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
Системи контролю версій. Git

Виконав:

Студент групи ІА-22

Птачик Р.С.

Перевірив:

Мягкий М.Ю.

Київ 2024

Лабораторна робота №1

Теоретичні відомості

Система контролю версій (СКВ) – це інструмент для відстеження змін у файлах з можливістю повернення до попередніх версій. Вона дозволяє зберігати історію всіх внесених змін. Існує 2 варіанта СКВ – централізована та розподілена СКВ:

Централізована СКВ – Вся історія змін зберігається на одному сервері. Якщо сервер недоступний, неможливо відстежити історію чи робити коміти.

Розподілена СКВ – Кожен користувач має локальну копію всього проекту з повною історією змін. Це дозволяє робити зміни та відстежувати історію локально, без підключення до серверу.

Історія змін зберігається в **репозиторії** - місце, де зберігається вся історія змін проекту. Репозиторій дозволяє відстежувати всі зміни, зроблені в проекті: історії комітів, змін, гілок і злиттів. Існує 2 варіанта репозиторію: локальний (наприклад **Git**) та віддалений (наприклад **GitHub**).

Робота з Git

Для того, щоб почати працювати з Git спершу необхідно ініціалізувати репозиторій. Для цього вводимо команду **git init** в папці, яка буде майбутнім репозиторієм. Також можна створити віддалений репозиторій, наприклад GitHub, та під’єднати до нього локальний.

Створення віддаленого репозиторію: віддалений репозиторій можна створити на платформі GitHub.

Підключення локального репозиторію до віддаленого: ініціалізовуємо локальний репозиторій (git init); виконуємо наступну команду:

- **git remote add origin https://github.com/username/repo.git**

Клонування віддаленого репозиторію: якщо потрібно перенести віддалений репозиторій в локальний, то тоді потрібно виконати наступну команду:

- **git clone https://github.com/username/repo.git**

Після ініціалізації ми можемо використовувати інші команди git та зберігати історію змін.

Як зберігати зміни в файлах: щоб зберегти зміни в історії репозиторію необхідно спочатку додати файли на **stage** (стадія індексації) – це процес підготовки файлів для коміту. Для цього є команда **git add**, яка має наступні параметри:

- **git add file.txt** – додавання на stage файл file.txt;
- **git add .** – додавання всіх змін починаючи з поточної директорії;
- **git add -A** – додавання абсолютно всіх змін у репозиторії.

За допомогою **git status** можна перевірити які файли є на стадії індексації.

Після додавання файлів на stage зміни можна зберегти за допомогою команди **git commit**:

- **git commit -m "text"** – додавання коміту з текстом "text".

Тепер зміни збережено в історії репозиторію. Історію змін в репозиторії можна переглянути за допомогою команди **git log**:

- **git log --oneline** - історія в 1 рядку;
- **git log --all** – перегляд всіх змін (зміни в інших гілках).

Окрім відображення тексту коміту, автора та часу додавання, git log також відображає **хеш** коміту.

Створення та видалення гілок

Коли історія репозиторію не порожня, тобто в нас є хоча б 1 коміт, git створює гілку **master (main)**, яка є основною гілкою репозиторію. Існує декілька способів створення гілок:

- **git branch** (назва гілки) – створення нової гілки залишаючись на поточній;
- **git switch -c** (назва гілки) – створення нової гілки та перехід на неї;
- **git checkout -b** (назва гілки) - створення нової гілки та перехід на неї.

Для перегляду які існують гілки та на якій гілці ми знаходимося є команда **git branch**.

Для того щоб зберігати коміти в інших гілках необхідно перейти на ці гілки. Для цього є наступні команди:

- **git checkout** (назва гілки) – універсальна команда для перемикання гілок, відновлення файлів і роботи з комітами;

- **git switch** (назва гілки) – більш проста та зручна команда для перемикання між гілкам.

Гілки можна видаляти за допомогою **git branch -d** (назва гілки). Також можна примусово видалити гілку завдяки **git branch -D** (назва гілки).

Об’єднання гілок

Якщо потрібно об’єднати гілки (тобто перенести історію змін), то є декілька способів це зробити завдяки **merge, rebase** та **cherry-pick**:

- **git merge** (назва гілки) – об'єднує зміни з однієї гілки в іншу та створює коміт злиття;
- **git rebase** (назва гілки) – переміщує коміти з поточної гілки поверх базової (наприклад, master), не створюючи нового коміту злиття;
- **git cherry-pick** (хеш коміту) – вибірково переносить окремий коміт з однієї гілки в іншу не об'єднуючи всю історію.

Також, для віддаленого репозиторію, існує ще один варіант злиття: **pull request**: створюється пул-реквест із гілки branch до master і, після схвалення змін, можна автоматично виконати merge або rebase через інтерфейс платформи GitHub.

При об’єднанні гілок можуть виникнути **конфлікти**: коли в різних гілках є різні зміни в одному файлі.

Вирішення конфліктів

Коли виникає конфлікт, в терміналі відображається що виник конфлікт в певному файлі (файлах). Можна також перевірити де саме виникли конфлікти за допомогою **git status**.

Конфлікти потрібно вирішувати вручну, тобто потрібно відкрити файл та відредагувати його (або залишити весь вміст, або залишити вміст з певної гілки). Приклад того, що відображається в конфліктному файлі:

<<<<<<< HEAD

123

=======

abc

>>>>>>> new-feature

Щоб вирішити даний конфлікт потрібно:

- **Якщо обидві версії потрібні:** прибрати “**<<<<<<< HEAD**”, “**=======**” та “**>>>>>>> new-feature**”;
- **Якщо лише одна з версій потрібна:** прибрати “**<<<<<<< HEAD**”, “**=======**” та “**>>>>>>> new-feature**” та прибрати вміст який нам не потрібен (“**123**” або “ **abc**”).

Після вирішення конфлікту зміни потрібно перенести на **stage** (**git add**) та продовжити злиття:

- для **merge** – **git merge –continue**;
- для **rebase** – **git rebase –continue**;
- для **cherry-pick** – **git cherry-pick –continue**.

Висновок: на цій лабораторній я дізнався що таке система контролю версій, Git, навчився створювати репозиторій, переносити файли на stage, створювати коміти, створювати гілки та об’єднувати їх і також навчився вирішувати конфлікти.