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:
Тимчук Владислав
ІА-34
Перевірив:
Мягкий М. Ю.

Київ-2025

AUDIO PLAYER

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.

ВСТУП

У сучасній розробці програмного забезпечення системи керування версіями є невід'ємною складовою командної роботи та організації процесу розробки. Вони забезпечують збереження історії змін у коді, можливість повернення до попередніх версій, паралельну роботу кількох розробників та ефективне злиття гілок розвитку проєкту.

У ході цієї лабораторної роботи буде опановано основні принципи роботи з Git, зокрема створення репозиторію, додавання та фіксація змін, створення та об'єднання гілок, а також перегляд історії комітів.

ЗМІСТ

1 Теоретичні відомості

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.

Такі системи найбільш широко використовуються при розробці програмного забезпечення для зберігання вихідних кодів програми, що розробляється. Однак вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю електронних документів, що безперервно змінюються. Зокрема, системи керування версіями застосовуються

у САПР, зазвичай у складі систем керування даними про виріб (PDM). Керування версіями використовується у інструментах конфігураційного керування (Software Configuration Management Tools).

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). Звичайно, такі маніпуляції над файловою системою навряд чи можна назвати хоч скільки повноцінною системою контролю версій (або системою взагалі). Для вирішення цих проблем 1982 року з'являється RCS.

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 Одина із перших найпопулярніших систем (і досі використовувана) система контролю

версій – CVS. Цю епоху можна охарактеризувати досить сформованим уявленням про системи контролю версій, їх можливості, появою центральних репозиторіїв (та синхронізації дій команди).

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 будьколи 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. Вони мають багато ідентичних команд, хоча «під капотом» вони мають різні підходи до реалізації. Досить довго вони конкурували одна з одною, але починаючи з 2018 Git поступово виходить на лідерську позицію серед безкоштовних систем контролю версій.

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.

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.

Прикладами візуальних оболонок для роботи з Git є Git Extension, SourceTree, GitKraken, GitHub Desktop та інші.

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. Процес роботи виглядає так, як зображено на рисунку 1.1.

Відповідно, є ряд основних команд для роботи [2]:

1. Клонувати репозиторій (git clone) – отримати копію репозиторію на локальну машину для подальшої роботи з ним;

2. Синхронізація репозиторіїв (git fetch або git pull) – отримання змін із віддаленого (вихідного, центрального, або будь-якого іншого такого ж) репозиторію;

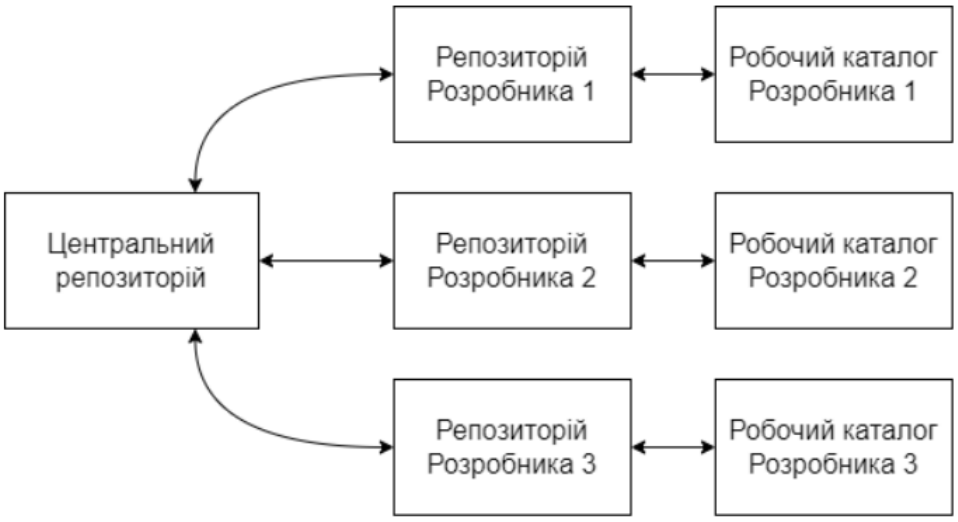

Рисунок 1.1. Схема процесу роботи з Git

3. Фіксація змін в репозиторій (git commit) – фіксація виконаних змін в програмному коді в локальний репозиторій розробника;

4. Синхронізація репозиторіїв (git push) – переслати зміни – push – передача власних змін до віддаленого репозиторію – Записати зміни – commit – створення нової версії;

5. Оновитись до версії – update – оновитись до певної версії, що є у репозиторії.

6. Об’єднання гілок (git merge) – об’єднання вказаною гілки в поточну (часто ще називається «злиттям»).

Таким чином, якщо розглядати основний робочий процес програміста в команді, то він виглядає наступним чином: На початку роботи з проєктом виконується клонування, після цього, в рамках виконання поставленої задачі, створюється бранч і всі зміни в коді, зроблені в рамках цієї задачі фіксуються в репозиторії (періодично виконується синхронізація з основним репозиторієм). Далі, коли задача виконана, то виконується об’єднання гілки з основною гілкою і фінальна синхронізація з центральним репозиторієм.

2 Хід роботи

Історія CMD:

```
C:\Studying\3 курс\ТРПЗ\Audio Player>git init lab1
Initialized empty Git repository in C:/Studying/3 курс/ТРПЗ/Audio Player/lab1/.git/

C:\Studying\3 курс\ТРПЗ\Audio Player>echo
ECHO is on.

C:\Studying\3 курс\ТРПЗ\Audio Player>echo "testing" > text.txt

C:\Studying\3 курс\ТРПЗ\Audio Player>cd C:\Studying\3 курс\ТРПЗ\Audio Player\lab1

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>echo "testing" > text.txt

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git add test.txt
fatal: pathspec 'test.txt' did not match any files

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git add text.txt

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git commit -m "INIT"
[master (root-commit) c11ccf6] INIT
 1 file changed, 1 insertion(+)
 create mode 100644 text.txt

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>mkdir child_dir

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>cd child_dir

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1\child_dir>echo
ECHO is on.

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1\child_dir>echo "inner" > inner.txt

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1\child_dir>cd lab1
The system cannot find the path specified.
```

```
C:\Studying\3 курс\ТРПЗ\Audio Player\lab1\child_dir>cd C:\Studying\3 курс\ТРПЗ\Audio Player\lab1

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>cd child_dir

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1\child_dir>git add child_dir
fatal: pathspec 'child_dir' did not match any files

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1\child_dir>cd C:\Studying\3 курс\ТРПЗ\Audio Player\lab1

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git add child_dir

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git commit -m "Add test child dir"
[master 9925c3a] Add test child dir
 1 file changed, 1 insertion(+)
 create mode 100644 child_dir/inner.txt

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git branch feature_branch

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git checkout feature_branch
Switched to branch 'feature_branch'

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>rmdir child_dir -r

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git add .

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git commit -m "child dir removed"
[feature_branch 2ccd0ce] child dir removed
 1 file changed, 1 deletion(-)
 delete mode 100644 child_dir/inner.txt

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git checkout master
Switched to branch 'master'
```

```
C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git merge feature_branch
Updating 9925c3a..2ccd0ce
Fast-forward
 child_dir/inner.txt | 1 -
 1 file changed, 1 deletion(-)
 delete mode 100644 child_dir/inner.txt

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>git log
commit 2ccd0ce8f28be8f9c25123c9aa8b22451b0bd4fd (HEAD -> master, feature_branch)
Author: fromz67 <git config --global user.email tycmhuk.vladyslav@lll.kpi.uagit config --global user.email>
Date:   Fri Sep 12 19:30:13 2025 +0300

    child dir removed

commit 9925c3ae9ef9662cff675ea9a6845aa52b000b06
Author: fromz67 <git config --global user.email tycmhuk.vladyslav@lll.kpi.uagit config --global user.email>
Date:   Fri Sep 12 19:28:39 2025 +0300

    Add test child dir

commit c11ccf6c815d0dac49e543a96ce276fad1d759af
Author: fromz67 <git config --global user.email tycmhuk.vladyslav@lll.kpi.uagit config --global user.email>
Date:   Fri Sep 12 19:25:45 2025 +0300

    INIT

C:\Studying\3 курс\ТРПЗ\Audio Player\lab1>
```

3 Висновок

Під час виконання лабораторної роботи було вивчено керування версіями та на практиці відпрацьовано базові команди Git. Було створено локальний репозиторій, додано та зроблено зміни у файлах, створено й об’єднано гілки, а також переглянуто історію комітів.