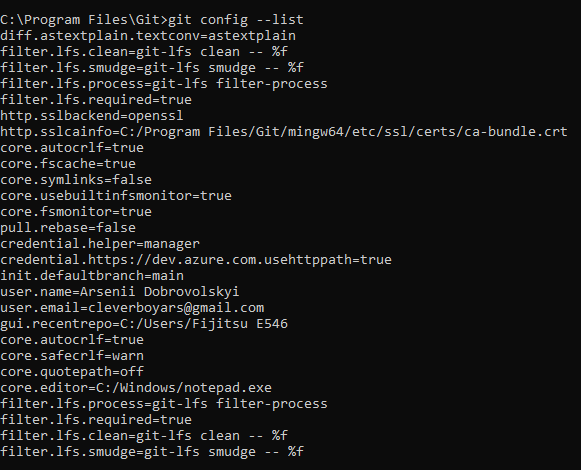
Налаштування імені користувача та адреси електронної пошти у глобальному файлі конфігурації:



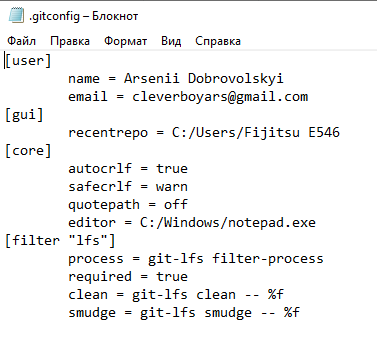
Налаштування текстового редактора за замовчуванням:



Перевірка налаштувань Git:

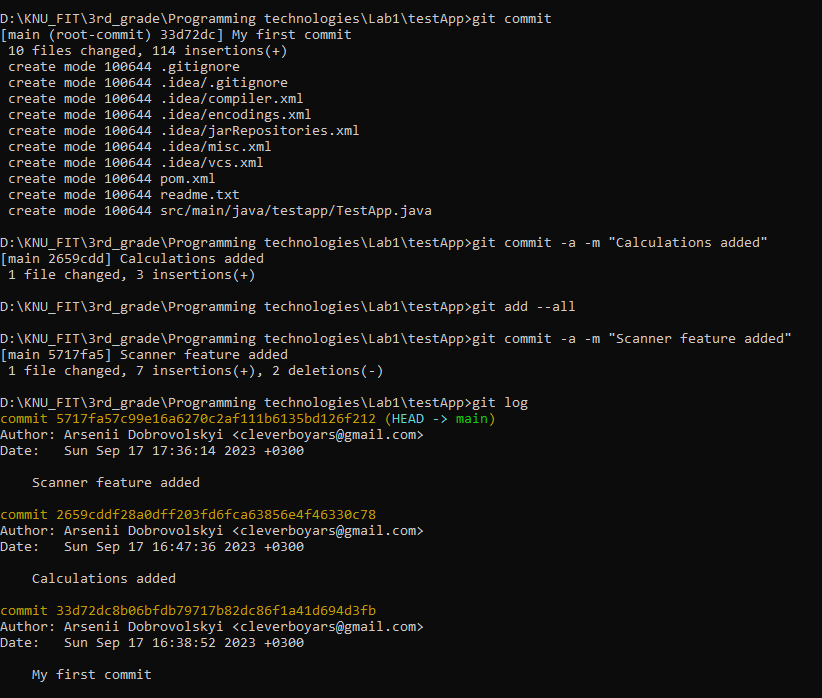


Основний конфігураційний файл має вигляд:



Дані, записані у цьому файлі, свідчать про те, що всі налаштування виконано правильно.

Створення комітів проєкту testApp та перегляд історії комітів наведено на скріншоті нижче:



До основних параметрів команди git log належать:

* -p (показує різницю, привнесену кожним комітом)
* --stat (коротка статистика змін, що включає кількість внесених змін, вставок та вилучень файлів для кожного коміту)
* --oneline (виводить кожен коміт в одному рядку, представляючи ідентифікатор коміту (SHA-1 хеш) і його короткий опис)
* --graph (виводить графічне представлення гілок і злиттів у історії комітів)
* --author="Ім'я автора" (фільтрує коміти за автором)
* -n (n – довільне число) – виводить вказану кількість комітів
* --grep="текст пошуку" (фільтрує коміти за текстом повідомлення коміту)

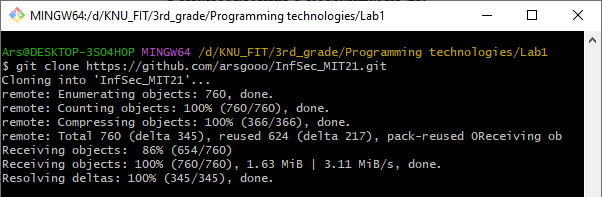
**Завдання для самостійної роботи**

Для виконання команд, що розглянуті далі, за основу взято репозиторій, створений в рамках вивчення минулорічної дисципліни.

Ознайомлення з командами git

1. git clone

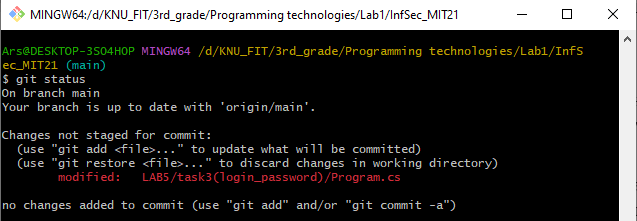
Ця команда використовується для створення копії віддаленого репозиторію Git на локальному комп'ютері.



1. git status

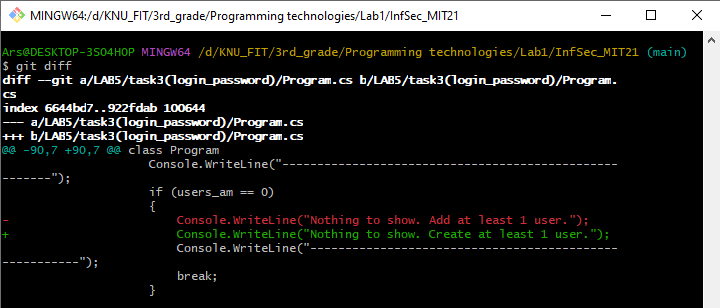
Надає інформацію про поточний стан локального репозиторію. Наприклад, вона відображає файли які було змінено, але не проіндексовано, файли, які вже містяться в індексі і очікують на потрапляння в коміт тощо.

Для того, щоб побачити роботу команди наяву, зроблю зміни в одному з файлів свого репозиторію. Таким чином команда git status виведе такий результат:



1. git diff

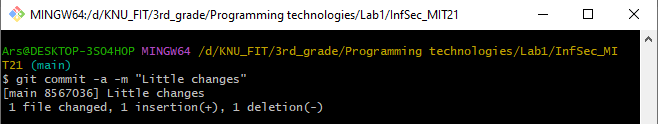
Виявляє та відображає різницю між двома будь-якими станами репозиторію. Зокрема, за допомогою цієї команди можна побачити різницю між робочою директорією та індексом, між індексом та останнім комітом (git diff --staged), між двома комітами або навіть гілками. Фактично git diff деталізує інформацію, яку відображає команда git status, тобто показує не лише файли, які було змінено, а й конкретні зміни:



В той час як команда git status відобразила лише те, що був змінений файл Program.cs, git diff показує, яких саме змін зазнав цей файл.

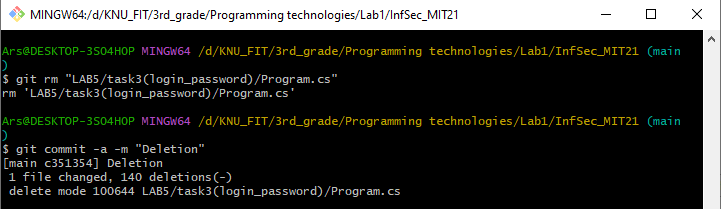
1. git commit

Створює новий коміт, тобто зберігає зміни, зроблені у локальному репозиторії. Перед цим слід проіндексувати файли, виконавши команду git add . або можна просто додати параметр -a до команди git commit. Також стане в пригоді параметр -m, що дозволяє додати коментар до коміту без відкриття текстового редактора.



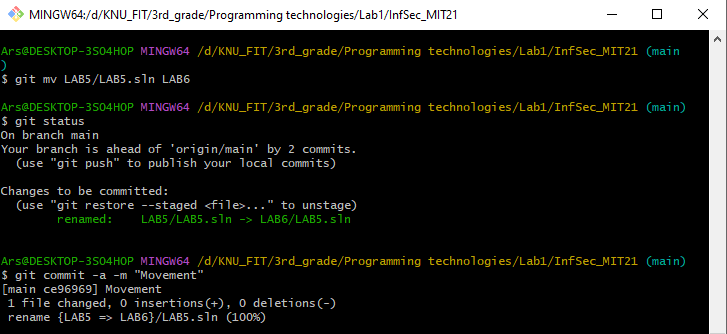
1. git rm

Видаляє файли або папки з індексу та робочої директорії. Для демонстрації роботи цієї команди видалю файл, в який було внесено зміни у попередніх підпунктах та виконаю команду git commit для збереження змін:



1. git mv

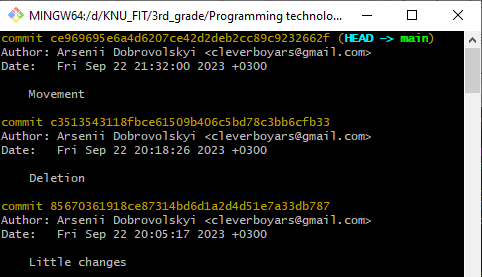
Переміщає або перейменовує файли в репозиторії. Виконання цієї команди також включає видалення вихідного файлу (git rm) та додавання в індекс нового (git add):



Слід зазначити, що на відміну від багатьох систем контролю версій, Git не відстежує переміщення файлів явно, тому він вважає, що відбулося перейменування файлу (про це свідчить результат виконання команди git status).

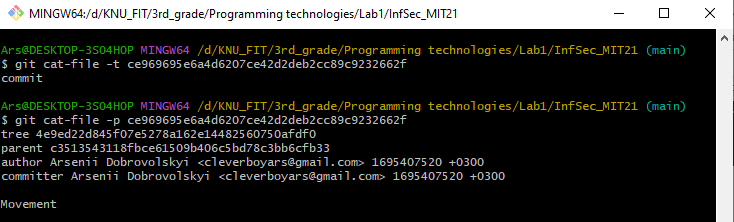
1. git log

Відображає історію комітів, починаючи від найновішого до найстарішого. Ця команда також виводить корисну інформацію про кожен коміт, таку як хеш-код, автор, дата, час створення і коментар:



1. git cat-file

Дозволяє переглядати вміст та різні корисні властивості об'єктів у репозиторії. До об’єктів відносяться коміти, дерева, теги та блоби. Якщо до команди додати параметр -t, на екран виведеться тип об'єкта, а якщо -p, то його вміст:

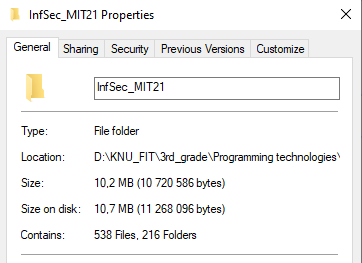


Додаткове завдання

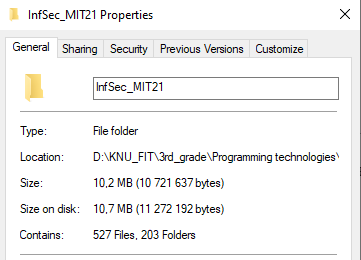
В Git спосіб зберігання даних не залежить від розміру окремого файлу в репозиторії. Він обробляє файли як послідовності байтів і стежить за змінами вмісту файлів, а не їхнім розміром. Розмір самого файлу впливає лише на обсяг даних, які Git повинен зберігати, але не на спосіб їх зберігання.

Команда git gc використовується для оптимізації роботи Git-репозиторію шляхом видалення непотрібних та застарілих файлів зі сховища об’єктів і упаковуванні решти файлів, тим самим зменшуючи розмір репозиторію.

Початковий розмір репозиторію:



Розмір репозиторію після виконання команди git gc:



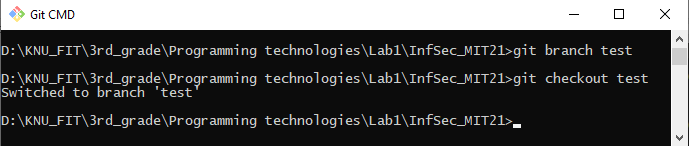
Частина №2

**Тема роботи:** Робота з різними гілками програми в Git. Конфлікти в Git.

**Мета роботи:** навчитися працювати з різними гілками програми в Git, розглянути причини виникнення конфліктів зливання у Git та засоби боротьби з ними.

**Завдання для самостійної роботи**

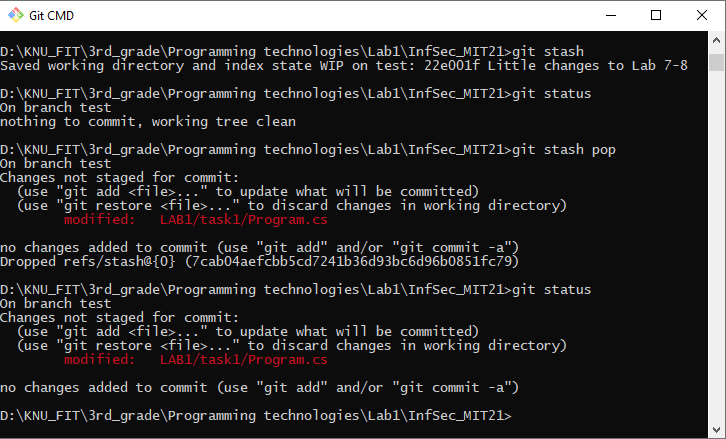
Спершу створюю нову гілку test у репозиторії з першої частини лабораторної роботи та перейду на неї. Для цього виконаю команди git branch та git checkout відповідно:



Нерідко виникає ситуація, коли потрібно перейти на іншу гілку, проте на поточній гілці містяться незбережені зміни, які поки що немає необхідності комітити. Тут стає в пригоді команда git stash. Вона дозволяє тимчасово приховати зміни, зроблені в поточній гілці, тим самим не вимагаючи виконання коміту. Згодом ці зміни можна буде дістати за допомогою команди git stash apply або git stash pop (ця команда не лише застосовує зміни, а й видаляє їх зі сховища, в якому вони тимчасово зберігалися).

Використання параметра -u з командою git stash дозволяє приховати невідстежувані Git-ом файли.

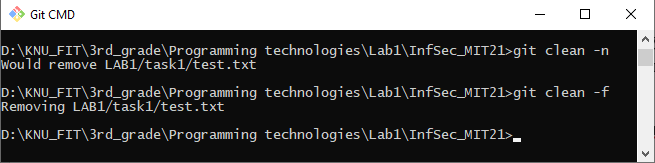
Зміню один із файлів у гілці test та приховаю його:



Як можна побачити, після приховування змін вивід команди git status свідчить про те, що репозиторій наразі чистий і не містить змін, які вимагали б нагального виконання коміту. Водночас після повернення змін командою git stash pop команда git status виявляє, що був змінений файл Program.cs.

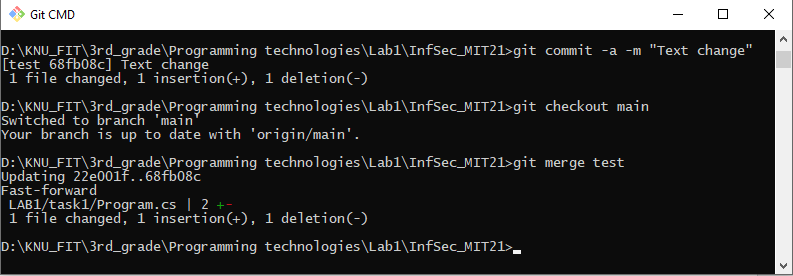
Також є можливість очищення репозиторію шляхом видалення невідстежуваних файлів за допомогою команди git clean. Вона вимагає використання щонайменше одного параметру, такого як, наприклад, -n (показує, які файли буде видалено, якщо виконати цю команду, проте не видаляє їх фактично) або -f (власне видаляє файли, тому цей параметр слід використовувати обережно!).

Для демонстрації роботи цієї команди створю файл test.txt і видалю його:

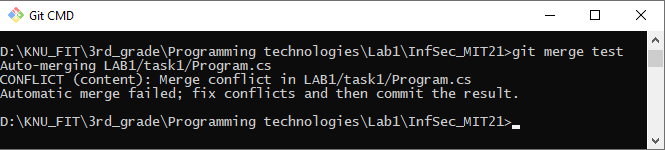


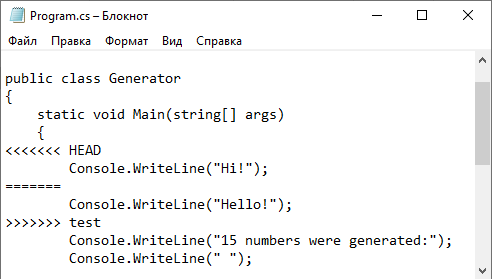
Перша команда git clean лише попереджає про те, що буде видалено файл, в той час як друга безпосередньо видаляє його без можливості відновлення.

Зроблю коміт у гілці test для збереження змін, які приховувалися раніше, та виконаю злиття цієї гілки з гілкою master за допомогою команди git merge:

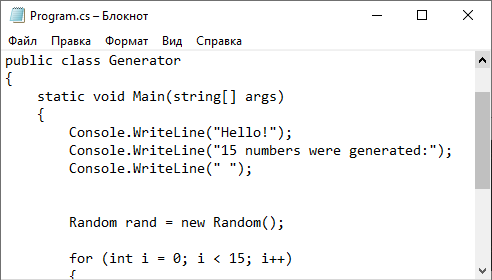


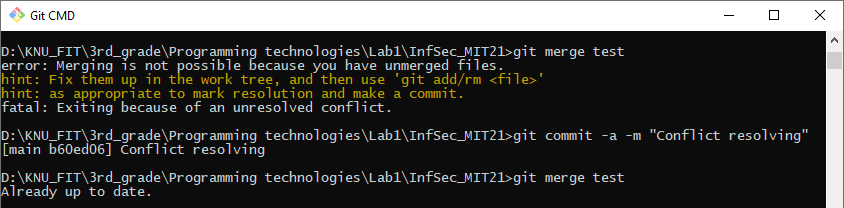
Тепер зміню файл Program.cs в одному місці одночасно у двох гілках. Це призведе до конфлікту, відповідно fast-forward злиття не зможе виконатися і буде призупинене до моменту вирішення конфлікту:





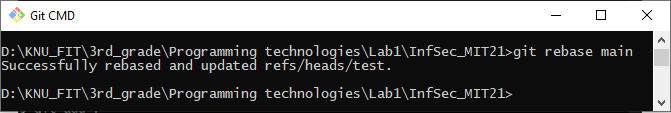
Вручну редагую файл і роблю коміт:





Інтегрувати зміни з гілки test у гілку main можна також за допомогою перебазовування (rebase). Цей спосіб полягає у перенесенні комітів поточної гілки поверх комітів тієї, в яку виконується перебазовування. Іншими словами, коміти цільової гілки «відтісняються» комітами поточної і об’єднуються з ними, внаслідок чого зберігається чиста, а головне, зрозуміла історія комітів.

Перебазовування здійснюється командою git rebase:

****

**Висновок:** в ході виконання лабораторної роботи було розглянуто призначення та функціональні можливості системи контролю версій Git. Ми навчилися виконувати базові операції в Git, працювати з гілками і зливати їх між собою, а також з’ясували причини виникнення конфліктів та дізналися про способи їх оперативного вирішення.