Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 1

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Системи контроля версій. Розподілена система контролю версій «Git».»

Виконала:
студентка групи ІА-33
Самойленко Анастасія

Перевірив:
асистент кафедри ІСТ
Мягкий Михайло Юрійович

Київ 2025

Вступ

Системи контролю версій є важливим інструментом у процесі розробки програмного забезпечення, оскільки вони дозволяють зберігати історію змін, організовувати командну роботу та забезпечувати надійність у керуванні проєктами. Використання таких систем дає змогу ефективно відстежувати зміни у файлах, повертатися до попередніх версій і синхронізувати роботу кількох розробників.

Однією з найпоширеніших сучасних систем контролю версій є **Git**. Це розподілена система, що надає кожному користувачу повну копію репозиторію з усією історією проєкту. Такий підхід робить роботу більш гнучкою, дозволяє працювати офлайн та легко інтегрувати зміни у спільний репозиторій. Git підтримує роботу з гілками, тегами, злиття змін та вирішення конфліктів, що робить його стандартом у програмуванні.

У межах даної лабораторної роботи ми повинні ознайомитися з основними можливостями системи Git, зокрема створенням і клонуванням репозиторіїв, виконанням базових операцій (коміти, робота з гілками, злиття, додавання тегів), а також використанням віддалених репозиторіїв для спільної роботи.

Теоретичні відомості

Призначення систем управління версіями

Системи управління версіями (Version Control Systems, VCS) — це програмні засоби, що дозволяють відстежувати та зберігати зміни у файлах, переважно у вихідному коді програм. Вони надають можливість створювати ревізії файлів, повертатися до попередніх версій, знаходити авторів змін та причини їх внесення. Такі системи широко застосовуються у розробці ПЗ, але також використовуються в інших сферах роботи з електронними документами.

Історія розвитку систем контролю версій

Розвиток систем контролю версій умовно поділяють на кілька етапів:

- Ранній етап – примітивне копіювання проєктів у різні папки; перша справжня система – RCS (1982), яка зберігала лише зміни у файлах.
- Централізовані системи (CVS, SVN) – з'явилися у 1990-х, використовували центральний сервер для спільної роботи. SVN стало більш надійною системою порівняно з CVS, однак мало проблеми з гілками та злиттями.
- Децентралізація (Git, Mercurial, ClearCase) – кожен розробник отримує повну копію репозиторію. Git (2005, Лінус Торвальдс) став революційною системою завдяки швидкості, гнучкості та зручній роботі з гілками. Mercurial був схожим конкурентом, але поступово втратив популярність.
- Хмарні платформи (GitHub, GitLab, Bitbucket) – з 2010-х років інтегрують Git з сервісами для командної роботи, CI/CD, DevOps, аналітики та автоматичного тестування.

Робота з Git

Git може використовуватися через:

- Командний рядок – забезпечує повний доступ до всіх команд і сценаріїв автоматизації.
- Графічні оболонки (GUI) – зручні для візуалізації репозиторію, але часто мають обмежений набір команд.

Основні можливості Git: створення локальних і віддалених репозиторіїв, виконання комітів, робота з гілками, злиття змін, розв'язання конфліктів, використання тегів та синхронізація з хмарними платформами.

Хід роботи

1. Для початку нам потрібно створити нову директорію, використовуючи команду mkdir з назвою «lab_1» та перейти в неї (рис.1).

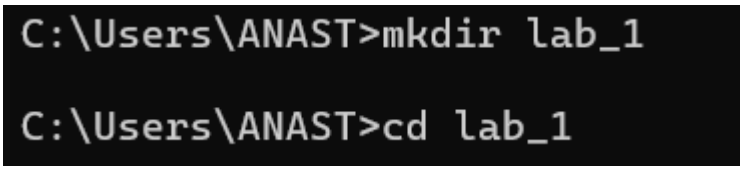

Рисунок 1 – Створення та перехід до нової директорії

2. Додамо локальний репозиторій. Для цього ми відкриємо термінал та пропишемо команду ***git init*** (рис.2). У результаті з'явився новий каталог з початковою структурою репозиторію Git.

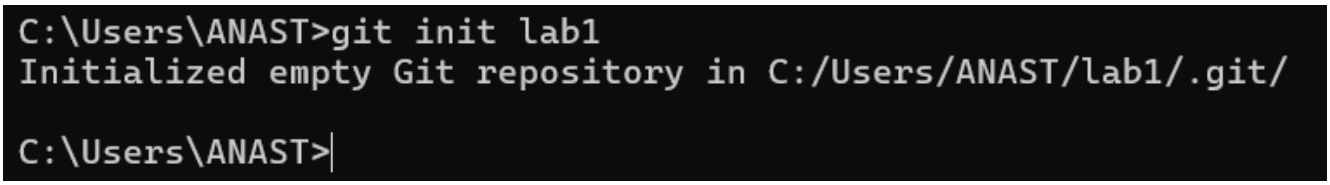

Рисунок 2 – Створення локального репозиторію

3. Створимо та додамо файл у список відстеження та виконаємо коміт (рис.3). Як бачимо тепер у репозиторії є перша зафіксована версія проєкту.

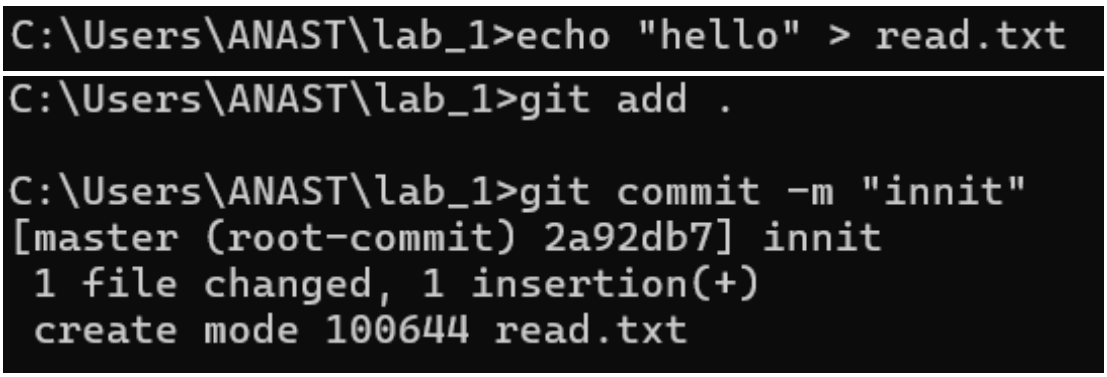

Рисунок 3 – Додавання файлу та його коміт

4. Нам потрібно створити нові гілки в нашому проєкті. Для цього початково перевіремо вже існуючі гілки. (рис.4). Далі додамо нову гілку ***f1*** та ***f2*** різними способами (рис.5):
   - **git branch f1 –** створення гілки f1
   - **git checkout -b f2 –** створення та перехід у гілку f2

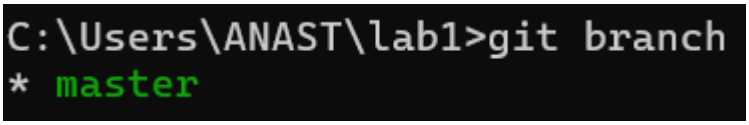

Рисунок 4 – Гілки проєкту

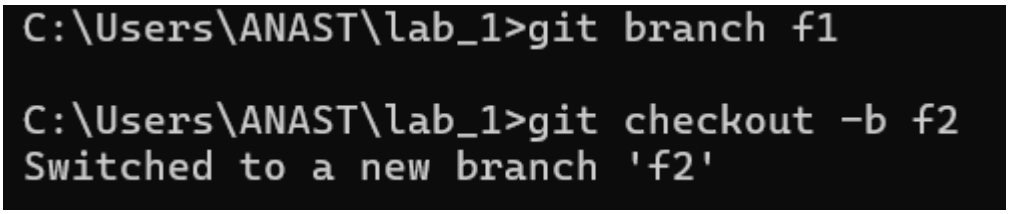

Рисунок 5 – Створення нових гілок проєкту

5. Далі створемо два текстових файли f1 та f2 з відповідними цифрами всередині(рис.6)

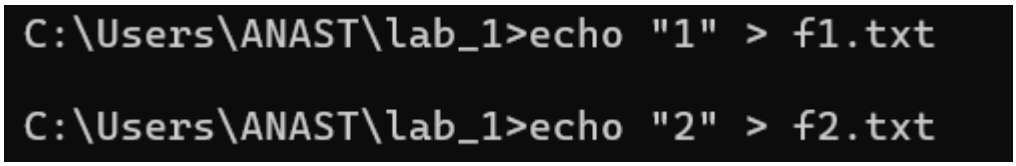

Рисунок 6 – Створення нових текстових файлів

6. Далі додамо до гілки в якій ми знаходимось (f2) файл з відповідною назвою та закомітемо зміни(рис.7)

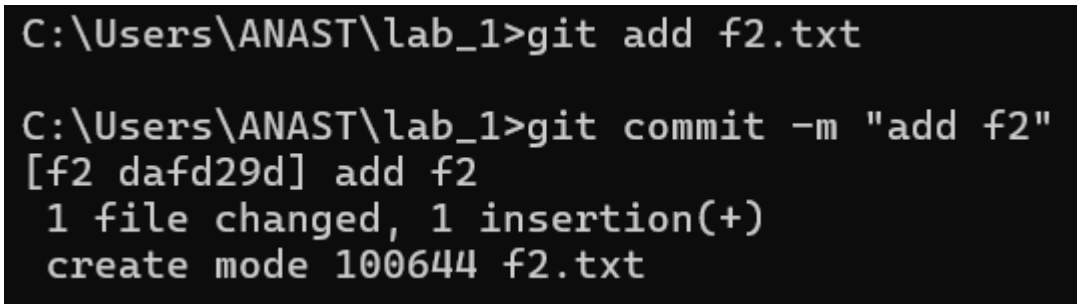

Рисунок 7 – Додавання текстового файлу f2.txt до гілки f2 та коміт змін

7. Аналогічно робимо дану процедуру з гілкою f1 для початку перейшовши в неї (риис.8).

```
C:\Users\ANAST\lab_1>git checkout f1
Switched to branch 'f1'

C:\Users\ANAST\lab_1>git add f1.txt

C:\Users\ANAST\lab_1>git commit -m "add f1"
[f1 5f6943d] add f1
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt
```

Рисунок 8 – Додавання текстового файлу f1.txt до гілки f1 та коміт змін

8. Перевіримо історію проєкту за допомогою команди **git log --all –graph** (рис.9)

```
C:\Users\ANAST\lab_1>git log --all --graph
* commit 5f6943dadfba146a23d084490c2c12aa86f6c032 (HEAD -> f1)
| Author: SamoilenkoAnastasia <samoilenko.anastasia@lll.kpi.ua>
| Date:   Sat Sep 13 13:09:42 2025 +0300
|
|     add f1
|
| * commit dafd29d5e50aa05d66e80796771b65766e356771 (f2)
|/  Author: SamoilenkoAnastasia <samoilenko.anastasia@lll.kpi.ua>
|   Date:   Sat Sep 13 13:08:49 2025 +0300
|
|       add f2
|
* commit 2a92db750ab121283e0ca3f0626ca222c01b49ca (master)
  Author: SamoilenkoAnastasia <samoilenko.anastasia@lll.kpi.ua>
  Date:   Sat Sep 13 13:04:36 2025 +0300

      innit
```

Рисунок 9 – Перегляд історії комітів у вигляді графа

9. Зробимо об’єднання гілок f1 та f2 за допомогою команди ***git merge*** (рис.10). Як бачимо, Git повідомляє про успішне злиття гілок за допомогою стратегії ort. В процесі злиття був створений новий файл f2.txt, до якого додано один рядок. Повідомлення також вказує, що загалом змінено один файл, і встановлено стандартні права доступу. Це означає, що злиття пройшло успішно без конфліктів, і зміни були додані до репозиторію.

```
C:\Users\ANAST\lab_1>git merge f2
Merge made by the 'ort' strategy.
 f2.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt
```

Рисунок 10 – Злиття гілок

10. Тепер переглянемо знову історію за допомогою графу та переконаємось, що все виконано успішно (рис.11).

```
C:\Users\ANAST\lab_1>git log --all --graph
*   commit cb0bf337f8e56c827f03fb87158ddc3758604230 (HEAD -> f1)
|\  Merge: 5f6943d dafd29d
| | Author: SamoilenkoAnastasia <samoilenko.anastasia@lll.kpi.ua>
| | Date:   Sat Sep 13 13:13:42 2025 +0300
| |
| |     Merge branch 'f2' into f1
| |
| * commit dafd29d5e50aa05d66e80796771b65766e356771 (f2)
| | Author: SamoilenkoAnastasia <samoilenko.anastasia@lll.kpi.ua>
| | Date:   Sat Sep 13 13:08:49 2025 +0300
| |
| |     add f2
| |
* | commit 5f6943dadfba146a23d084490c2c12aa86f6c032
|/  Author: SamoilenkoAnastasia <samoilenko.anastasia@lll.kpi.ua>
|   Date:   Sat Sep 13 13:09:42 2025 +0300
|
|       add f1
|
* commit 2a92db750ab121283e0ca3f0626ca222c01b49ca (master)
  Author: SamoilenkoAnastasia <samoilenko.anastasia@lll.kpi.ua>
  Date:   Sat Sep 13 13:04:36 2025 +0300

      innit
```

Рисунок 11 -Перегляд історії комітів у вигляді графа

Висновки

У ході лабораторної роботи я ознайомилася з принципами роботи систем контролю версій та практично опанувала роботу з розподіленою системою Git. Я створила локальний репозиторій, додала файли та директорії, виконала коміти для фіксації змін. Створила додаткову гілку для внесення змін, успішно її злила з основною гілкою та завершила merge. Під час роботи я навчилася працювати з гілками, комітами та синхронізувати зміни, що дало практичні навички організації версійного контролю у проєктах.