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
Технології розроблення програмного забезпечення
«Системи контроля версій.
Розподілена система контролю версій Git»

Виконав
студент групи ІА–34:
Бородай А.С.

Київ 2025

Зміст

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

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.

Теоретичні відомості

Призначення систем управління версіями

Система управління версіями – програмне забезпечення яке призначено допомогти команді розробників керувати змінами в вихідному коді під час роботи. Система керування версіями дозволяє додавати зміни в файлах в репозиторій і таким чином після кожної фіксації змін мани нову ревізію файлів.

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, коли більшість проєктів просто копіювалася з місця на місце зі зміною назви, як правило у вигляді zip архіву або подібних (arj, tar).

Однією з основних нововведень RCS було використання дельт для зберігання змін (тобто зберігаються ті рядки, які змінилися, а не весь файл). Не було центрального репозиторію; кожен версіонований файл мав власний репозиторій як rcs файлу поруч із самим файлом.

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. Одна із перших найпопулярніших систем – CVS.

SVN – 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 2005 році було створено дві знакові системи контролю версій Git та Mercurial. Вони стали революційними системами, які забезпечили швидкість, надійність і гнучкість роботи. Вони мають багато ідентичних команд, хоча «під капотом» вони мають різні підходи до реалізації.

Гіт є системою розподіленого контролю версій, коли кожен розробник має власний репозиторій, куди він вносить зміни. Далі система гіт синхронізує репозиторії із центральним репозиторієм. Це дозволяє проводити роботу незалежно від центрального репозиторію, 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).

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.Можна виділити такі ключові хмарні платформи на основі Git: GitHub, GitLab, Bitbucket. Вони підтримують CI/CD, спільну роботу та інтеграції, інструменти для DevOps, аналітики та автоматичного тестування.

Таким чином, 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.

Робота з Git

Робота з Git може виконуватися з командного рядка, а також за допомогою візуальних оболонок. Командний рядок використовується програмістами, як можливість виконання всіх доступних команд, а також можливості складання складних макросів. Візуальні оболонки як правило дають більш наглядне представлення репозиторію у вигляді дерева, та більш зручний спосіб роботи з репозиторієм, але, дуже часто доступний не весь набір команд Git, а лише саме ті, що найчастіше використовуються.

Основна ідея Git, як і будь-якої іншої розподіленої системи контроля версій – кожен розробник має власний репозиторій, куди складаються зміни (версії) файлів, та синхронізація між розробниками виконується за допомогою синхронізації репозиторіїв.

Хід роботи

1. Створити локальний репозиторій.

```
PS D:\MindMap> git init lab1
Initialized empty Git repository in D:/MindMap/lab1/.git/
```

2. Додати довільний файл з довільним текстом (в даному випадку текст – test).

```
PS D:\MindMap> cd lab1
PS D:\MindMap\lab1> echo "foofoo" > foo.txt
```

3. Зафіксувати додавання файлу.

```
PS D:\MindMap\lab1> git add foo.txt
PS D:\MindMap\lab1> git commit -m "FooFoo added"
[master (root-commit) dae8705] FooFoo added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 foo.txt
```

4. Додати нову директорію з довільним ім'ям.

```
PS D:\MindMap\lab1> mkdir plan


    Directory: D:\MindMap\lab1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        12.09.2025     17:07                plan
```

5. Додати файл у директорію.

```
PS D:\MindMap\lab1> cd plan
PS D:\MindMap\lab1\plan> echo "adadadad" > ada.txt
```

6. Зафіксувати додавання директорії із файлом.

```
PS D:\MindMap\lab1\plan> cd ..
PS D:\MindMap\lab1> git add plan
PS D:\MindMap\lab1> git commit -m "Add Plan directoty with ada text"
[master 7791535] Add Plan directoty with ada text
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 plan/ada.txt
```

7. Створити гілку і перейти на неї.

```
PS D:\MindMap\lab1> git branch bugFix
PS D:\MindMap\lab1> git checkout bugFix
Switched to branch 'bugFix'
```

8. Видалити додану директорію і зафіксувати зміни.

```
PS D:\MindMap\lab1> rmdir plan -r
PS D:\MindMap\lab1> git add .
PS D:\MindMap\lab1> git commit -m "Remove plan"
[bugFix 8d9d37f] Remove plan
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 plan/ada.txt
```

9. Злити зміни з основною гілкою.

```
PS D:\MindMap\lab1> git checkout master
Switched to branch 'master'
PS D:\MindMap\lab1> git merge bugFix
Updating 7791535..8d9d37f
Fast-forward
 plan/ada.txt | Bin 22 -> 0 bytes
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 plan/ada.txt
```

10. Вивести історію на екран.

```
PS D:\MindMap\lab1> git log
commit 8d9d37f0e59d2ff7b832d9c517dea729d070e6ae (HEAD -> master, bugFix)
Author: AndriiMind01 <borodai.andrii@lll.kpi.ua>
Date:   Fri Sep 12 17:12:38 2025 +0300

    Remove plan

commit 7791535f854f79498d153c85c80a0d62c5065fa8
Author: AndriiMind01 <borodai.andrii@lll.kpi.ua>
Date:   Fri Sep 12 17:10:06 2025 +0300

    Add Plan directoty with ada text

commit dae8705ae96f400c7763595d506fd5d21b20bdbf
Author: AndriiMind01 <borodai.andrii@lll.kpi.ua>
Date:   Fri Sep 12 17:07:06 2025 +0300

    FooFoo added
```

Відповіді на питання до лабораторної роботи

1. *Що таке система контролю версій (СКВ)?*

Програмне забезпечення яке призначено допомогти команді розробників керувати змінами в вихідному коді під час роботи. Система керування версіями дозволяє додавати зміни в файлах в репозиторій і таким чином після кожної фіксації змін мани нову ревізію файлів

2. *Поясніть відмінності між розподіленою та централізованою СКВ.*

У централізованій СКВ один центральний сервер, що зберігає всю історію версій та кожен користувач має лише робочу копію останніх файлів, коли у розподіленої СКВ кожен користувач має повну локальну копію всього репозиторію, включаючи всю його історію.

У централізованих СКВ сервер є точкою відмови, коли розподілені СКВ більш надійні, адже якщо будь-який сервер "вмирає", будь-який клієнтський репозиторій може бути використаний для його відновлення.

Для розподілених СКВ більшість операцій миттєві, оскільки виконуються локально, а ніж у централізованих, де деякі операції часто повільніші, бо йдуть по мережі.

3. Поясніть різницю між stage та commit в Git.

git add — готує зміни в staging area, а git commit — фіксує підготовлені зміни (staging area) в історії (створюється хеш, додається автор, дата коміту та повідомлення).

4. Як створити гілку в Git?

git branch <назва>

**Ця команда створить нову гілку, але не переключить на неї. Щоб переключитися використовуйте: git checkout -b <назва>*

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

Скопіювати репозиторій:

git clone <url-репозиторію>

Створити новий локальний репозиторій і під'єднати його до віддаленого:

```
mkdir my-project
cd my-project
git init
git remote add origin <url-репозиторію>
// створюєте деякі файли, зробите коміт (git add . і git commit -m "message")
git push -u origin <назва>
```

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

Конфлікт злиття виникає, коли Git не може автоматично об'єднати зміни з різних гілок. Зазвичай це трапляється, коли дві гілки змінили одну й ту саму область одного й того ж файлу.

Щоб створити конфлікт змініть один і той же рядок у одному й тому ж файлі в двох різних гілках.

Git позначить конфліктні місця у файлі спеціальними маркерами:

```
<<<<<<< HEAD
Це зміни в поточній гілці (наприклад, main).
=======
Це зміни з гілки, яку ви намагаєтеся злити (наприклад, feature).
>>>>>>> feature
```

Вручну відредагуйте файл, вибравши, які зміни залишити, потім підготуйте виправлений файл до коміту: *git add <ім'я-файлу>* та завершіть злиття: *git commit*.

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

Merge - використовується для об'єднання історії двох гілок. Створює новий коміт злиття, який має двох батьків.

Rebase - "перемотує" зміни з однієї гілки на кінець іншої, переписуючи історію, роблячи її лінійнішою та чистішою. Використовується для ваших локальних гілок, щоб синхронізувати їх з оновленою main гілкою перед злиттям.

Cherry-pick - використовується, коли потрібно взяти один конкретний коміт з однієї гілки і застосувати його до поточної. Корисно для виправлення помилок, коли виправлення зроблено в одній гілці, а потрібно перенести його в іншу.

8. Як переглянути історію змін Git репозиторію в консолі?

git log

git log --oneline // стислий вивід (хеш і заголовок коміту).

git log --graph // показати історію у вигляді дерева гілок.

git log --oneline --graph --all // поєднання всіх варіантів для кращої візуалізації.

git log -p // показати детальну інформацію по кожному коміту, включаючи diff (зміни у файлах).

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

git checkout -b <назва-нової-гілки>

**Можна використовувати також: git switch -c <назва-нової-гілки>*

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

git add .

11. Як підготувати всі зміни в дочірній папці до коміту?

git add <шлях до папки> // наприклад: git add src/components/

12. Як переглянути перелік наявних гілок в репозиторії?

git branch

**У випадку із віддаленими гілками: git branch -a*

13. Як видалити гілку?

git branch -d <назва>

**У випадку із віддаленими гілками: git push origin --delete <назва>*

14. Які є способи створення гілки та в чому між ними різниця?

- Від поточного стану:

git branch <назва> // нова гілка створюється від коміту, на якому зараз знаходиться HEAD (поточна гілка).

- Від конкретного коміту:

git branch <назва> <хеш> // можна створити гілку, яка буде вказувати на будь-який коміт в історії, вказавши його хеш.

- Від тегу:

git branch <назва> <тег> // створює гілку, яка починається з коміту, позначеного тегом.

- Від віддаленої гілки:

git switch -c <local-name> origin/<remote-name> // створює локальну гілку, яка відстежує віддалену (стандартний спосіб почати роботу над гілкою, створеною іншим розробником).

Висновок: у ході виконання нашої лабораторної роботи ми навчитися виконувати основні операції в роботі з децентралізованими системами контролю версій на прикладі роботи з сучасною системою Git.