Міністерство освіти і науки України

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»

Виконав

студент групи ІА–33:

Китченко Д.В.

Перевірив:

Мягкий М.Ю.

Київ 2025

Зміст

# Вступ

Системи контролю версій (СКВ) є невід'ємною частиною сучасного процесу розробки програмного забезпечення та управління проєктами. Вони дозволяють ефективно відстежувати зміни у файлах, повертатися до попередніх станів, організовувати паралельну роботу кількох розробників та запобігати втраті даних.

Серед них особливе місце займає розподілена система Git, яка стала де-факто стандартом у галузі завдяки своїй швидкості, гнучкості та потужним можливостям для роботи з гілками. Розуміння принципів роботи Git є ключовою компетенцією для будь-якого IT-фахівця.

Метою даної лабораторної роботи є здобуття практичних навичок роботи з децентралізованою системою контролю версій Git. В ході роботи необхідно навчитися виконувати основні операції: створення репозиторію, виконання комітів, робота з гілками, їх злиття та вирішення можливих конфліктів, що виникають під час командної розробки.

## Теоретичні відомості

### 1. Призначення систем управління версіями

Система управління версіями (від англ. Version Control System, VCS) – це програмне забезпечення, яке допомагає команді розробників керувати змінами у вихідному коді з часом. СКВ дозволяє фіксувати зміни у файлах у репозиторії, створюючи нові версії (ревізії). Це дає можливість повертатися до попередніх версій коду для аналізу, пошуку помилок або скасування невдалих змін. Таким чином, можна точно відстежити, хто, коли і які зміни зробив у коді, а також зрозуміти причину цих змін. Такі системи широко використовуються не лише в розробці ПЗ, а й в інших сферах, де ведеться робота з великою кількістю електронних документів, що постійно змінюються, наприклад, у САПР або для ведення технічної документації.

### 2. Історія розвитку систем контролю версій

Розвиток СКВ можна умовно поділити на кілька етапів:

1) Ранній етап (Локальні СКВ): Початково контроль версій зводився до простого копіювання директорій проєкту (проєкт_v1, проєкт_фінал). Першою повноцінною системою була RCS (1982), яка зберігала не повні копії файлів, а лише різницю між версіями (дельти). Її головним недоліком була робота лише з окремими файлами та відсутність централізованої моделі для командної роботи.

2) Етап централізованих систем: На початку 90-х з'явилися системи, що працювали за моделлю клієнт-сервер. Існував один центральний репозиторій, до якого підключалися всі розробники.

- CVS: Одна з перших популярних систем, яка дозволила командну роботу.
- SVN (Subversion): Наступний крок еволюції. Більш надійна та швидка система, проста у використанні (commit, update). Проте, робота з гілками та вирішення конфліктів залишалися складними операціями. Головний недолік – повна залежність від центрального сервера.

3) Етап децентралізації (DVCS): Революційний підхід, де кожен розробник має на своєму комп'ютері повну копію всього репозиторію з усією історією.

- Git (2005): Створений Лінусом Торвальдсом для розробки ядра Linux. Дозволяє працювати автономно, робить операції з гілками швидкими та легкими. Зміни зберігаються у вигляді "знімків" (snapshots), а не різниць, що забезпечує високу швидкість та цілісність даних.
- Mercurial (2005): Схожа на Git система, яка довгий час була її головним конкурентом.

4) Етап хмарних платформ: З 2010-х років акцент змістився на інтеграцію СКВ (переважно Git) з хмарними сервісами, які надають додаткові інструменти для співпраці та автоматизації.

- GitHub, GitLab, Bitbucket: Це не СКВ, а веб-платформи для хостингу Git-репозиторіїв. Вони пропонують графічний інтерфейс, інструменти для код-рев'ю (Pull/Merge Requests), відстеження завдань, CI/CD (безперервна інтеграція та доставка) та багато іншого.

3. Робота з Git

Робота з Git може виконуватися з командного рядка, що надає повний доступ до всього функціоналу, або за допомогою візуальних клієнтів (SourceTree, GitKraken), які спрощують виконання базових операцій.

Основна ідея Git – кожен розробник має власний локальний репозиторій. Синхронізація між розробниками відбувається через обмін змінами з одним або кількома віддаленими репозиторіями.

Основні команди та робочий процес:

1 git clone: Клонування (створення локальної копії) віддаленого репозиторію.
2 git pull: Отримання останніх змін з віддаленого репозиторію.
3 git checkout -b <назва-гілки>: Створення нової гілки для роботи над конкретним завданням. Це ізолює нові зміни від стабільної версії коду.
4 git add <файл>: Додавання змінених файлів до індексу (Staging Area) – проміжної зони, де готуються зміни для наступного коміту.
5 git commit -m "Опис змін": Фіксація змін з індексу в локальному репозиторії. Створюється нова версія ("знімок") проєкту.

6 git push: Відправка локальних комітів на віддалений репозиторій для обміну з командою.

7 git merge <назва-гілки>: Об'єднання (злиття) змін з однієї гілки в поточну.

Цей цикл є основою щоденної роботи програміста в команді.

Хід роботи

Для виконання лабораторної роботи було виконано послідовність дій з демонстрацією ключових можливостей системи Git: від створення репозиторію до вирішення конфлікту злиття.

Крок 1. Створення локального репозиторію

Спершу створюємо нову директорію для нашого проєкту та ініціалізуємо в ній порожній Git-репозиторій за допомогою команди git init.

```
C:\Users\User>mkdir lab1

C:\Users\User>cd lab1

C:\Users\User\lab1>git init
Initialized empty Git repository in C:/Users/User/lab1/.git/

C:\Users\User\lab1>
```

Ця команда створила приховану папку .git, де Git буде зберігати всю історію та конфігурацію проєкту.

Крок 2. Створення файлу та перший коміт

Створимо файл README.md з початковим текстом, додамо його до індексу (staging area) та зафіксуємо (зробимо коміт) в репозиторії.

```
C:\Users\User\lab1>echo "My First Git Project" > README.md

C:\Users\User\lab1>git add README.md

C:\Users\User\lab1>git commit -m "Initial commit: Add README.md"
[master (root-commit) 2530997] Initial commit: Add README.md
 1 file changed, 1 insertion(+)
 create mode 100644 README.md

C:\Users\User\lab1>
```

Крок 3. Створення директорії та другий коміт

Для демонстрації роботи з директоріями створимо папку data та файл info.txt всередині неї.

```
C:\Users\User\lab1>mkdir data

C:\Users\User\lab1>echo "Some useful information" > data/info.txt

C:\Users\User\lab1>git add data

C:\Users\User\lab1>git commit -m "Add data directory with info file"
[master 0fab574] Add data directory with info file
 1 file changed, 1 insertion(+)
 create mode 100644 data/info.txt
```

Крок 4. Робота з гілками: створення, перемикання та злиття

Створимо нову гілку feature/update-readme для того, щоб внести зміни в файл README.md, не зачіпаючи основну гілку master.

```
C:\Users\User\lab1>git checkout -b feature/update-readme
Switched to a new branch 'feature/update-readme'
```

Тепер, знаходячись у новій гілці, внесемо зміни та зафіксуємо їх.

```
C:\Users\User\lab1>git add README.md

C:\Users\User\lab1>git commit -m "Update README with project description"
[feature/update-readme 1b1d12c] Update README with project description
 1 file changed, 1 insertion(+)
```

Повернемося до основної гілки та об'єднаємо зміни з нашої feature гілки.

```
C:\Users\User\lab1>git checkout master
Switched to branch 'master'

C:\Users\User\lab1>git merge feature/update-readme
Updating 0fab574..1b1d12c
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

Злиття пройшло успішно, оскільки зміни не конфліктували між собою.

Крок 5. Створення та вирішення конфлікту злиття

Для демонстрації однієї з найважливіших навичок, змоделюємо ситуацію, коли Git не зможе автоматично об'єднати зміни.

З гілки master створюємо нову гілку conflict-branch.

В цій гілці змінимо першу строку файлу info.txt та зробимо коміт.

```
C:\Users\User\lab1>git checkout -b conflict-branch
Switched to a new branch 'conflict-branch'

C:\Users\User\lab1>echo "Information from conflict-branch" > data/info.txt

C:\Users\User\lab1>git add data/info.txt

C:\Users\User\lab1>git commit -m "Update info from conflict-branch"
[conflict-branch f3bee72] Update info from conflict-branch
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Повертаємося в master і змінюємо ту саму строку в тому самому файлі, але на інший текст.

```
C:\Users\User\lab1>git checkout master
Switched to branch 'master'

C:\Users\User\lab1>echo "Information from main branch" > data/info.txt

C:\Users\User\lab1>git add data/info.txt

C:\Users\User\lab1>git commit -m "Update info from master"
[master bbab9d5] Update info from master
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Спробуємо злити conflict-branch в master, що має спричинити конфлікт.

```
C:\Users\User\lab1>git merge conflict-branch
Auto-merging data/info.txt
CONFLICT (content): Merge conflict in data/info.txt
Automatic merge failed; fix conflicts and then commit the result.
```

Вирішення конфлікту. Git повідомив про помилку. Відкриваємо файл data/info.txt і бачимо спеціальні маркери, що позначають конфліктні зміни:

<<<<<<< HEAD

"Information from main branch"

=======

"Information from conflict-branch"

>>>>>>> conflict-branch

Редагуємо файл вручну, залишаючи той варіант, який нам потрібен, та видаляючи маркери. Наприклад, оберемо компромісний варіант:
"Information from main branch"
Після ручного редагування фіксуємо вирішення конфлікту, додавши файл до індексу та створивши новий коміт злиття.

```
C:\Users\User\lab1>git add data/info.txt

C:\Users\User\lab1>git commit -m "Merge conflict-branch and resolve conflict in info.txt"
[master 99bb270] Merge conflict-branch and resolve conflict in info.txt
```

Крок 6. Перегляд фінальної історії
Щоб переглянути всю виконану роботу, використаємо команду git log з опціями для кращої візуалізації.

```
C:\Users\User\lab1>git log --oneline --graph --all
*   99bb270 (HEAD -> master) Merge conflict-branch and resolve conflict in info.txt
|\
| * f3bee72 (conflict-branch) Update info from conflict-branch
* | bbab9d5 Update info from master
|/
* 1b1d12c (feature/update-readme) Update README with project description
* 0fab574 Add data directory with info file
* 2530997 Initial commit: Add README.md
```

## Висновки

Під час виконання даної лабораторної роботи було досягнуто поставлену мету, а саме – здобуто глибокі практичні навички роботи з розподіленою системою контролю версій Git.

Було освоєно базові операції, що складають основу щоденної роботи розробника: ініціалізація локального репозиторію (git init), фіксація змін у файлах та директоріях (git add, git commit), а також робота з гілками. Зокрема, було продемонстровано створення ізольованої гілки для розробки нової функціональності (git checkout -b) та її успішне, безконфліктне злиття з основною гілкою (git merge).

Особливу увагу було приділено моделюванню та вирішенню конфлікту злиття. Цей практичний досвід є критично важливим, оскільки він імітує реальні робочі ситуації, що виникають під час паралельної роботи кількох розробників над одним файлом. Отримані знання дозволяють не лише виконувати послідовні дії, але й розуміти логіку розподіленої системи контролю версій, що є основою для ефективної командної розробки, підтримки чистоти та цілісності історії проєкту.