

Git та GitHub: Основи Контролю Версій

Git – інструмент контролю версій, створений Лінусом Торвальдсом у 2005 році, що дозволяє відстежувати зміни та керувати версіями проектів.

GitHub – платформа для хостингу Git репозиторіїв, що забезпечує інструменти співпраці та спільної розробки коду.

 by Oleksandr Denishchuk





Що таке Git?



Розподілена система контролю версій

Git відстежує зміни, а не версії, забезпечуючи цілісність даних через контрольні суми



Історія змін

Зберігає повну історію змін проекту, дозволяючи відкотити будь-які зміни



Спільна робота

Дозволяє командам ефективно працювати над одним проектом, відстежуючи внесок кожного



Відкритий код

Безкоштовний та з відкритим вихідним кодом, працює на всіх платформах

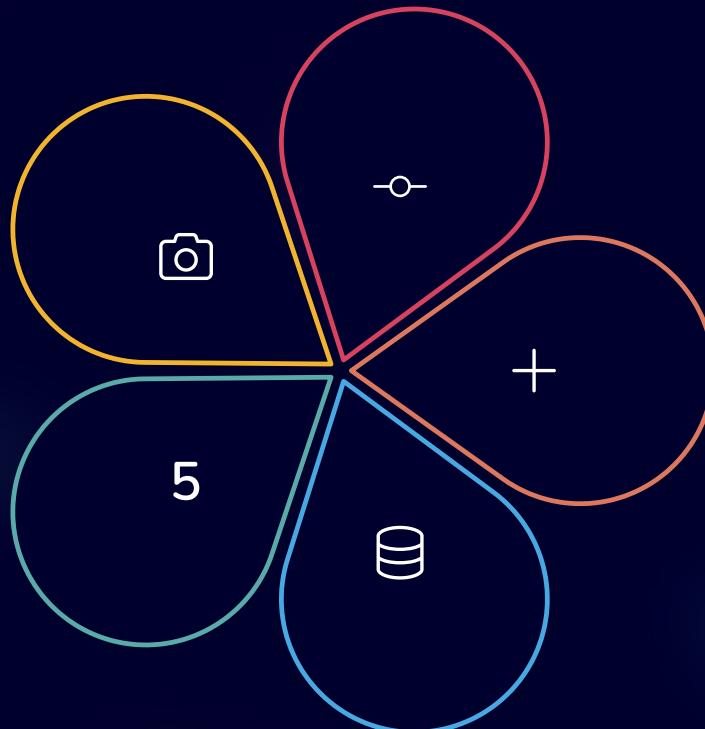
Ключові Концепції Git

Snapshots (Знімки стану)

Git зберігає запис усіх ваших файлів та їхньої історії, що дозволяє повернутися до попередніх знімків стану.

Branch (Гілка)

Незалежна лінія розробки. Основною гілкою в репозиторії на GitHub часто є main.



Commit (Коміт)

Команда для створення snapshot. Фіксує стан проекту, записує зміни до репозиторію з обов'язковим повідомленням.

Add (Додати)

Додає зміни до staging area (область індексації). Після git add файл починає відстежуватися Git.

Repository (Репозиторій)

Містить файли та їхню історію. Може існувати локально або на віддаленому сервері (GitHub).

Що таке GitHub?

Платформа для хостингу

GitHub – це найбільша платформа для веб-хостингу Git репозиторіїв, яка дозволяє розміщувати віддалені репозиторії та співпрацювати з іншими розробниками.

Платформа була придбана компанією Microsoft і продовжує розвиватися, додаючи нові функції для розробників.

Ключові можливості

- Створення та управління репозиторіями
- Інструменти для співпраці (issues, pull requests)
- Перегляд коду та обговорення
- Хостинг статичних веб-сайтів через GitHub Pages
- Інтеграція з іншими інструментами розробки

Альтернативи

Хоча GitHub є найпопулярнішою платформою, існують альтернативи, такі як GitLab та Bitbucket, які також пропонують хостинг Git репозиторіїв з різними наборами функцій.



Git

Налаштування Git для Першого Використання



Встановлення Git

Завантажте та встановіть Git з офіційного сайту



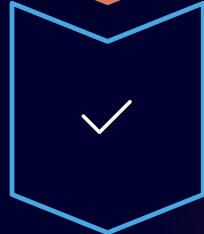
Налаштування Імені

```
git config --global user.name "Ваше Ім'я"
```



Налаштування Email

```
git config --global user.email "ваш_email@example.com"
```



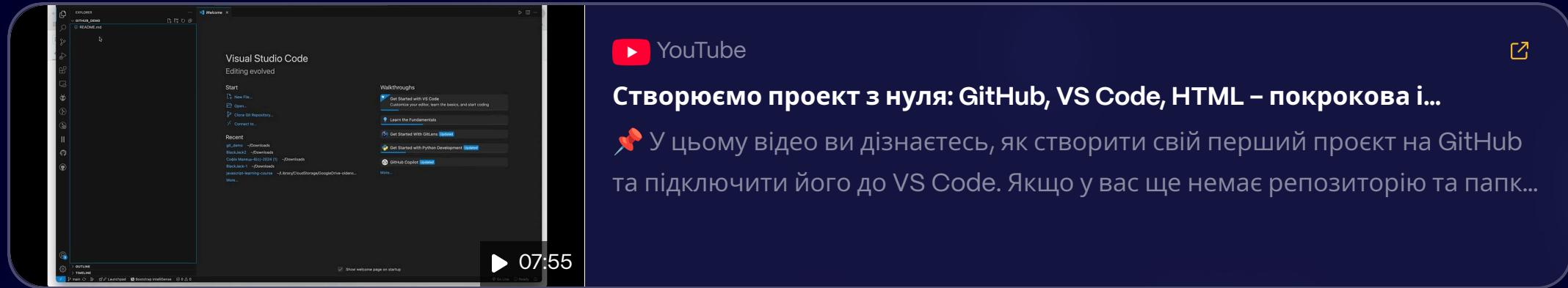
Готовність до роботи

Тепер Git знає, хто ви

Якщо ви використовуєте Git вперше, вам потрібно налаштувати ваше ім'я користувача та email. Ці дані будуть використовуватися для ідентифікації ваших комітів. Налаштування потрібно зробити лише один раз для вашого комп'ютера.

Ці команди встановлюють глобальні налаштування, які будуть застосовуватися до всіх ваших Git репозиторіїв. Якщо ви хочете використовувати різні дані для різних проектів, можна опустити пропор --global та налаштувати ці параметри окремо для кожного репозиторію.

Створення Проекту з Нуля: Крок за Кроком



Створення репозиторію на GitHub

Перейдіть на GitHub, натисніть New, назвіть репозиторій, виберіть налаштування приватності та додайте README файл для початкового вмісту.



Створення файлів

У локальній папці проекту створіть нові файли (наприклад, index.html) та додайте до них вміст.



Синхронізація з GitHub

Відправте локальні зміни на віддалений репозиторій (git push) та перевірте їх на GitHub.



4

Клонування репозиторію

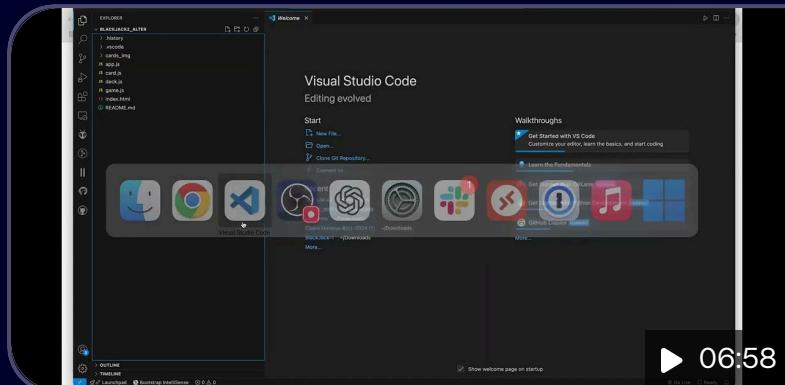
Скопіюйте посилання на репозиторій, відкрийте IDE, виберіть опцію "Клонувати Git репозиторій" та вставте посилання.



Додавання та фіксація змін

Додайте файли до staging area (git add), напишіть повідомлення для коміту та зафіксуйте зміни (git commit).

Підключення Існуючого Проекту до GitHub



Як підключити існуючий проект до GitHub? Покрокова інструкція...

- У цьому відео ви навчитеся ініціалізувати Git у вже створеному проекті та підключити його до GitHub! Якщо у вас вже є папка з кодом, але ви ще н...



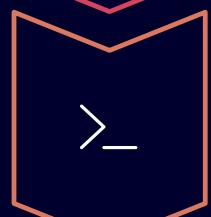
Відкрийте проект

Відкрийте папку з існуючим проектом у вашій IDE



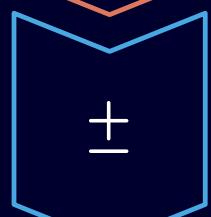
Створіть репозиторій

Створіть новий порожній репозиторій на GitHub (без README)



Ініціалізуйте Git

Виконайте команду `git init` у папці проекту



Додайте файли

Додайте всі файли до staging area: `git add .`



Зробіть коміт

Зафіксуйте зміни: `git commit -m "Initial commit"`



Підключіть репозиторій

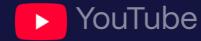
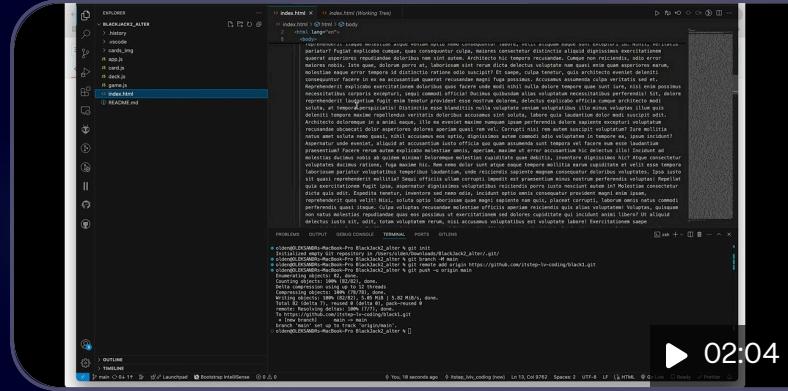
Підключіть локальний репозиторій до віддаленого: `git remote add origin URL`



Відправте зміни

Відправте локальні зміни на GitHub: `git push -u origin main`

Відкочування до Попередньої Версії



Як відкотитися до попередньої версії в Git? Простий спосіб!

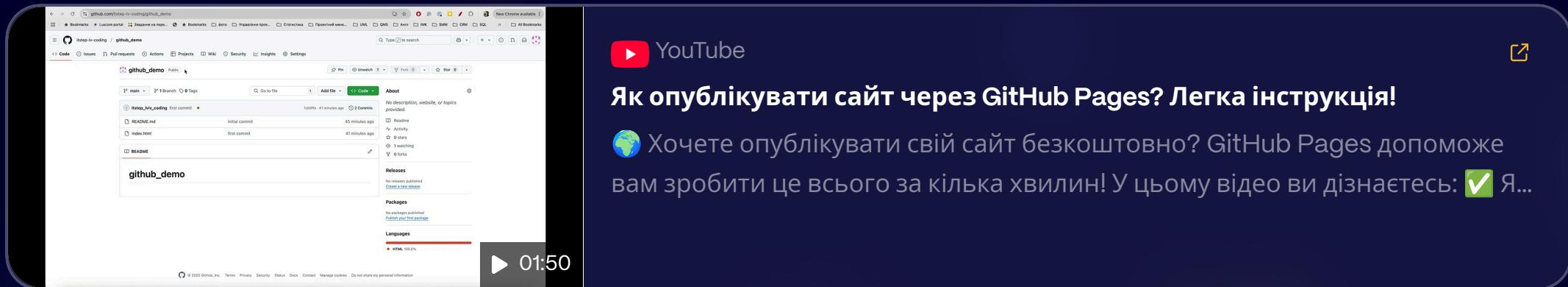
➡ Внесли неправильні зміни та хочете повернути все, як було? У цьому відео ви дізнаєтесь, як відкотитися до потрібної версії у Git і синхронізуват...

Одна з ключових переваг Git – можливість відкочуватися до попередніх версій проекту, якщо щось пішло не так. Історія комітів показує різні стани проекту, і ви можете повернутися до будь-якого з них.

Найпростіший спосіб повернутися до попередньої версії – використовувати команду `checkout`. У Visual Studio Code можна побачити історію комітів у меню контролю версій, вибрати потрібний коміт, натиснути правою кнопкою миші і вибрати `Checkout detached`.

Це поверне стан проекту до обраного коміту, і всі наступні зміни стануть недоступними в основній гілці, поки ви не повернетесь. Це дозволяє безпечно експериментувати та виправляти помилки.

Публікація Сайту через GitHub Pages



1

Створення репозиторію

Створіть репозиторій з бажаним вмістом. Файл index.html має бути в корені репозиторію.

2

Налаштування Pages

Перейдіть у settings репозиторію, знайдіть розділ pages, виберіть гілку для деплою.

3

Деплой

GitHub автоматично запустить процес деплою, який можна відстежити у вкладці actions.

GitHub Pages – це безкоштовний хостинг для статичних веб-сайтів безпосередньо з репозиторію GitHub. Кожен раз, коли ви заливаєте новий код у відстежувану гілку репозиторію, він автоматично деплоїться на GitHub Pages.

4

Публікація

Після успішного виконання ваш сайт буде доступний за наданою URL-адресою.

git

```
d: ': head' {  
ead ' cherrry--pick  
herge: { -<  
rarry-merger, '>  
herryced < -7.-rissle);  
nit: [(> (=2) -- '2 pait);  
harp 'ms=it.adlaned () );  
  
rge: 'int.'{-  
it: barger.usa? }  
>
```

Додаткові Терміни Git

HEAD

HEAD – це посилання на останній коміт у поточній гілці. Це важливий орієнтир, який показує, де ви знаходитесь у історії репозиторію.

Cherry-pick

Команда, яка використовується для додавання конкретних змін з одного коміту в іншу гілку, не зливаючи всю гілку цілком.

Merge (Злиття)

Команда для злиття змін з однієї гілки в іншу. Використовується як для злиття віддалених змін, так і для об'єднання локальних гілок.

Init (Ініціалізація)

Команда для створення нового Git репозиторію. Створює приховану папку .git, де зберігається вся інформація про історію змін.

Робота з Гілками (Branches) (опційно)

1

Створення гілки

`git branch new-branch`



Перехід до гілки

`git checkout new-branch`

3

Внесення змін

Додавання файлів та комітів у новій гілці



Злиття змін

`git checkout main, потім git merge new-branch`

5

Оновлення віддаленого репозиторію

`git push origin main`

Гілки дозволяють працювати над різними функціями або виправленнями паралельно, не впливаючи на основний код. Це особливо корисно при роботі в команді, де кожен розробник може працювати у своїй гілці, а потім об'єднувати зміни через Pull Request на GitHub.

Pull Requests на GitHub

Створення Pull Request

Після надсилання гілки на GitHub натисніть "Compare & pull request" на сторінці репозиторію. Це ключовий інструмент для співпраці на платформі.

Опис змін

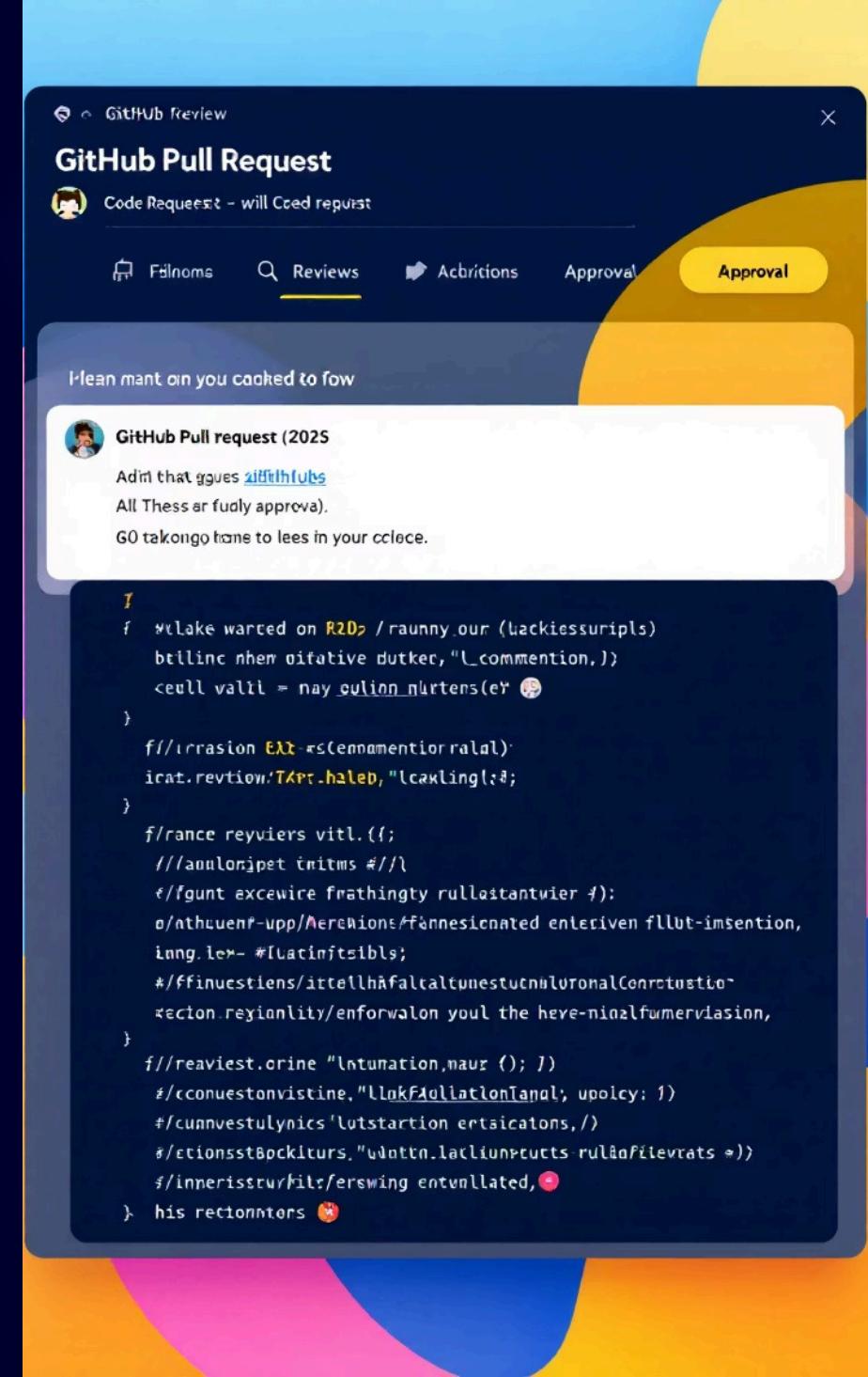
Додайте чіткий опис внесених змін та їх мети. Корисно додати посилання на issues, які вирішує даний PR.

Перегляд коду

Учасники проекту можуть переглянути код, коментувати та пропонувати зміни для забезпечення якості.

Злиття змін

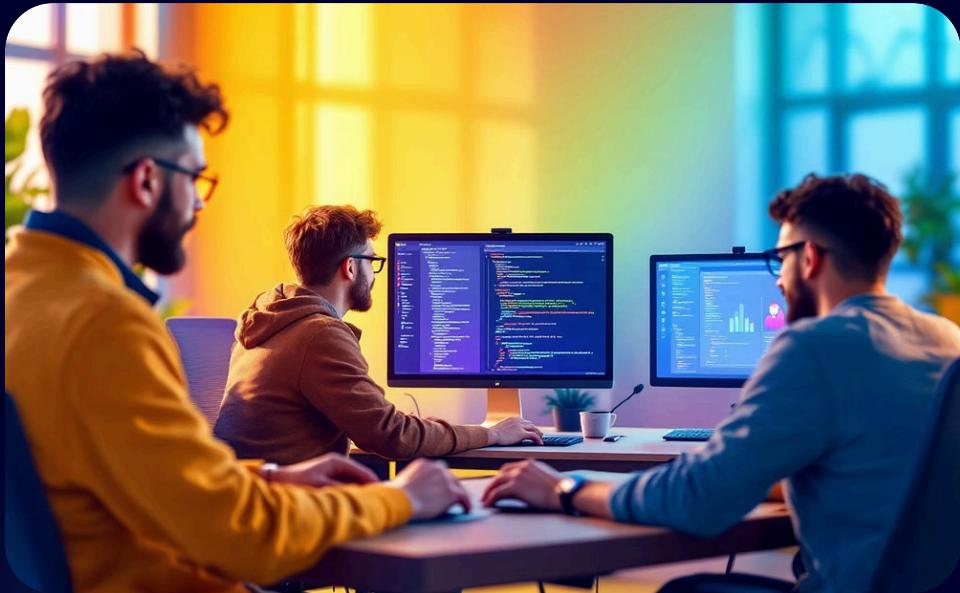
Після схвалення PR можна об'єднати зміни з основною гілкою. GitHub пропонує різні стратегії: звичайне злиття, squash або rebase.



Типовий Робочий Процес з Git



Переваги Використання Git та GitHub



Ефективна співпраця

Git та GitHub дозволяють командам розробників працювати над одним проектом одночасно, не заважаючи один одному. Кожен може працювати у своїй гілці, а потім інтегрувати зміни через Pull Request.

Повна історія проекту

Git зберігає повну історію змін проекту, дозволяючи відстежувати, хто і коли вніс які зміни, та повернутися до попередніх версій у разі потреби. Це забезпечує прозорість та безпеку розробки.

```
git commit -m "Initial commit for project A"
[master (root-commit) 1234567] Initial commit for project A
  Author: John Doe <john.doe@example.com>
  Date:   Mon Dec 14 12:00:00 2020 +0000

  This is the initial commit for project A. It contains the initial codebase and setup files.

git commit -m "Add feature X"
[master 1234567] Add feature X
  Author: John Doe <john.doe@example.com>
  Date:   Mon Dec 14 12:05:00 2020 +0000

  This commit adds a new feature X to the project. It includes the implementation code and unit tests.

git commit -m "Fix bug in feature X"
[master 1234567] Fix bug in feature X
  Author: John Doe <john.doe@example.com>
  Date:   Mon Dec 14 12:10:00 2020 +0000

  This commit fixes a bug found in the recently added feature X. It includes the fix and its corresponding test cases.

git commit -m "Refactor code for better performance"
[master 1234567] Refactor code for better performance
  Author: John Doe <john.doe@example.com>
  Date:   Mon Dec 14 12:15:00 2020 +0000

  This commit refactors the code to improve its performance. It includes the refactored code and its performance metrics.
```

Паралельна розробка

Завдяки гілкам Git дозволяє працювати над різними функціями або виправленнями паралельно, не впливаючи на основний код. Це прискорює розробку та зменшує ризики.

