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

Тема: Системи контролю версій. Розподілена система контролю версій “Git”.

Виконала
студентка групи ІА–32:
Слюсарева А.А.

Київ 2025

ЗМІСТ

ЛАБОРАТОРНА РОБОТА № 1

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

Мета: Навчитися виконувати основні операції в роботі з децентралізованими системами контролю версій на прикладі роботи з сучасною системою Git.

1.1. Завдання

· Ознайомитись із короткими теоретичними відомостями.

·Створити Git репозиторій.

· Клонувати Git репозиторій.

· Продемонструвати базову роботу з репозиторієм: створення версій, додавання тегів, робіт з гілками (створення та злиття), робота з комітами, вирішення конфліктів, а також робота з віддаленим репозиторієм.

1.2. Теоретичні відомості

1.2.1. Призначення систем управління версіями

Система управління версіями (від англ. Version Control System або Source Control System) – програмне забезпечення яке призначено допомогти команді розробників керувати змінами в вихідному коді під час роботи [1]. Система керування версіями дозволяє додавати зміни в файлах в репозиторій і таким чином після кожної фіксації змін мани нову ревізію файлів. Це дозволяє повертатися до попередніх версій коду для аналізу внесених змін або пошуку, які зміни привели до появи помилки.

Таким чином можна знайти хто, коли і які зміни зробив в коді, а також чому ці зміни були зроблені.

1.2.2. Історія розвитку систем контролю версій

Умовно, розвиток систем контролю версій можна розбити на наступні етапи: ранній етап, етап централізованих систем, етап децентралізації та етап хмарних платформ.

Ранній етап

На цьому етапі основна увага приділялася роботі з окремими файлами у локальному середовищі.

Найпершою системою контролю версій була система «скопіювати і вставити», коли більшість проєктів просто копіювалася з місця на місце зі зміною назва (проєкт_1; проєкт_новий; проєкт_найновіший і т.д.), як правило у вигляді zip архіву або подібних (arj, tar).

RCS

Однією з основних нововведень RCS було використання дельт для зберігання змін (тобто зберігаються ті рядки, які змінилися, а не весь файл). Однак він мав низку недоліків.

Насамперед він був тільки для текстових файлів. Не було центрального репозиторію; кожен версіонований файл мав власний репозиторій як rcs файлу поруч із самим файлом. Тобто якщо на проєкті було 100 файлів, поруч лягало 100 rcs файлів. У кращому випадку ці 100 файлів утворювалися в директорії RCS (при правильному налаштуванні). Найменування версій і гілок було неможливим.

Етап централізованих систем

На початку 90-х почалася епоха централізованих систем контролю версій. У цей період розробники почали переходити до централізованих систем, що дозволяли працювати кільком користувачам одночасно через сервер

SVN

SVN – у порівнянні з CVS це був наступний крок. Надійна та швидкодіюча систему контролю версій, яка зараз розробляється в рамках проєкту Apache Software Foundation. Вона реалізована за технологією клієнт-сервер та відрізняється неймовірною простотою – дві кнопки (commit, update). Порівняно з CVS, це удосконалена централізована система з кращим управлінням комітами та резервними копіями.

Незважаючи на це, SVN дуже погано вміє створювати та зливати гілки та погано вирішує конфліктні ситуації з версіями. Але, в багатьох проєктах до цих пір використовується SVN.

Етап децентралізації

Децентралізовані системи усунули залежність від центрального сервера та дозволили кожному розробнику мати повну копію репозиторію. У 1992 році з'явився один з основних представників світу систем розподіленого контролю версій. ClearCase був однозначно попереду свого часу і для багатьох він досі є однією з найпотужніших систем контролю версій будь- коли створених.

Дана система дозволяла користуватися віртуальною файловою системою для зберігання та отримання змін; мала широкий діапазон повноважень щодо зміни, впровадження у процес розробки (аудит збірок товару, версії, зливання змін, динамічні уявлення); запускалася на безлічі різних систем.

У 2005 році було створено дві знакові системи контролю версій Git та Mercurial. Вони стали революційними системами, які забезпечили швидкість, надійність і гнучкість роботи.

Git

Лінус Торвальдс, т.зв. Батько Лінукса, розробив і впровадив першу версію Гіт для надання можливості розробникам ядра Лінукс проводити контроль версій не тільки в BitKeeper.

Гіт є системою розподіленого контролю версій, коли кожен розробник має власний репозиторій, куди він вносить зміни [2]. Далі система гіт синхронізує репозиторії із центральним репозиторієм. Це дозволяє проводити роботу незалежно від центрального репозиторію (на відміну від SVN, коли версіонування передбачало наявність зв'язку з центральним сервером), перекладає складності ведення гілок та склеювання змін більше на плечі системи, ніж розробників та ін.

Зміни зберігаються у вигляді наборів змін (changeset), що отримує унікальний ідентифікатор (хеш-сума на основі самих змін).

Mercurial

Mercurial був створений як і Git після оголошення про те, що BitKeeper більше не буде безкоштовним для всіх. Багато в чому схожий на Git, Mercurial також використовує ідею наборів змін, але на відміну від Git, зберігає їх у не у вигляді вузла в графі, а вигляді плоского набору файлів і папок, званих revlog.

Етап хмарних платформ

Приблизно з 2010 року і до цих пір також можна виділити етап хмарних платформ, основним лозунгом яких є «Інтеграція та автоматизація».

У сучасну епоху акцент робиться на інтеграції систем контролю версій із хмарними платформами та автоматизації розробки. І в більшості випадків такою системою контролю версій є Git. Основною характеристикою цього етапу є інтеграція систем контролю версій в хмарні сервіси для глобальної співпраці, які додатково підтримують розширену функціональність для автоматизації процесів та інтеграції з іншими сервісами.

1.3 Хід роботи

```
D:\Programming>git init test
Initialized empty Git repository in D:/Programming/test/.git/

D:\Programming>cd test
```

1. Ініціалізуємо пустий репозиторій

```
D:\Programming\test>git checkout -b "branch1"
Switched to a new branch 'branch1'
```

```
D:\Programming\test>echo "test" > test.txt

D:\Programming\test>git add .

D:\Programming\test>git commit -m "initial"
[test1 (root-commit) 4c3e912] initial
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
```

2. Створюємо першу гілку, додаємо в неї коміт

```
D:\Programming\test>git branch "test2"

D:\Programming\test>git branch -a
* test1
  test2
```

3. Створюємо другу гілку, перевіряємо їх наявність

```
D:\Programming\test>echo "test2244" >> text.txt
```

4. Змінюємо вміст файлу у гілці 2

```
D:\Programming\test>git add .

D:\Programming\test>git commit -m "t"
[test2 12a16ac] t
 1 file changed, 1 insertion(+)
```

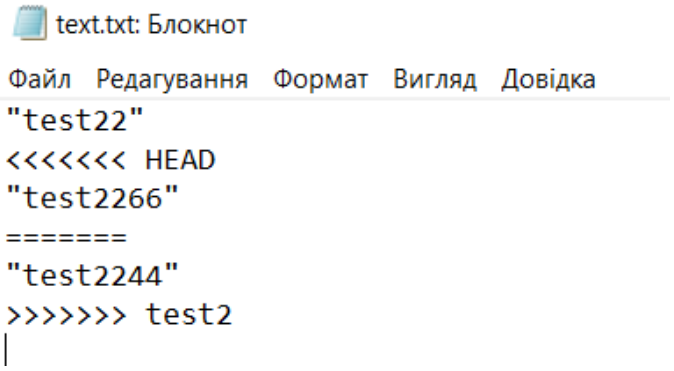
5. Комітимо зміни у гілці “test2”

```
D:\Programming\test>git checkout "test1"
Switched to branch 'test1'

D:\Programming\test>echo "test2266" >> text.txt

D:\Programming\test>git add .

D:\Programming\test>git commit -m "t"
[test1 37b061a] t
 1 file changed, 1 insertion(+)
```

6. Повторюємо попередні дії для гілки “test1”

```
D:\Programming\test>git merge "test2"
Auto-merging text.txt
CONFLICT (content): Merge conflict in text.txt
Automatic merge failed; fix conflicts and then commit the result.
```

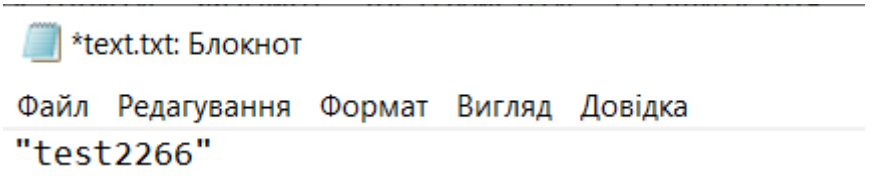
7. Робимо спробу змерджити гілку “test2” у гілку “test1”



8. Отримуємо конфлікт

```
D:\Programming\test>git log --graph
* commit 37b061ae26a55e500431561e24eed7f47d80f07f (HEAD -> test1)
| Author: nnstsya <tonastyaket@gmail.com>
| Date:   Mon Sep 22 10:00:50 2025 +0300
|
|     t
|
* commit e850049436ee26da7c38e8262dd214782d189dc8
| Author: nnstsya <tonastyaket@gmail.com>
| Date:   Mon Sep 22 09:54:35 2025 +0300
|
|     t
|
* commit 4c3e9128763ba020039a5aa091795bda87fe9751 (main)
  Author: nnstsya <tonastyaket@gmail.com>
  Date:   Mon Sep 22 09:52:00 2025 +0300

      initial
```

9. Візуалізуємо історію комітів



10. Виправляємо файл з конфліктом, залишаючи потрібні нам зміни

```
D:\Programming\test>git add .

D:\Programming\test>git commit -m "t"
[test1 17d5827] t

D:\Programming\test>git branch
  main
* test1
  test2

D:\Programming\test>git merge "test2"
Already up to date.
```

11. Зливаємо гілки

1.4 Висновок

Під час даної роботи було відпрацьовано основні гіт команди, вирішення конфліктів між гілками. Вивчення теоретичних відомостей про історію створення і етапи систем контролю версій.

1.5 Контрольні запитання

1. Що таке система контролю версій (СКВ)?

Система для збереження історії змін у файлах і спільної роботи з ними..

2. Поясніть відмінності між розподіленою та централізованою СКВ.

Централізована – один сервер з історією. Розподілена – копія історії у кожного.

3. Поясніть різницю між stage та commit в Git.

Stage – підготовка змін. Commit – фіксація змін в історії.

4. Як створити гілку в Git?

git branch “new-branch” або git checkout -b “new-branch”

5. Як створити або скопіювати репозиторій Git з віддаленого серверу?

git clone “url”

6. Що таке конфлікт злиття, як створити конфлікт, як вирішити конфлікт?

Конфлікт – різні зміни в одному місці. Створюється при зливанні гілок з комітами, які мають конфлікт. Вирішується ручним редагуванням.

7. В яких ситуаціях використовуються команди: merge, rebase, cherry-pick?

merge – об’єднання гілок. rebase – перенесення комітів. cherry-pick – якщо потрібно взяти окремий коміт.

8. Як переглянути історію змін Git репозиторію в консолі?

git log

9. Як створити гілку в Git не використовуючи команду git branch?

git checkout -b “new-branch”

10. Як підготувати всі зміни в поточній папці до коміту?
git add .

11. Як підготувати всі зміни в дочірній папці до коміту?
git add path/to/folder/

12. Як переглянути перелік наявних гілок в репозиторії?
git branch

13. Як видалити гілку?
git branch -d branchname

14. Які є способи створення гілки та в чому між ними різниця?
Через git branch (створення без переходу) або git checkout -b (створення + перехід).