МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Волинський національний університет імені Лесі Українки
Кафедра комп’ютерних наук та кібербезпеки

Л. В. Булатецька, В. В. Булатецький

СИСТЕМА КОНТРОЛЮ ВЕРСІЯМИ GIT

Методичні рекомендації до організації виконання лабораторних робіт з освітнього компонента «Технології платформи .Net»

Луцьк 2022

УДК 004.655

Рекомендовано до видання науково-методичною радою
Волинського національного університету імені Лесі Українки
(протокол № 3 від 16 листопада 2022 р.)

Рецензенти:
Собчук О. М. – кандидат пед. наук, доцент кафедри загальної математики та методики навчання інформатики Волинського національного університету імені Лесі Українки

Багнюк Н.В. кандидат технічних наук, доцент кафедри комп'ютерної інженерії та кібербезпеки ЛНТУ

Система контролю версіями Git : методичні рекомендації до організації виконання лабораторних робіт з освітнього компонента «Технології платформи .Net» / Укл: Л. В. Булатецька, В. В. Булатецький. Луцьк : ВНУ ім. Лесі Українки, 2022. 35 с. Електронні текстові данні (1 файл: 4,21 МБ).

Викладено теоретичні відомості про програмні засоби, які допомагають записувати зміни, внесені у файли, відстежуючи зміни, внесені в код, та розвивати навички командної роботи. Розглянуто систему віддаленого управління версіями Git, яка дозволяє організувати й забезпечити виконання комплексних лабораторних робіт, як складових частин більш складного програмного проекту, що виконується в команді. Рекомендовано для здобувачів вищої освіти, які вивчають програмування, зокрема для організації виконання лабораторних робіт з освітнього компонента «Технології платформи .Net», спеціальності 122 Комп'ютерні науки, освітньо-професійної програми Комп'ютерні науки та інформаційні технології.

УДК 004.655

Зміст

Основні терміни Git

Репозиторій (Repository) – каталог файлової системи, де зберігаються файли розробки.

Віддалений репозиторій (Origin) – репозиторій, що знаходиться на сервері. Це загальний репозиторій, до якого приходять усі зміни.

Локальний репозиторій – репозиторій, розташований на локальному комп'ютері розробника. Саме з ним працює програміст.

Clone – клон Origin у локальний репозиторій.

Fork – копія репозиторію. Fork дозволяє розробнику вносити зміни без ризику зіпсувати вихідний код.

Commit – запис (підтвердження) змін до локального репозиторію.

Push – відправка всіх ненаправлених підтверджених змін у віддалений репозиторій.

Pull – отримання останніх змін із віддаленого репозиторію.

Pull Request – запит на злиття гілок, або Fork репозиторію з основним репозиторієм. Запит на злиття може бути прийнятий або відхилений власником репозиторію.

Вступ

Одним із завдань підготовки здобувачів першого (бакалаврського) рівня вищої освіти спеціальності 122 Комп'ютерні науки за освітньою програмою Комп'ютерні науки та інформаційні технології є формування теоретико-практичних знань, щодо сучасних підходів і методик розробки програмного забезпечення та навичок працювати в команді. Навички командної роботи, отримані здобувачами вищої освіти в процесі виконання командних проєктів, дозволять їм у майбутній професійній діяльності легко адаптуватися до роботи в колективі, так як уміння працювати в команді є необхідними для ІТ фахівців, оскільки сфера їх професійної діяльності вимагає об'єднання в команди для генерації нових ідей, створення нових проектів і технологій, а також продукування ефективних рішень. При такій формі організації роботи виникає необхідність декільком розробникам працювати над одним і тим самим додатком, або одним і тим самим файлом, або навіть у сусідніх стрічках коду проекту. Саме для фіксації змін, які виконані паралельно, і використовують системи контролю версій. Git (розподілена система контролю версіями) – найбільш популярна на цей час система. Git дозволяє фіксувати зміни паралельно за допомогою декілька віток (branch), а потім зливати усі зміні в одну вітку, наприклад основну кодову базу.

Одним із різновидів роботи здобувачів вищої освіти у команді може слугувати організація виконання комплексних лабораторних робіт, як складових частин більш складного програмного проекту, що виконується в команді. Виконання комплексних лабораторних робіт дозволяє організувати індивідуальну роботу кожного учасника в команді. Набуття умінь здійснювати розробку певних частин проекту паралельно з іншими учасниками команди сприятиме отриманню досвіду майбутніми ІТ фахівцями щодо ефективності їх подальшої участі у розробці програмного забезпечення будь-якої складності.

1. Системи контролю версіями

Контроль версій – це практика відстеження змін, внесених у документи або програмне забезпечення, і керування ним.

Системи контролю версій (Version Control System, VCS) – це категорія програмних засобів, які допомагають записувати зміни, внесені у файли, відстежуючи зміни, внесені в код.

Оскільки, програмний продукт розробляється у співпраці групою розробників, кожен з яких вносить свої зміни у якусь певну функціональність розробки, то для злагодженої роботи потрібне програмне забезпечення, яке допомагає команді розробників ефективно спілкуватися та керувати (відстежувати) усі зміни, внесені у вихідний код, а також інформацію про те, хто і які зміни виконав. Таким програмним забезпеченням є системи контролю версій. Є два типи систем контролю версій:

- централізована система контролю версій (Centralized Version Control System – CVCS);
- розподілена система керування версіями (Distributed Version Control System – DVCS).

Централізована система контролю версій (CVCS) використовує центральний вузол (Server) для зберігання всіх файлів і забезпечує командну співпрацю. Вона працює в єдиному сховищі на центральному вузлі, до якого користувачі можуть отримати доступ (рис. 1).

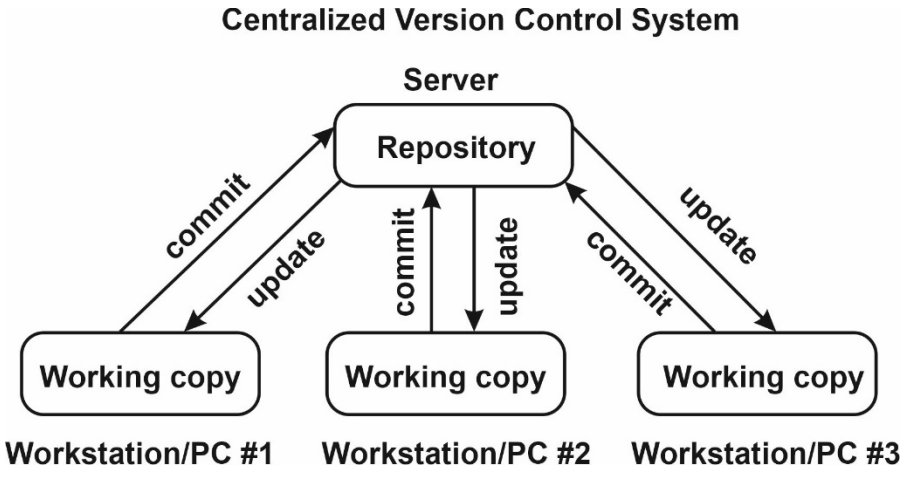


Рис. 1. Централізована система контролю версій

Репозиторій на рис. 1 вказує на центральний вузол (Server), який може бути локальним або віддаленим, безпосередньо підключеним до кожної робочої станції користувача.

Кожен користувач може оновлювати свої робочі станції за допомогою даних, наявних у сховищі, або може вносити і фіксувати зміни в сховищі. Кожна операція виконується безпосередньо в репозиторії.

Незважаючи на те, що підтримувати єдиний репозиторій дуже зручно, це має деякі серйозні недоліки. Деякі з них:

- репозиторій недоступний локально, тобто вам завжди потрібно бути підключеним до мережі, щоб виконувати будь-які дії;
- оскільки все централізоване, у будь-якому випадку збій або пошкодження центрального сервера призведе до втрати всіх даних проекту.

Розподілені системи контролю версій не обов'язково використовують сервер для зберігання всіх версій файлу проекту.

У розподіленій системі контролю версій кожен учасник має локальну копію або «клон» головного сховища, тобто кожен підтримує власний локальний репозиторій, який містить усі файли та метадані, які присутні в головному сховищі (рис. 2).

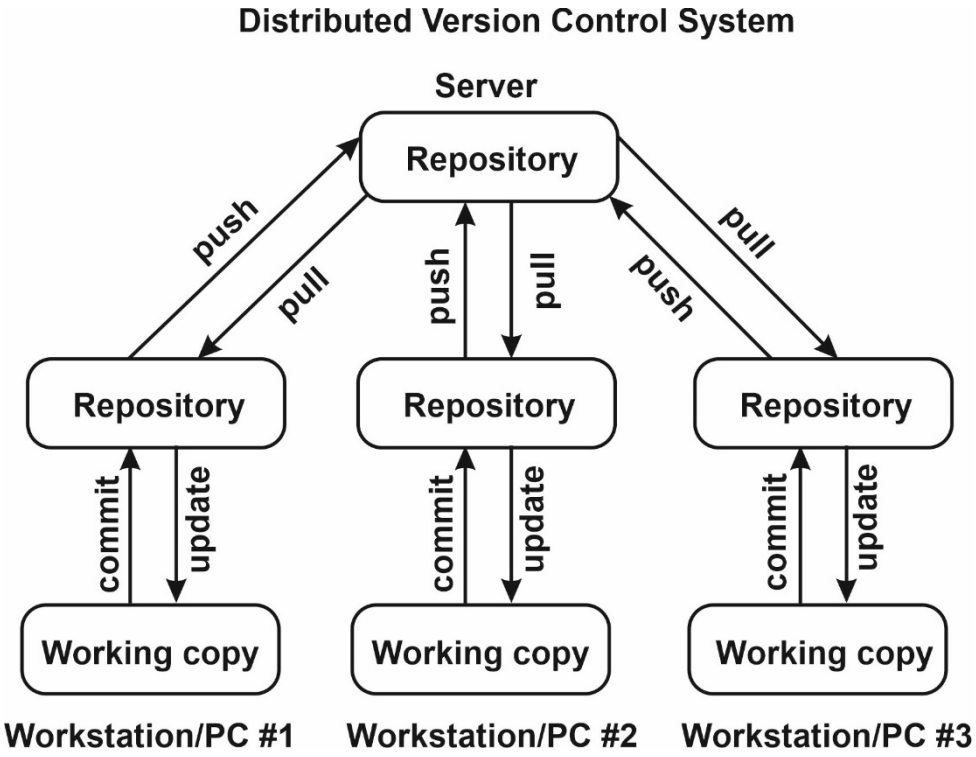


Рис. 2. Розподілена система контролю версій

В розподілених системах контролю версій (рис. 2), кожен учасник команди самостійно підтримує локальне сховище, яке насправді є копією або клоном центрального сховища на його жорсткому диску. Користувачі можуть фіксувати та оновлювати свій локальний репозиторій без будь-якого втручання. Також вони можуть оновлювати свої локальні сховища новими даними з центрального сервера за допомогою операції під назвою «pull» і відправити зміни в основне сховище за допомогою операції під назвою «push» із свого локального сховища.

Клонування цілого сховища на користувацьку робочу станцію для отримання локального сховища дає наступні переваги:

- усі операції (крім push & pull) виконуються дуже швидко, оскільки інструменту потрібен доступ лише до жорсткого диска, а не до віддаленого сервера, тому для роботи не завжди потрібне підключення до Інтернету;

- підтвердження нових наборів змін може здійснюватися локально без маніпулювання даними в основному сховищі, а коли буде готова група наборів змін, тоді відправити їх в головне сховище можна усі одночасно;
- оскільки кожен учасник має повну копію репозиторію проекту, вони можуть ділитися змінами один з одним, якщо хочуть отримати відгук, перш ніж вносити зміни в головне сховище;
- якщо центральний сервер виходить з ладу в будь-який момент часу, втрачені дані можна легко відновити з будь-якого з локальних сховищ авторів.

Git – це безкоштовний інструмент розподіленої системи керування версіями з відкритим вихідним кодом, розроблений для швидкої та ефективної роботи з будь-якими проектами, від малих до дуже великих. Він був створений Лінусом Торвальдсом у 2005 році для розробки ядра Linux. Git має функціональність, продуктивність, безпеку та гнучкість, необхідні більшості команд та окремих розробників. Git в основному використовується для керування проектом, що містить набір коду/текстових файлів, які можуть змінюватися. Git дозволяє повертати окремі файли і весь проект до попереднього стану, переглядати зміни, що відбуваються з часом. Визначати, хто останнім вносив зміни в модуль, що раптово перестав працювати, відстежити ланцюг подій, що призвела до помилок і багато іншого.

Одним з найпопулярніших ресурсів для роботи з Git є GitHub.

GitHub – сервіс для роботи із системою контролю версій Git, яка є важливим інструментом командної розробки.

Для зручності роботи з ним існує графічний клієнт GitHub Desktop та консольний Git Shell.

2. Графічний клієнт GitHub Desktop

Для роботи з сервісом GitHub розроблено програму GitHub Desktop. Вона дозволяє легко почати працювати із сервісом. Для цього потрібно зробити наступні кроки:

- завести обліковий запис на GitHub;
- завантажити та встановити програму GitHub Desktop;
- створити новий репозиторій, або підключитись до існуючого.

Реєстрація на GitHub. Для того щоб створити власний репозиторій потрібно зареєструватися на GitHub: https://github.com/ (рис. 1).

Для налаштування репозиторію Git використовуючи клієнт GitHub Desktop, скачаємо та встановимо GitHub Desktop з офіційного сайту: https://desktop.github.com/ (рис. 2).

Після першого входу до GitHub Desktop з’явиться запит на введення логіну та паролю від GitHub.com. Після цього користувач отримає доступ до всіх репозиторіїв, збережених у зареєстрованому профілі.

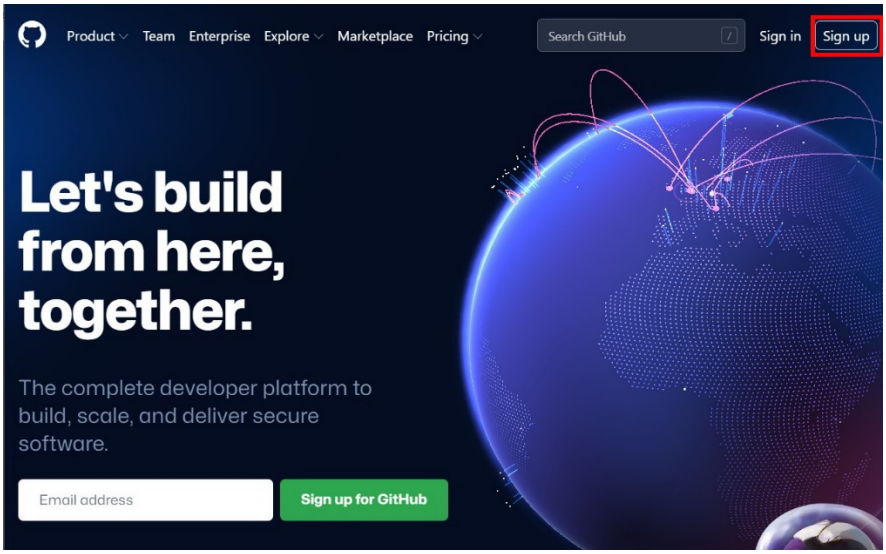


Рис. 1. Стартова сторінка онлайн-сервісу GitHub

Рис. 2. Офіційний сайт GitHub Desktop

3. Створення нового репозиторію

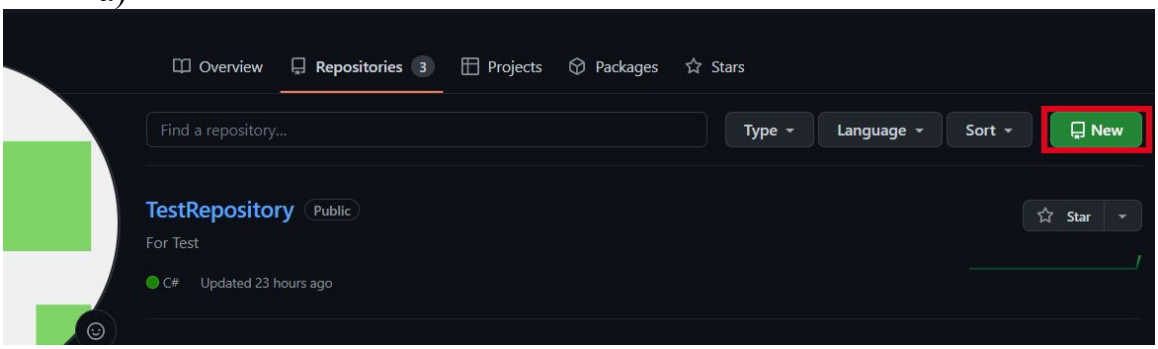
Створення репозиторію в GitHub.com

Після реєстрації на GitHub.com потрібно зайти на головну сторінку у своєму аккаунті, натиснути на кнопку створення нового репозиторію (рис. 3а) та заповнити форму, що з'явиться (рис. 3 б). У вікні «Create new repository» задати ім'я репозиторію (поле Name), опис, якщо потрібно (поле Description), задати модифікатор доступу та натиснути кнопку «Create repository» (рис. 3 б).

Якщо репозиторій було створено на GitHub, то його потрібно клонувати на локальну машину. Для клонування репозиторію існуючого проекту потрібно зайти на сторінку GitHub проекту, обрати кнопку < > Code - Open with GitHub

Desktop (рис. 4а). У вікні «Clone a repository» потрібно задати посилання на репозиторій та вибрати папку на локальному комп'ютері, куди зберігатимуться файли (рис. 4б). При цьому, якщо GitHub Desktop не був встановлений, то з'явиться пропозиція для його встановлення.

а)



б)

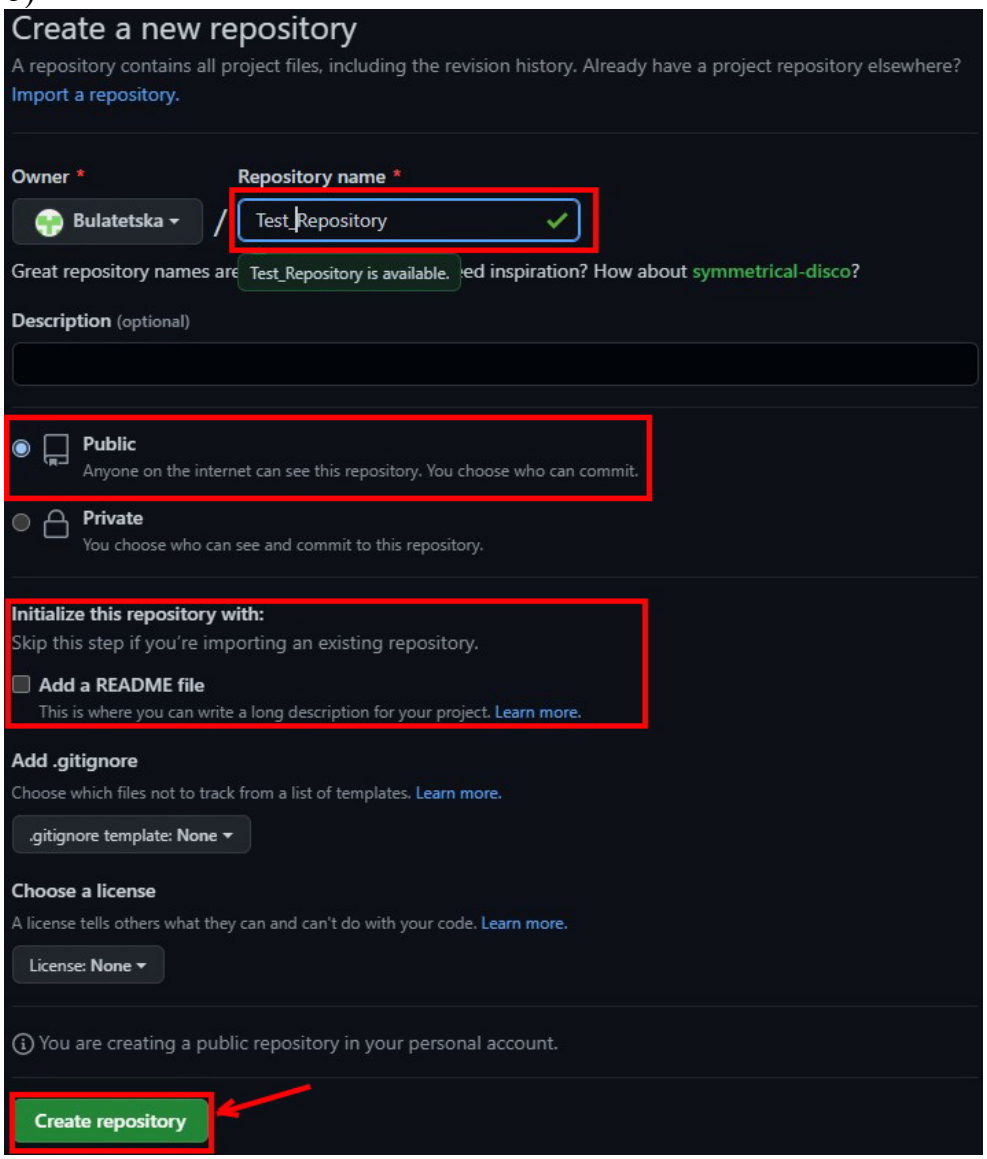
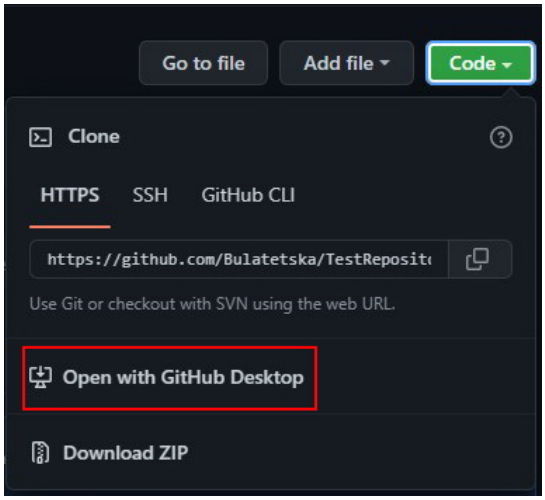


Рис. 3. Створення нового репозиторію на GitHub: а) кнопка New; б) вікно «Create a new repository».

а)



б)

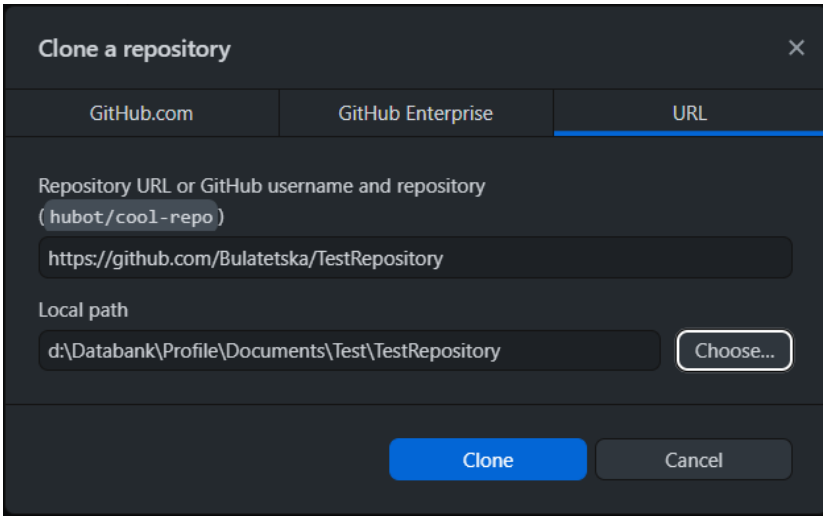


Рис. 4. Клонувати репозиторій на локальну машину: а) кнопка «Code»; б) вікно «Clone a repository»

Створення нового репозиторію в GitHub Desktop

Також новий репозиторій можна створити і в GitHub Desktop. Для цього в вибрати меню File->New repository, або у лівій частині вікна кнопку Add->Create new repository (рис. 5). У вікні «Create new repository» задати ім'я репозиторію (поле Name), опис якщо потрібно (поле Description), шлях до папки (поле Local Path), де буде розміщено репозиторій та натиснути кнопку «Create repository» (рис. 6).

В папці, яку задавали при створенні, появився репозиторій (рис. 7). На даний момент репозиторій розташований локально на комп'ютері в папці Test (рис. 7). Щоб репозиторій з'явився в обліковому записі GitHub і зберігався там, натискаємо «Publish repository» (опублікувати репозиторій) (рис. 8). У вікні, що відкрилося (рис. 9) пункт «Keep this code private» залишаємо вибраним, щоб репозиторій був приватний, потім в будь-який момент репозиторій можна буде зробити відкритим (публічним), щоб його бачили інші користувачі GitHub. Натискаємо «Publish repository». Після цього репозиторій з'явиться у вашому профілі на GitHub.com.

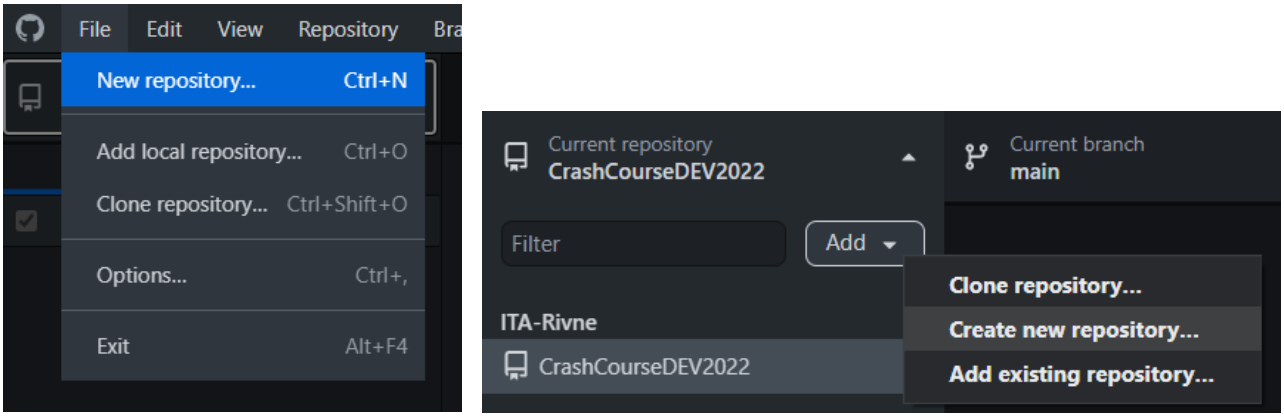


Рис. 5. Створити новий репозиторій в GitHub Desktop

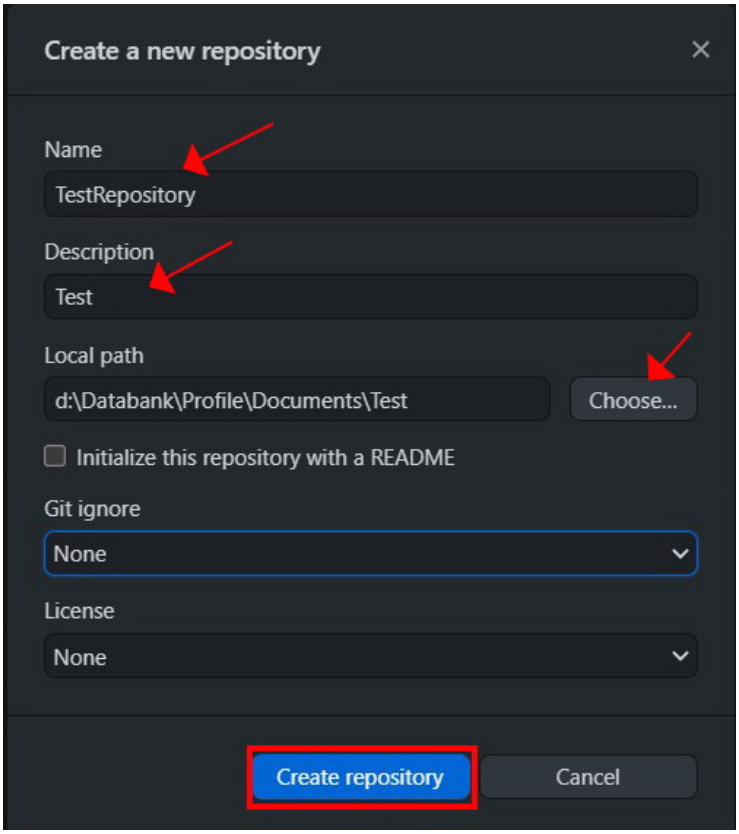
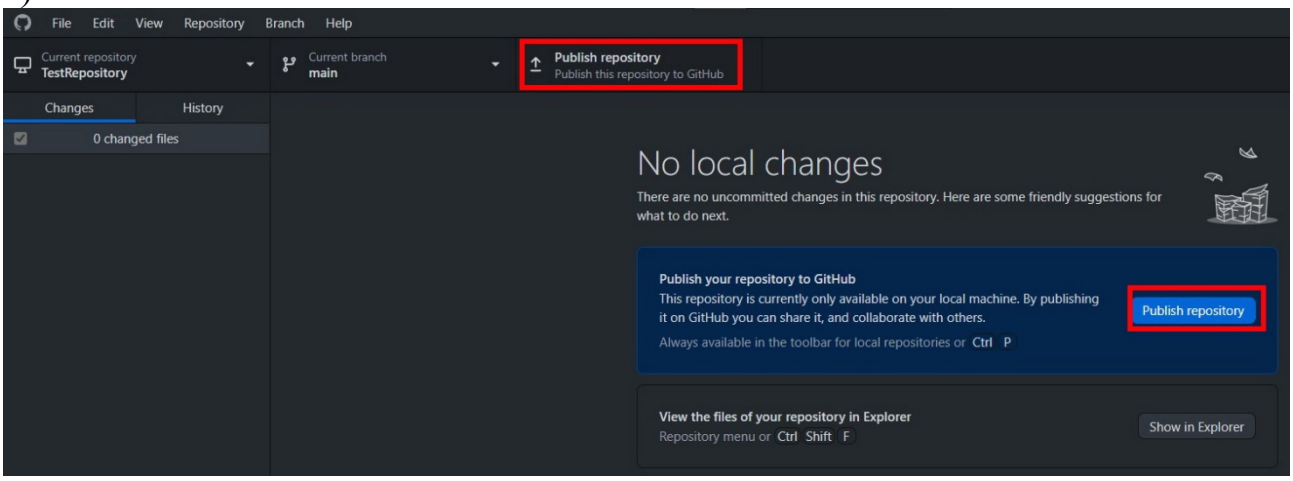


Рис. 6. Задання параметрів при створенні нового репозиторію в GitHub Desktop.

Рис. 7. Вміст папки нового локального репозиторію

а)



б)

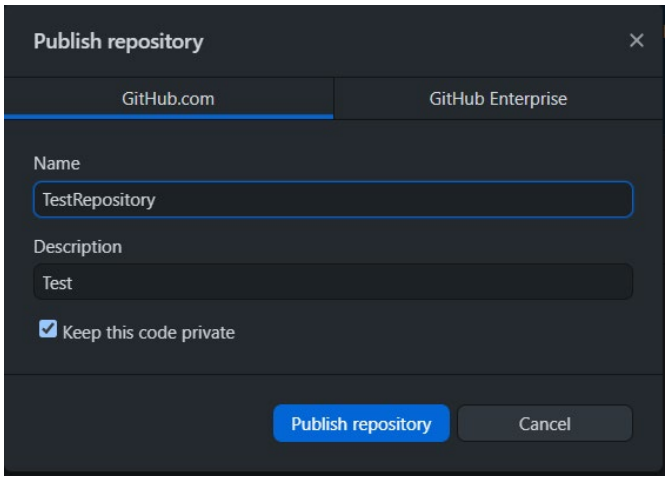


Рис. 8. Публікація нового репозиторію на GitHub.

4. Додавання співавторів

Якщо над проектом працює команда розробників, то потрібно надати всім учасникам право виконувати зміни, тобто додати їх до «співавторів» (collaborators). Для цього всі члени команди повинні мати облікові записи на GitHub. Для того, щоб надати можливість учасникам змінювати дані у сховищі потрібно додати їх до проекту на GitHub. Для цього репозиторій повинен бути публічним (Public). Потрібно зайти в репозиторій на GitHub обрати посилання «Settings», внизу в області «DangerZone» вибрати «Change visibility» і «Make public» (рис. 9). В цьому ж меню «Settings» вибрати «Collaborators» (співавтори) з меню ліворуч і додати співавторів задавши його акаунт (рис. 10). Якщо вам треба скасувати доступ, просто обрати «Remove» з правого боку рядка потрібного користувача.

5. Fork проєкту

Для розробки типових завдань програмісту не завжди потрібно розпочинати розробку з самого початку. Можна знайти схожий проект на GitHub і виконати «Fork» проекту. «Fork» це копія проекту, яка може розроблятися

автономно. При необхідності можна зробити запит на з’єднання (Pull Request) з оригіналом (рис. 11).

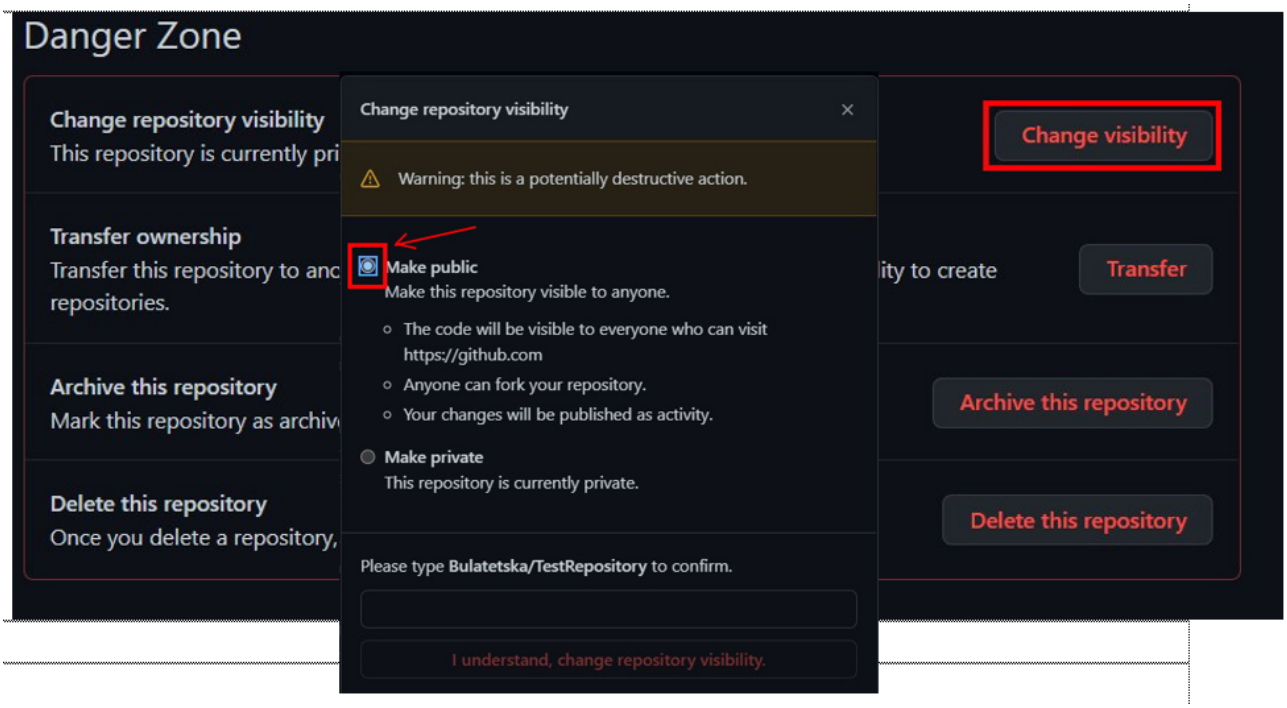


Рис. 9 Зміна модифікатора доступу до репозиторію

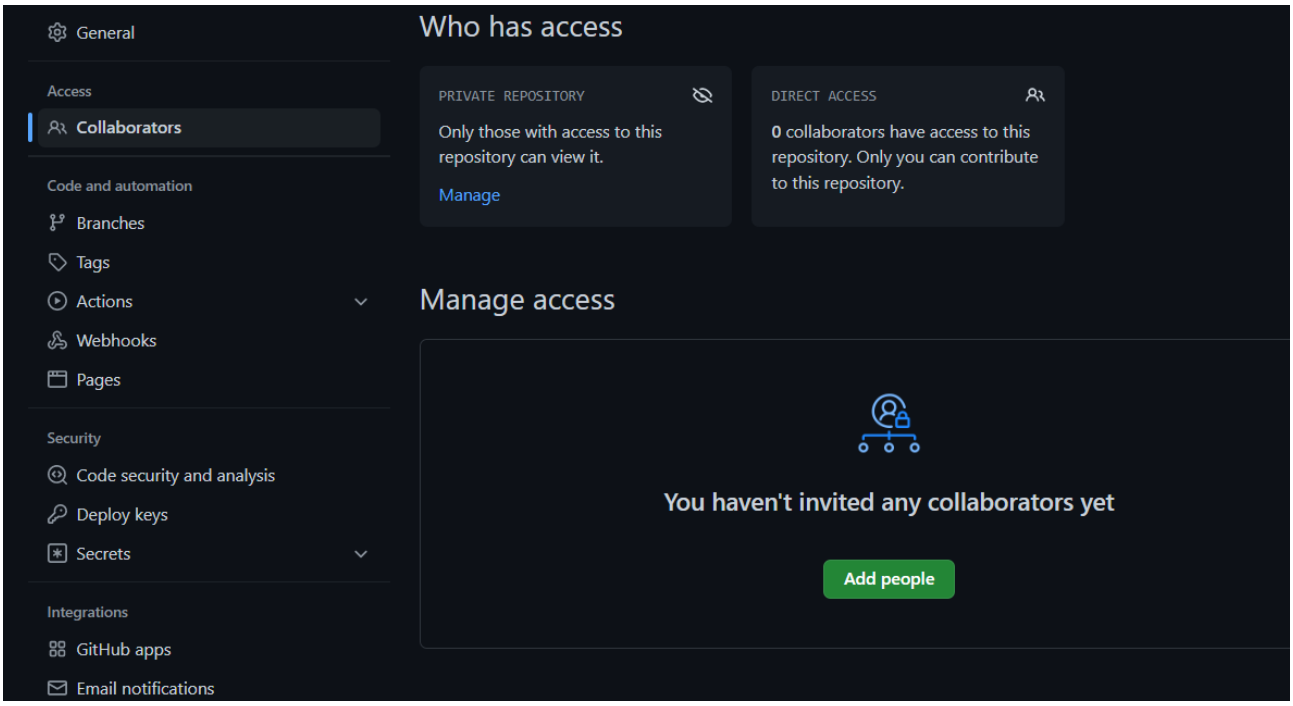


Рис. 10. Додавання нових користувачів.

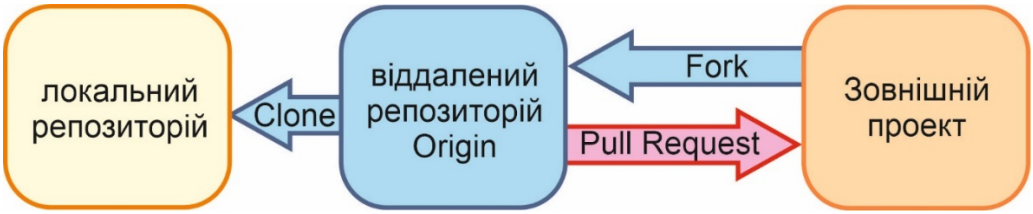
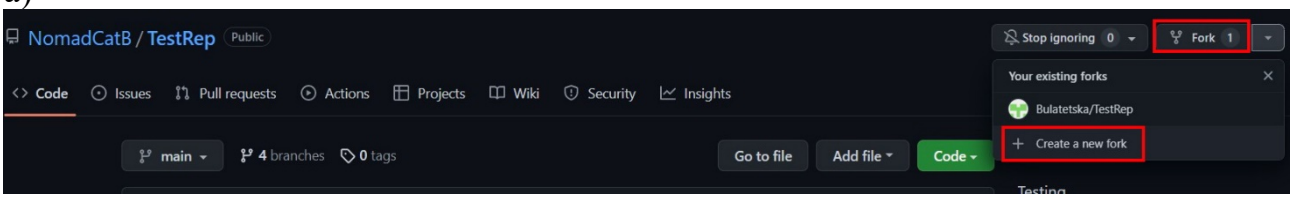


Рис. 11. Використання «Fork» для створення проекту

Етапи створення Fork:

- знайти проект на GitHub;
- натиснути на посилання Fork (рис. 12);
- у GitHub Desktop клонувати проект і вибрати For my own purposes (рис. 13). Після цього проект буде автономним.

а)



б)

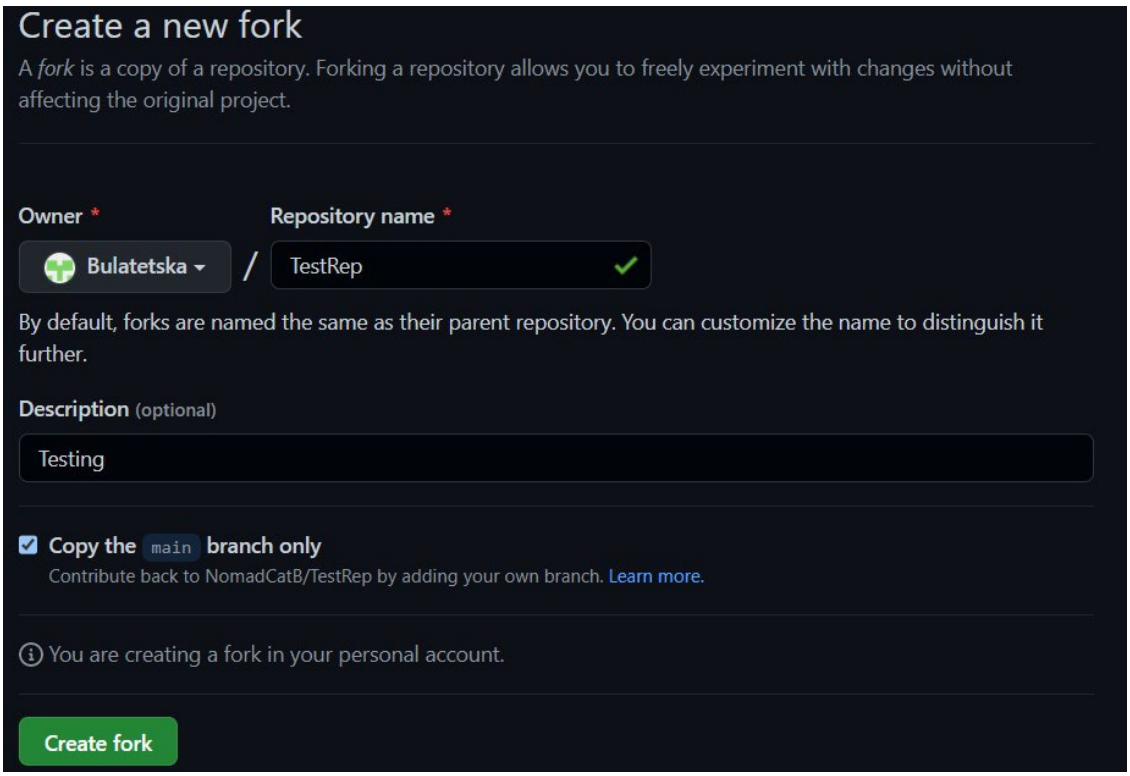


Рис. 12. Створення «Fork»

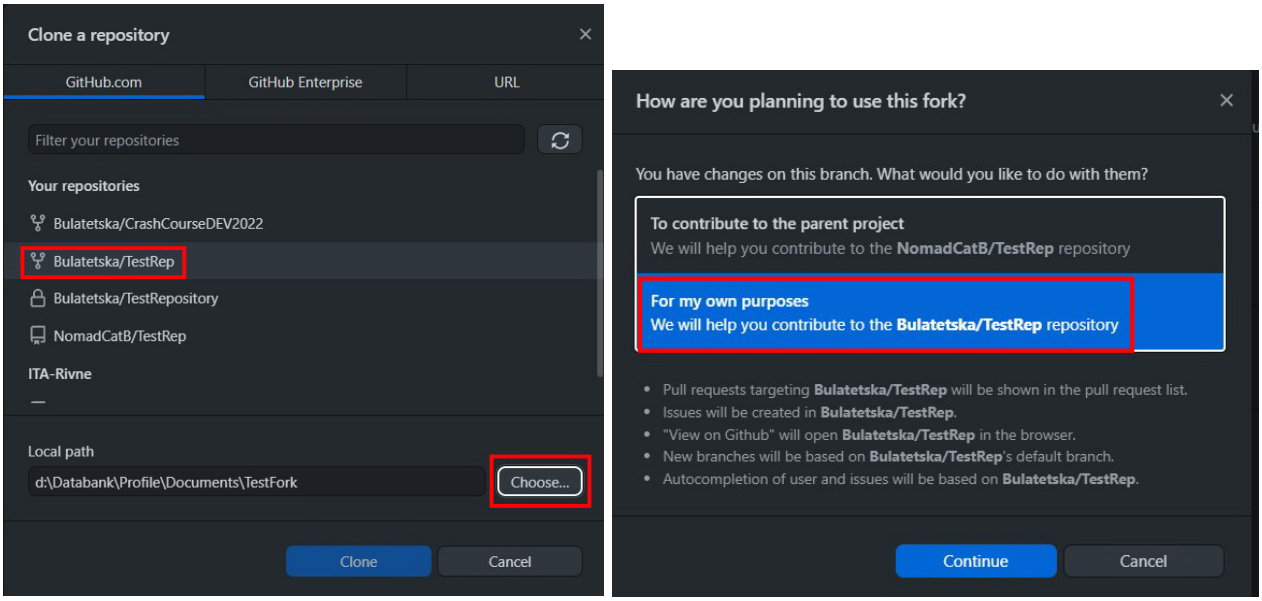


Рис. 13. Клонувати проект Fork у GitHub Desktop.

6. Робота з репозиторієм.

На рис. 14 подано вікно GitHub Desktop, з відкритим репозиторієм, в якому поки що нічого не змінювали незалежно від того, був створений репозиторій чи клонований. Зліва – поле для змінених файлів, праворуч – службова інформація, ліворуч знизу – поле для підтвердження змін.

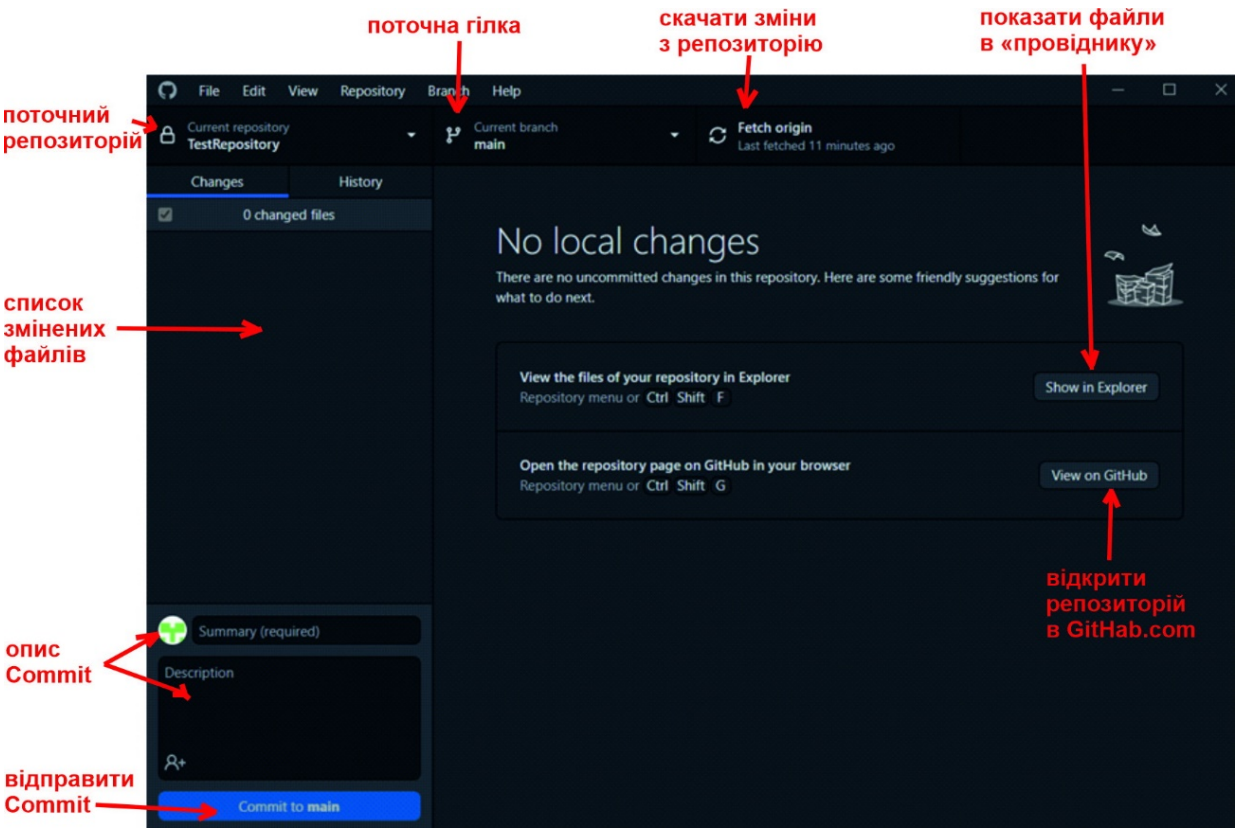


Рис. 14. Вікно GitHub Desktop

Репозиторій просто каталог на комп'ютері (рис. 7). Можна натиснути «Show in Explorer» у Windows і відкриється папка, де лежать усі файли, які є в репозиторії.

Спробуємо додати файл в репозиторій. Наприклад, додамо до локального репозиторію файл test.txt (можна завантажити файл із кодом проекту або змінити вже існуючий). Відразу після додавання або зміни файлу у вікні GitHub Desktop в розділі «Changes» будуть відображені зміни. Якщо ми додали цілий новий файл, то всі рядки в списку змінених файлів будуть із плюсами та зелені. Це означає, що вони були додані до файлу і GitHub Desktop раніше їх ніколи не бачив (рис.15 а). Якщо в списку змінених файлів файли позначені жовтою крапкою, то був змінений існуючий файл (рис.15 б).

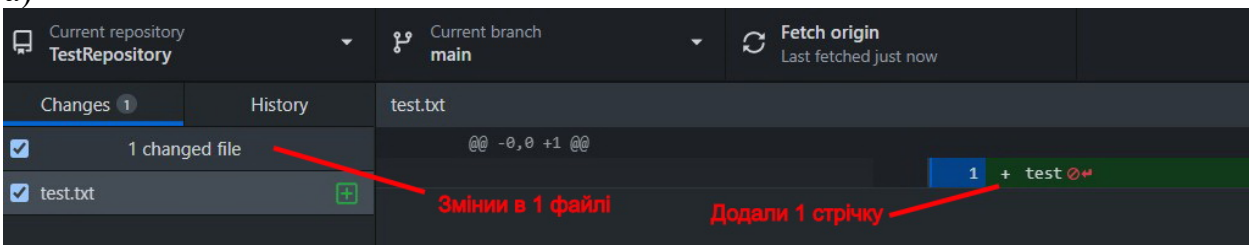
Після того, як було додано якийсь документ або код до репозиторію, потрібно зафіксувати всі зміни і дати їм назву (виконати підтвердження змін «Commit»). Текст назви має бути лаконічним і водночас повідомляти про те, що робить «Commit». Потрібно задати ім'я та натиснути кнопку «Commit to main» (рис. 14). Коли зміни підтверджені, в розділі «Changes» зникає список незафіксованих змін.

Якщо потрібно скасувати зміни: натиснути правою кнопкою миші на галочку навпроти файлу. Відобразиться вкладка Discard changes. Щоб скасувати зміни у всіх файлах – натисніть на верхню галочку, потім Discard all changes (рис. 17). Якщо зміни вже підтверджені, то потрібно зайти на вкладку «History» і відмінити зміни (ліва кнопка миші на «Commit») (рис. 18), а потім відмінити зміни (рис. 17).

Зміни, які були внесені та збережені, поки що локальні. Їх потрібно надіслати на GitHub. Щоб опублікувати зміни в репозиторії на GitHub, натисніть «Push origin» (рис. 19).

Тепер, якщо зайти на GitHub.com, до нашого репозиторію, побачимо вже змінений файл, який ми щойно відправили (рис. 20).

а)



б)

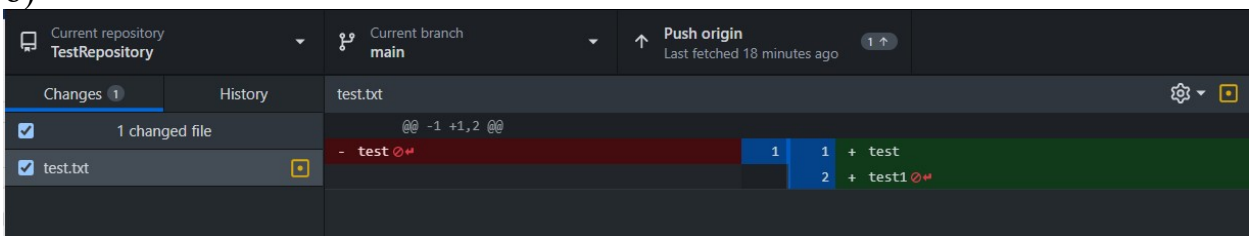


Рис. 15. Відображення змін в локальному репозиторії: а) створення нового файлу, б) зміна існуючого.

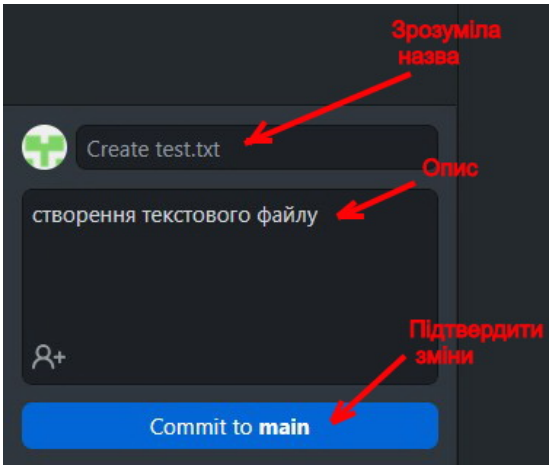


Рис. 16. Підтвердження змін, створення підтвердження змін (Commit)

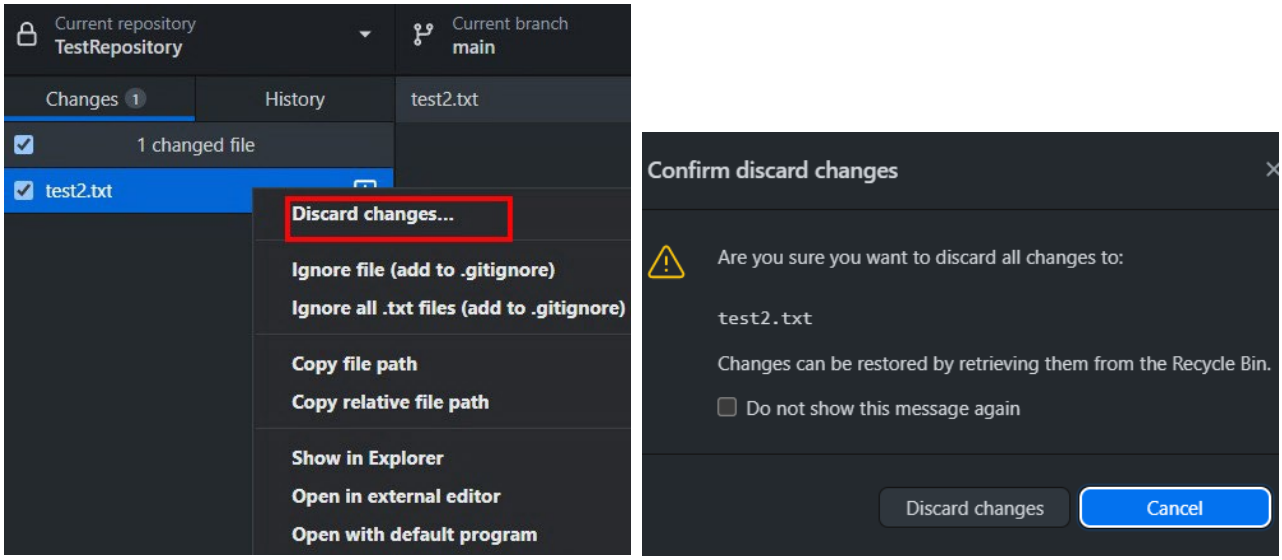


Рис.17. Відміна змін

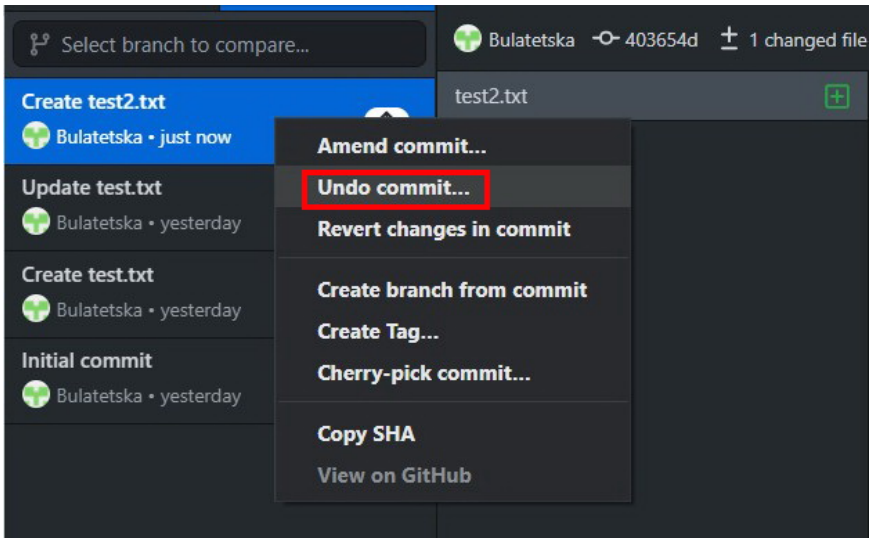


Рис. 18. Відміна підтвердження змін

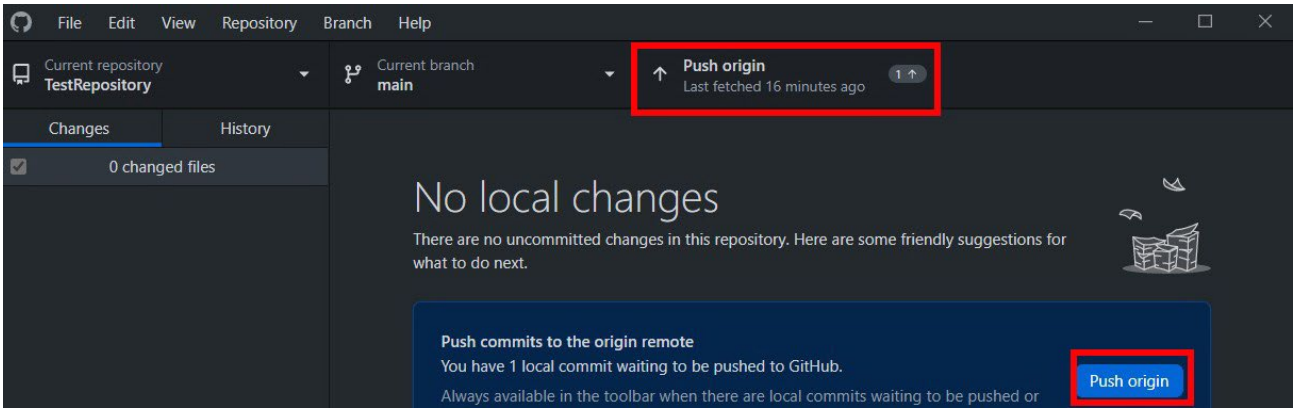


Рис. 19. Надсилання змін на віддалений репозиторій

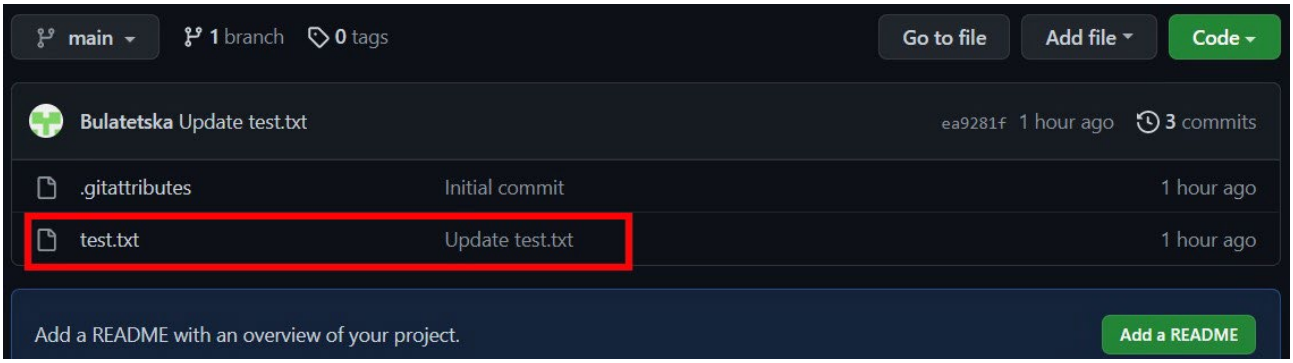
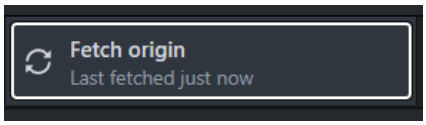


Рис. 20. Відображення доданого файлу в репозиторій

Якщо у віддаленому репозиторії відбулися якісь зміни, то їх потрібно завантажити на локальний репозиторій. Для цього натиснути кнопку «Fetch origin» (рис. 21, а) і ми побачимо (рис. 21, б), що у віддаленому репозиторії були якісь зміни (виконано 1 підтвердження змін «Commit») – кнопка «Pull origin» отримати зміни.

а)



б)

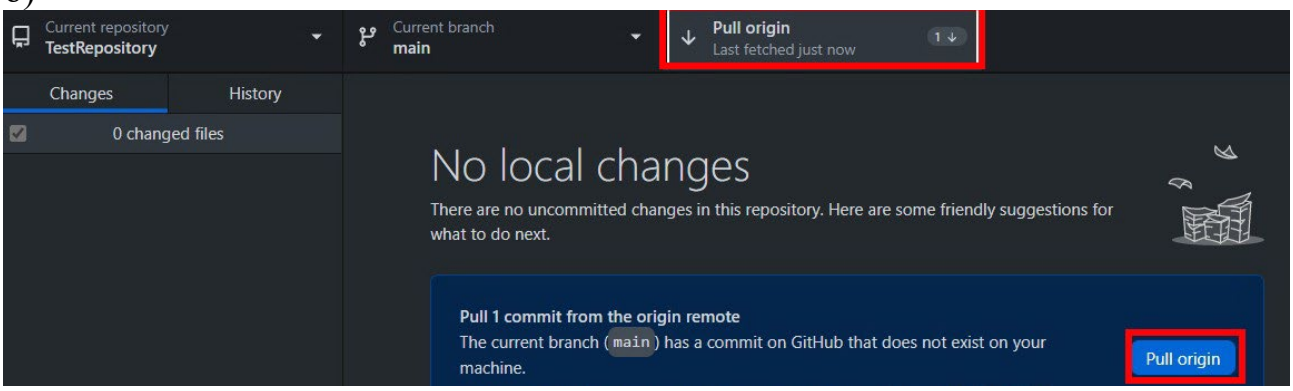


Рис. 21. Отримання змін на локальний репозиторій з віддаленого

Отже, під час роботи з Git найчастіше програміст повинен виконати наступні кроки:

- отримати робочу версію проекту (Pull);
- внести зміни до тексту програми;
- зафіксувати зміни у локальному репозиторії (Commit);
- надіслати всі зафіксовані зміни у віддалений репозиторій (Push);
- при необхідності отримати зміни віддаленого репозиторію (Fetch).

7. Робота з гілками

Git дозволяє створювати кілька різних, паралельних версій проекту – гілок, кожна з яких призначена для різних цілей. Основна гілка проекту – main. У вітці main зберігається остання стабільна версія. Для додавання до проекту нової функціональності необхідно створити окрему гілку, при цьому ці зміни ніяк не відобразяться на основній версії. ***Можна перемикатися між гілками, що призведе до зміни файлів локальної версії репозиторію*** (рис. 22).

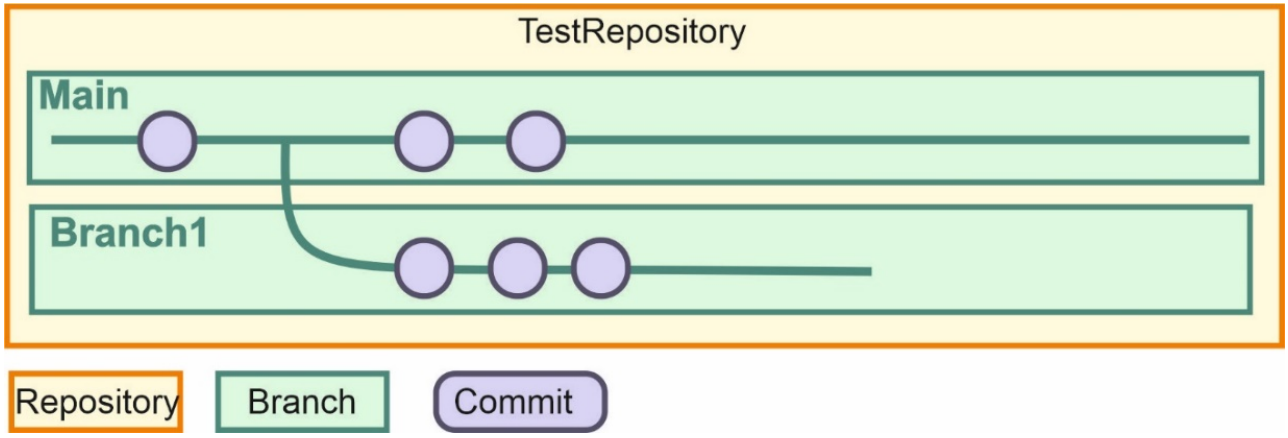


Рис. 22. Гілка main та Branch1 репозиторію TestRepository

Щоб створити нову гілку (рис. 23), натисніть в GitHub Desktop кнопку ліворуч від назви гілки (рис. 24). Нова гілка утворюється від тієї, що вказана (якщо створюється перша гілка, то вона створиться в гілці main); зміни, які є в поточній гілці, але відсутні в гілці main, будуть перенесені в нову гілку. Після створення гілки, її потрібно опублікувати (рис. 24 в).

Перед початком роботи потрібно вибрати гілку, в якій будемо працювати. Після внесення змін в гілці (локального репозиторію) потрібно ці зміни зафіксувати (операція Commit) (рис. 25). Підтверджені зміни («Commit») будуть збережені лише локально.

Тепер потрібно внести зміни у «Fork» (рис. 19) на GitHub (або у віддалений репозиторій) (рис. 26).

Для того, щоб інші учасники команди дізналися про внесені зміни та розглянули їх для внесення в одну з гілок, існує дія під назвою «Pull request» (запит на злиття гілок) (рис. 27). Для цього потрібно вибрати в GitHub Desktop відповідну кнопку (рис. 28 а), перейти на сторінку GitHub.com, обрати до якої

гілки потрібно злити створену гілку, вказати рецензентів (якщо робота в команді) і обрати кнопку, запит на злиття (рис. 28 б). Після відправки запиту на злиття гілок «Pull request», розробники знайомляться зі змінами, залишають коментарі до коду, вносять правки та за необхідності додають підтверджені зміни («Commit»). Поки злиття гілок не відбулося, можна знову додавати підтверджені зміни «Commit» у створену раніше гілку, і вони відобразяться у запиті на злиття «Pull Request». Коли всі зміни будуть узгоджені, гілки зливають (Marge pull request (рис. 29)), «Pull Request» буде закритий і всі зміни будуть у гілці, в яку він був виконаний.

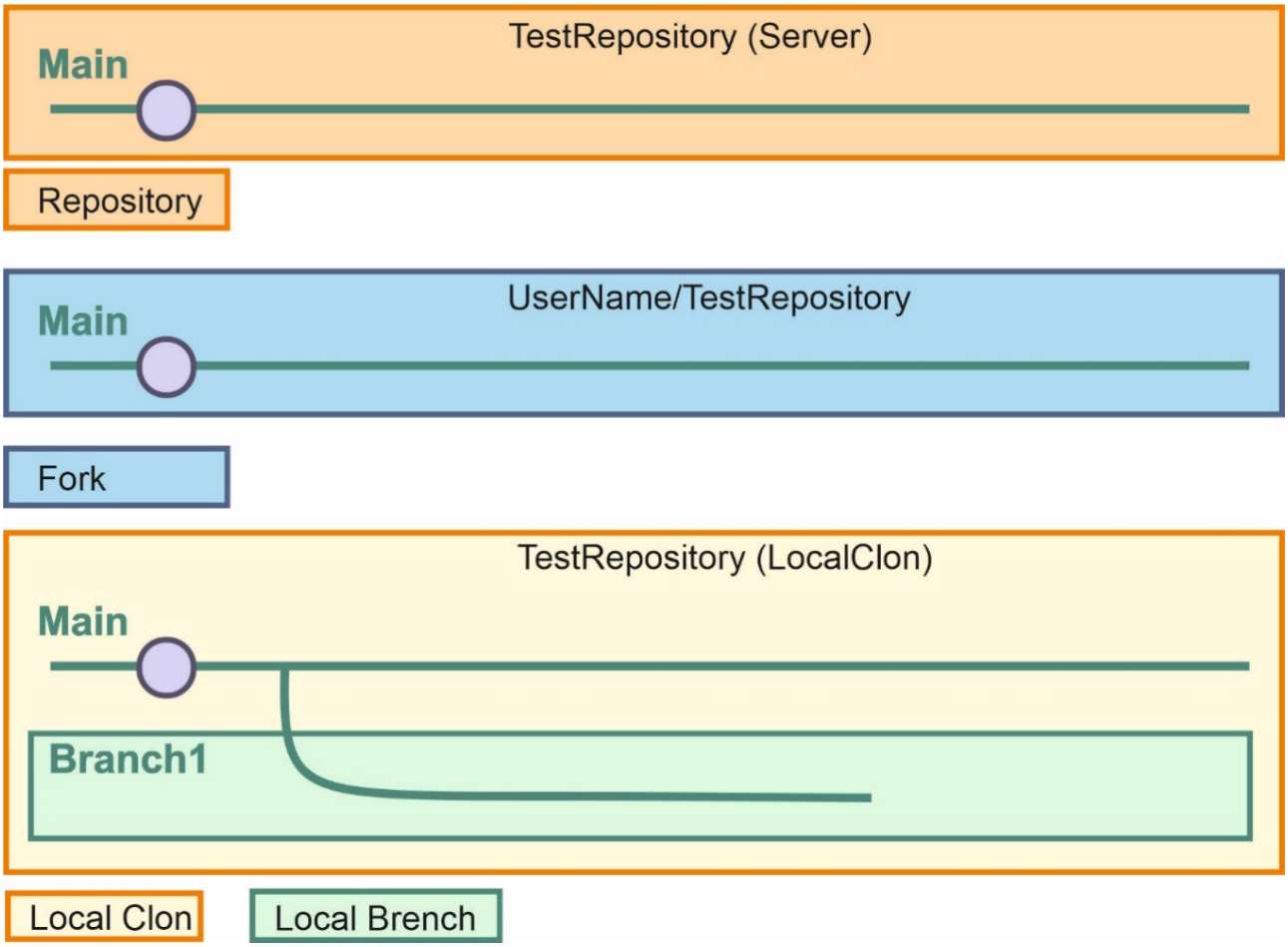
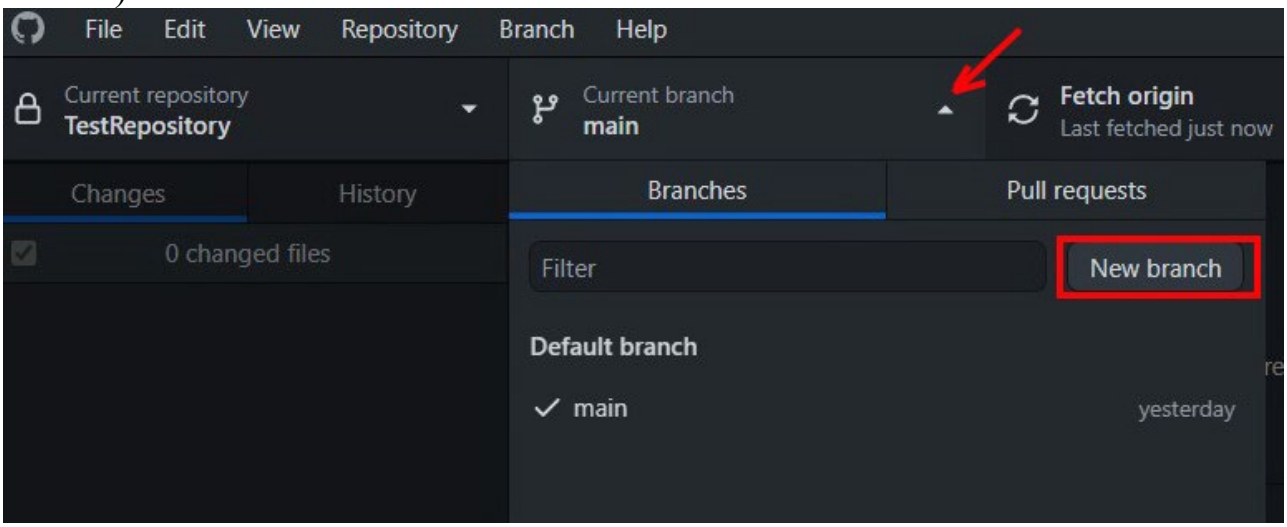


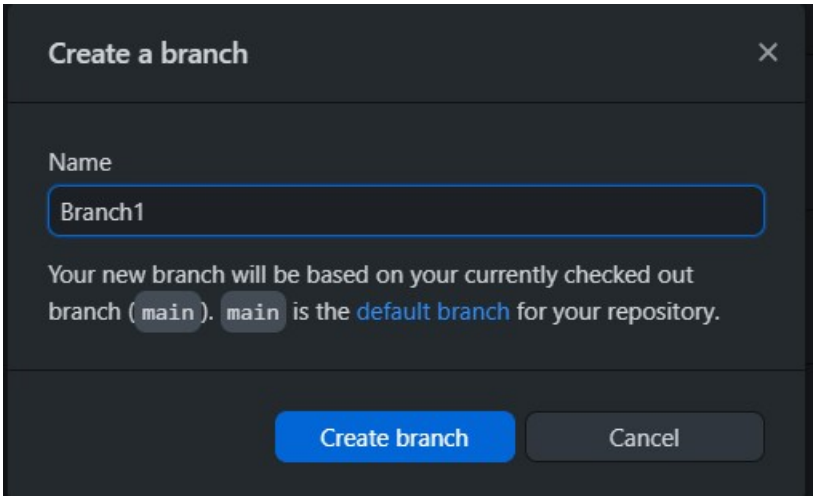
Рис. 23. Створення нової гілки

Під час роботи з гілками варто працювати з останньою актуальною версією, для цього потрібно постійно синхронізовуватися з віддаленим репозиторієм. У локальному репозиторії з’являться останні версії файлів. Якщо не синхронізуватися перед початком роботи, може виникнути ситуація, коли відбулися зміни старої версії файлу. У такому випадку виникає конфлікт змін зі змінами актуальної версії. Поєднувати їх доведеться вручну. Процес цей може бути досить довгим і трудомістким.

а)



б)



в)

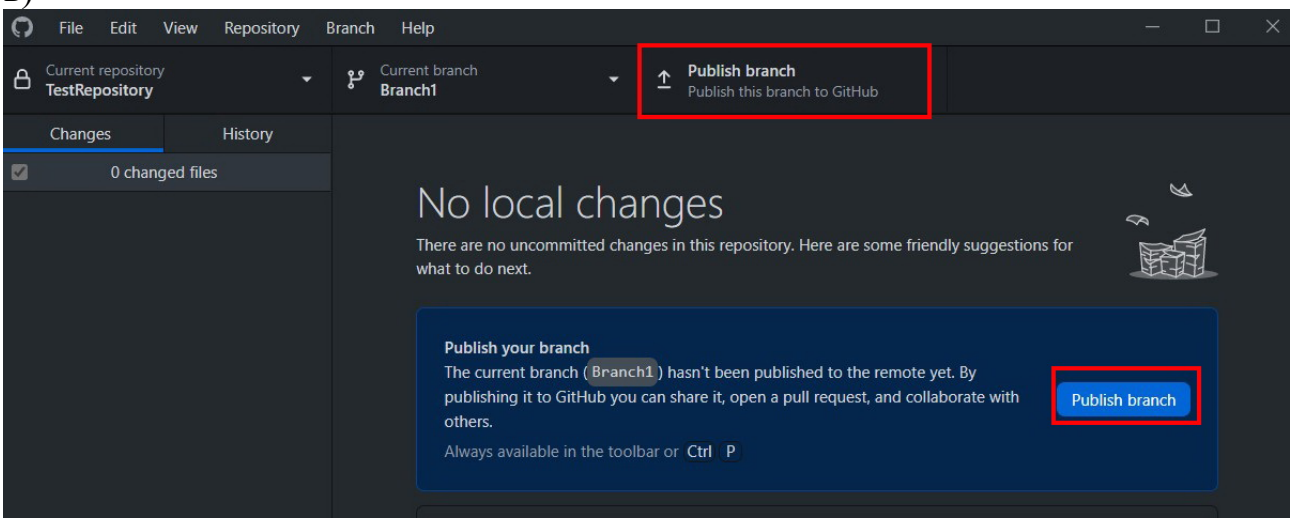


Рис. 24. Створення нової гілки

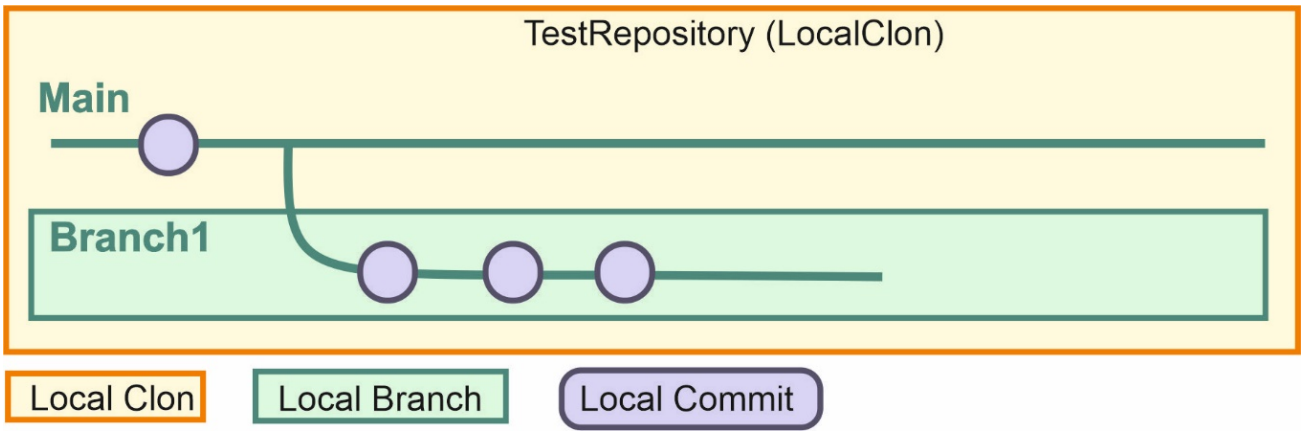


Рис. 25. Внесені та підтверджені зміни у вітці Branch1

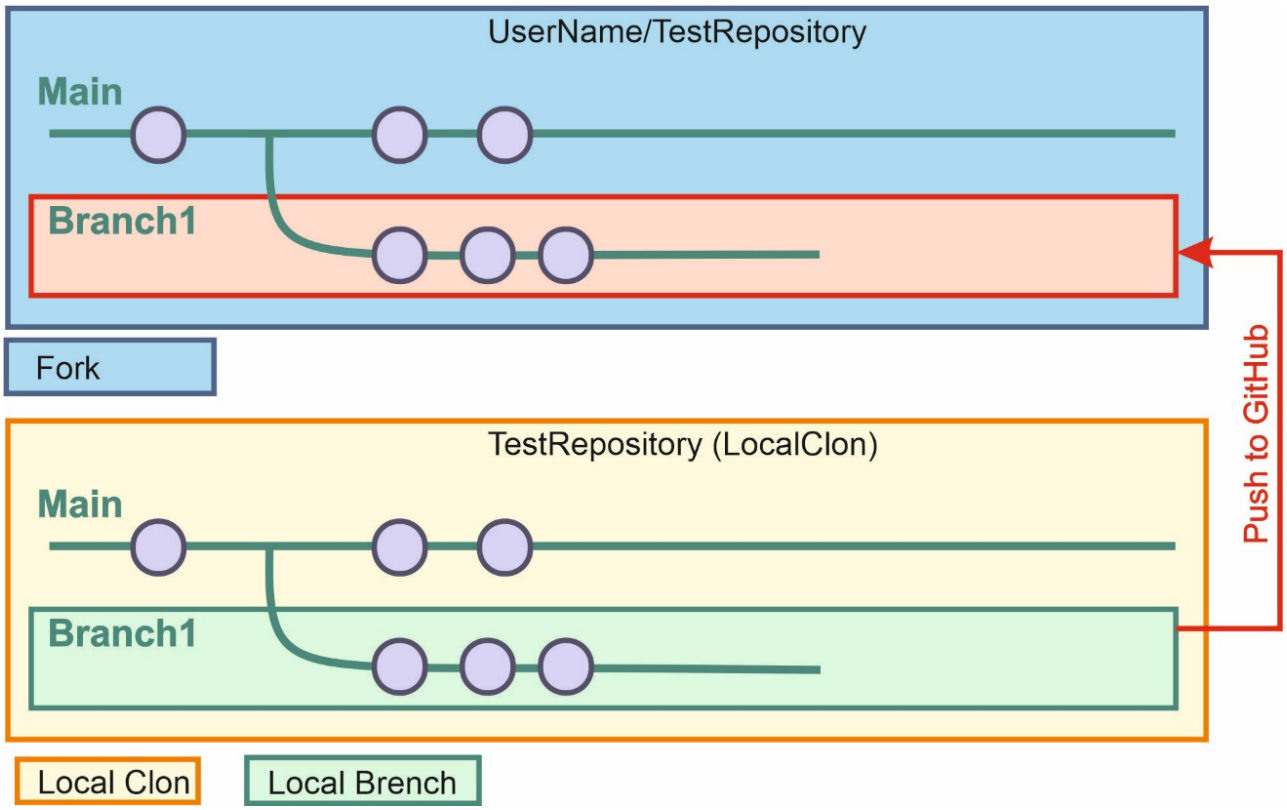


Рис. 26. Внесення змін з локального репозиторію у віддалений

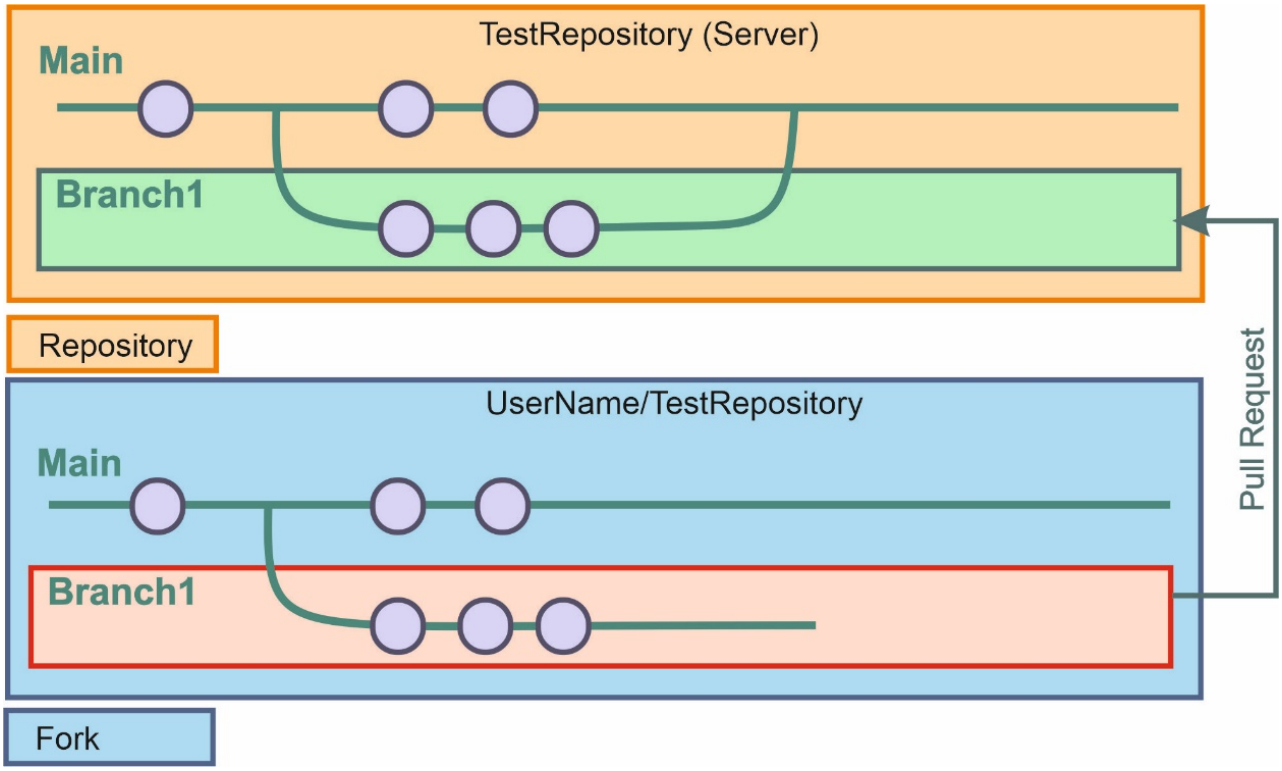
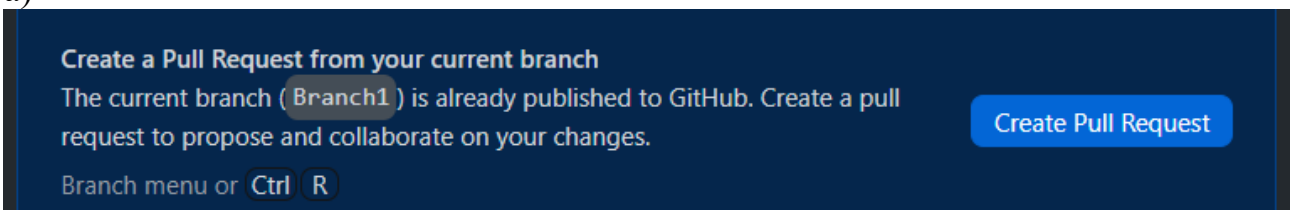


Рис. 27. Запит «Pull Request» на з’єднати гілки Branch1 з основною гілкою main.

а)



б)

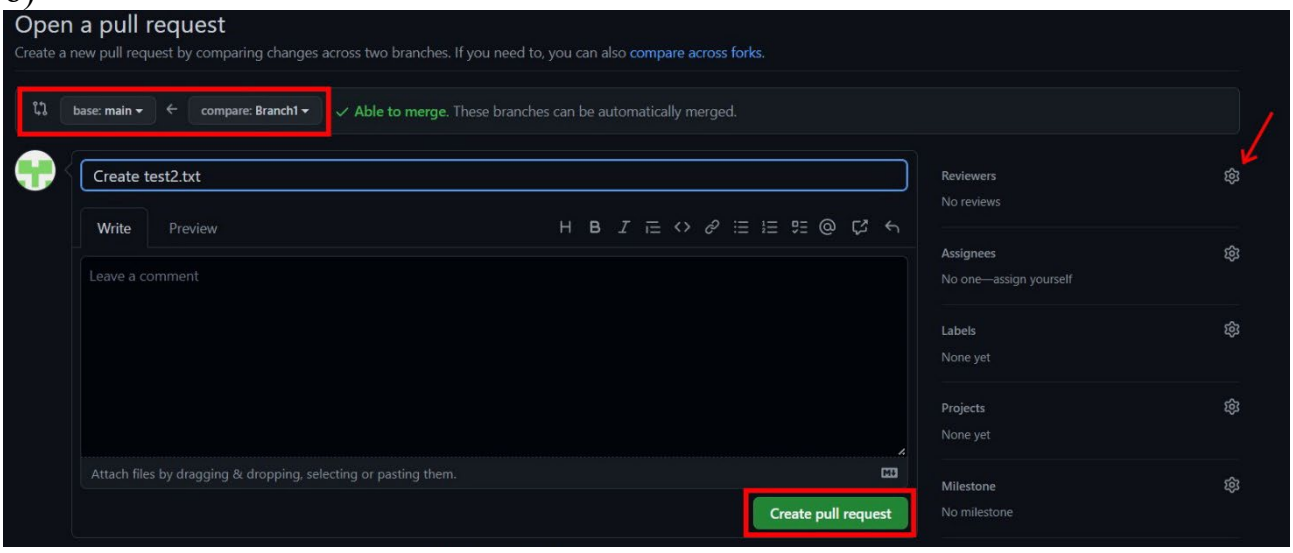


Рис. 28. Запит «Pull Request» на з’єднати гілки Branch1 з основною гілкою main.

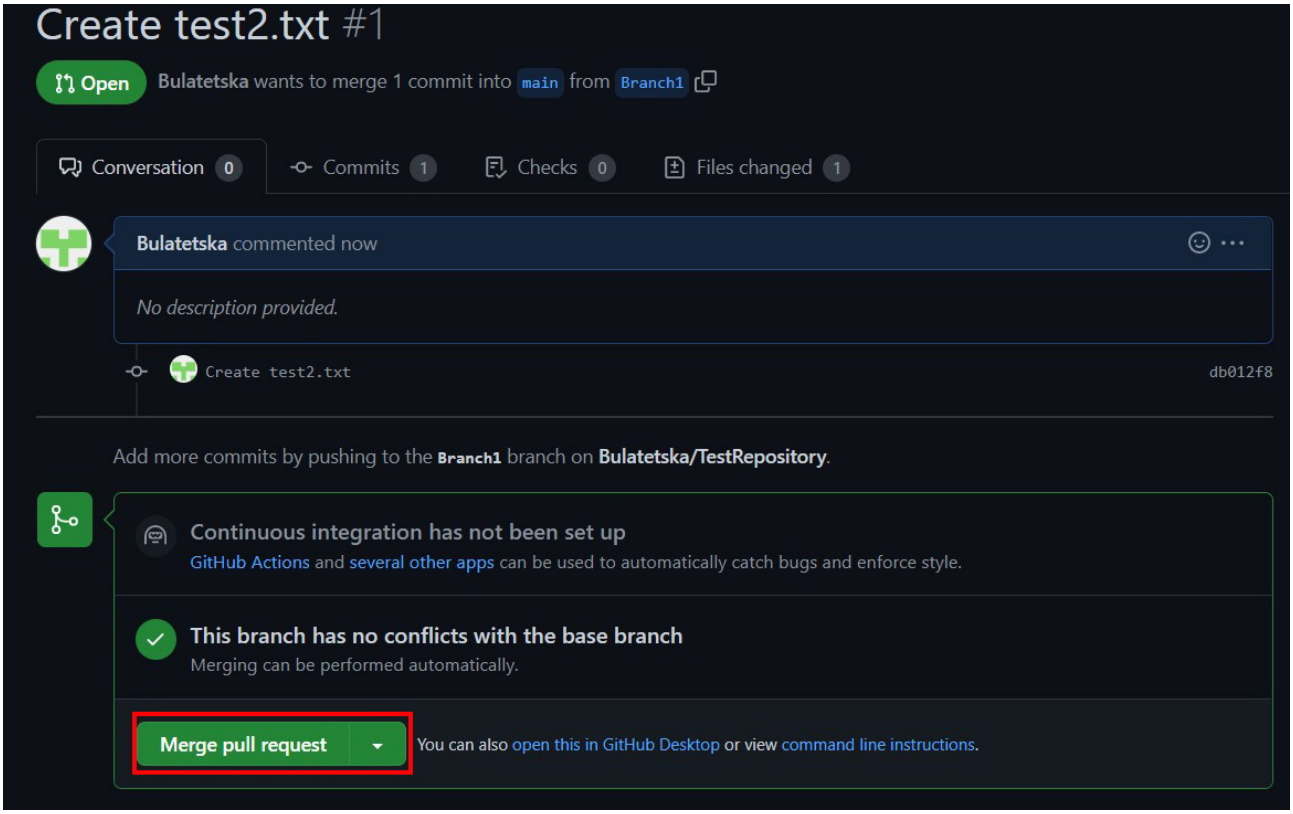


Рис. 29. Підтвердження злиття гілки Branch1 з основною гілкою main

При створенні однієї гілки на базі іншої (рис. 30), потрібно при створенні обрати гілку на базі якої буде створена ця гілка (рис. 31). Коли над проектом працює команда, то схема гілок буде складнішою, наприклад такою, як подано на рис. 32.

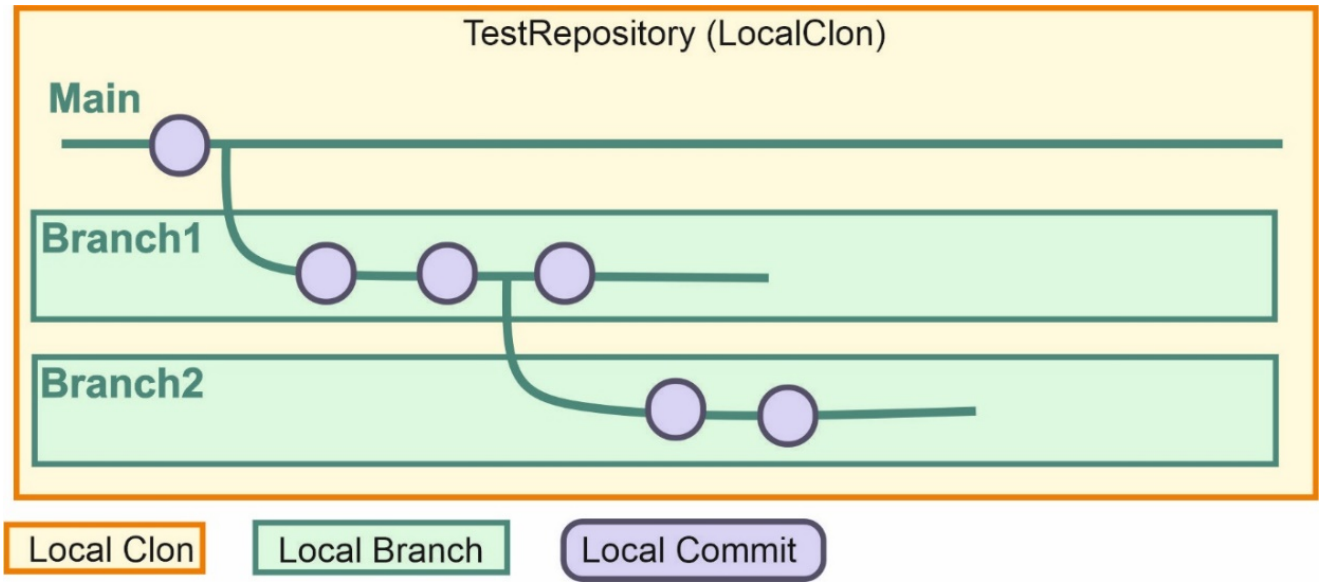


Рис. 30. Гілка Branch2 створена на базі Branch1

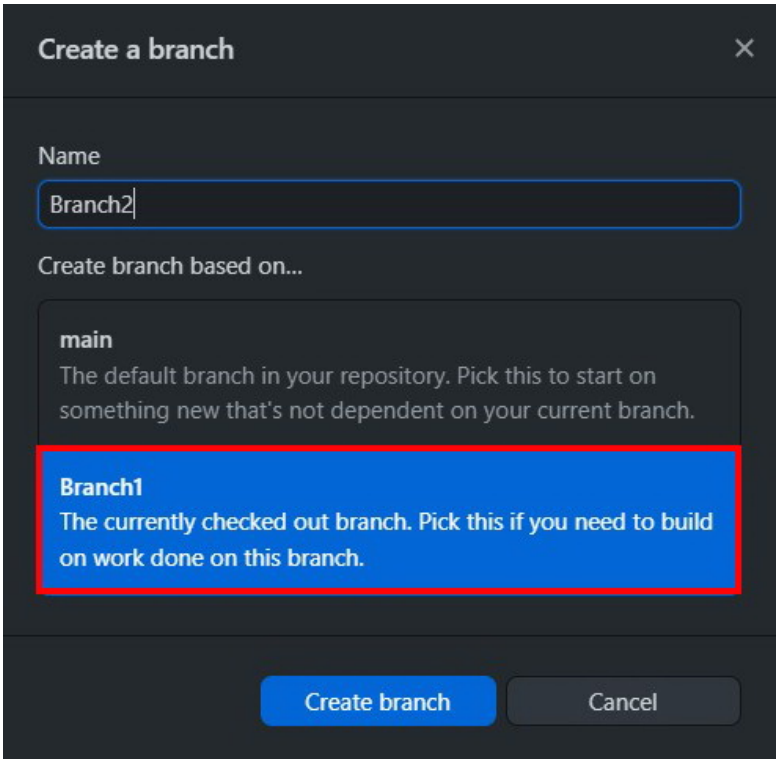


Рис. 31. Створення гілки Branch2 на базі Branch1

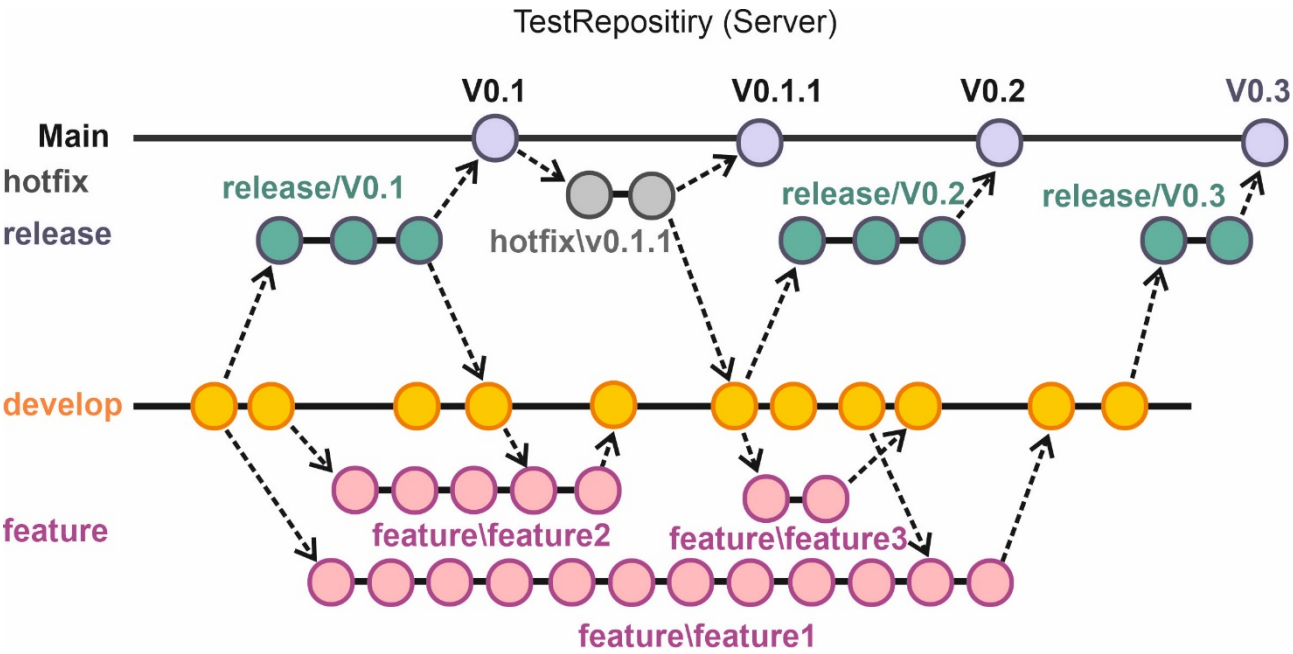


Рис. 32. Приклад гілок командної розробки

За допомогою вкладки History можна вивчити історію розробки проекту: коли і ким були внесені зміни до файлів і рядків, як розвивався проект в цілому.

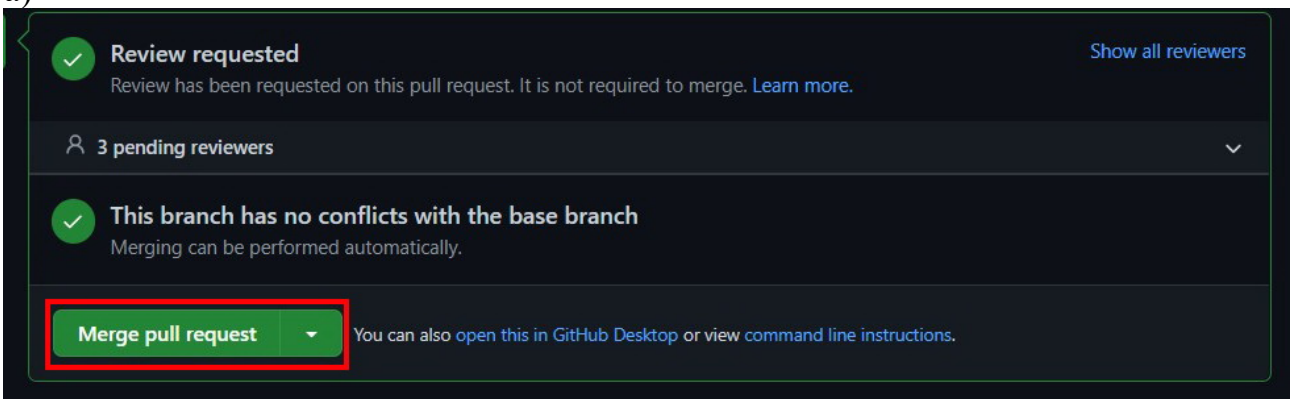
8. Вирішення конфліктів

При роботі над спільним проєктом може статись ситуація, коли зміни були передані файлу у віддаленому сховищі одним членом команди і при цьому також локально було внесено зміни в той самий файл іншим членом команди. У цьому

випадку конфлікти будуть помічені, і їх потрібно буде вирішити, щоб синхронізувати віддалене і локальний сховище.

Для прикладу створимо гілку Brench1 і Brench2. В Brench1 відредагуємо файл test.txt (додамо до test.txt нові рядки і змінимо існуючий текст), підтвердимо зміни (Commit) і відправимо зміни на сервер. Паралельно на віці Brench2 теж відредагуємо файл test.txt (додамо до test.txt нові рядки і змінимо існуючий текст, але не так як на вітці Brench1), підтвердимо зміни (Commit) і відправимо зміни на сервер. Після цього подаємо запит на злиття гілки Brench1 з гілкою main. Так як конфліктів на даному етапі немає (рис. 29 а), то виконаємо злиття (кнопка Marge pull request) (рис. 29 б).

а)



б)

Рис. 29 Запит на злиття гілок

Тепер подаємо запит на злиття гілки Brench2 з гілкою main і у нас появиться повідомлення, що стався конфлікт (рис. 30). Для вирішення конфлікту потрібно обрати кнопку Resolve conflict. З'явиться вікно, де будуть вказані зміни у гілках main і Brench2. Потрібно відредагувати текст (погодившись на зміни, як внесла гілка Brench1, або Brench2, або врахувати зміни кожної гілки) і натиснути кнопку «Mark as resolved».

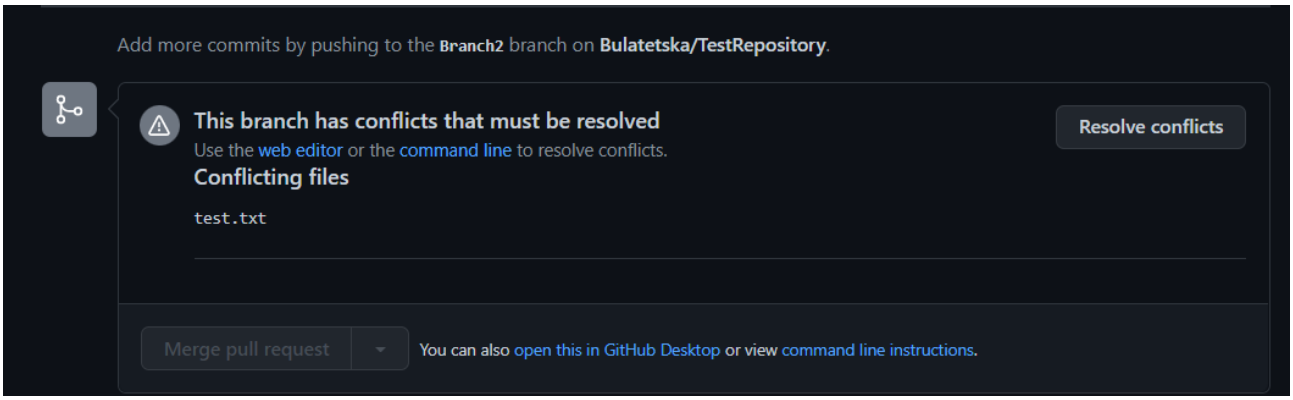
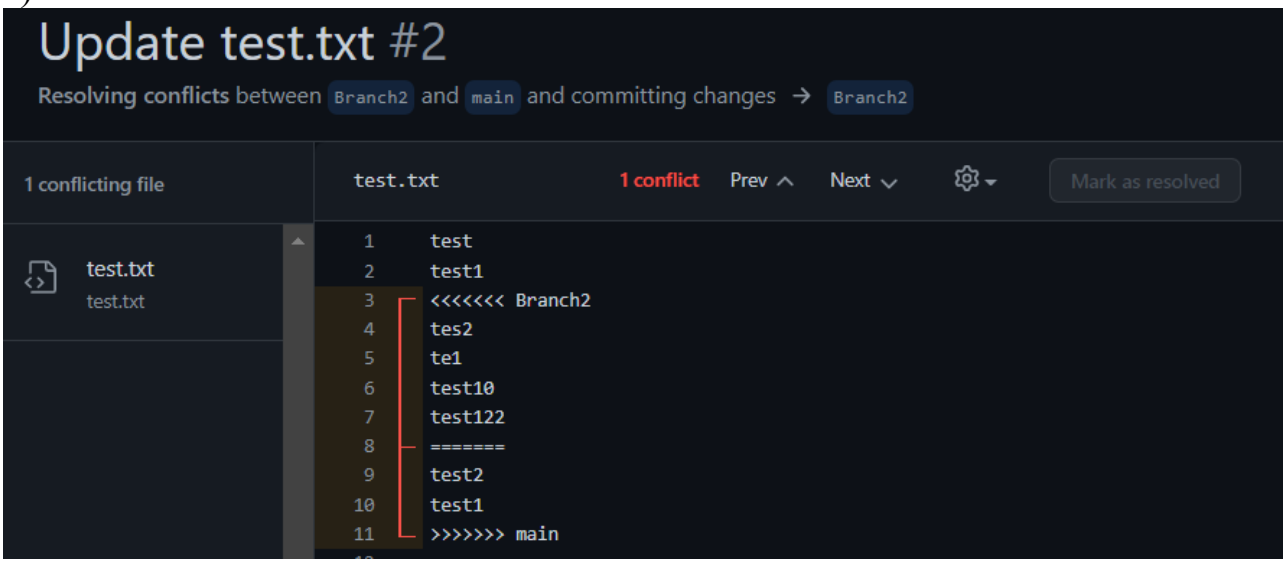


Рис. 30. Повідомлення про конфлікт

а)



б)

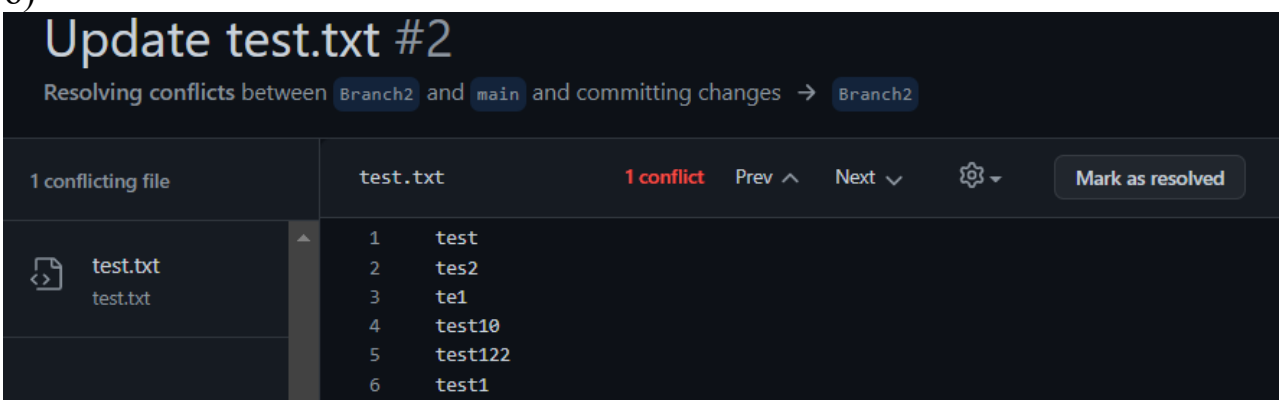


Рис. 31 . Вікно вирішення конфліктів.

Рис. 32. Запит на злиття з врахуванням виправлень

9. Керування версіями у Visual Studio за допомогою Git

За допомогою Git можна легко керувати версіями Visual Studio.

При клонуванні репозиторію або відкритті локального репозиторію Visual Studio перемикається на контекст Git. Оглядач рішень завантажує папку в корені репозиторію Git і перевіряє дерево каталогів на наявність файлів, що переглядаються, наприклад CMakeLists.txt або файлів з розширенням .sln.

Для повсякденного робочого процесу Git Visual Studio надає простий спосіб взаємодії з Git під час написання коду без необхідності залишати свій код.

Створення гілки Git у Visual Studio. У меню Git виберіть «New Branch…» рис. 33. У діалоговому вікні «Create a new branch» введіть назву гілки.

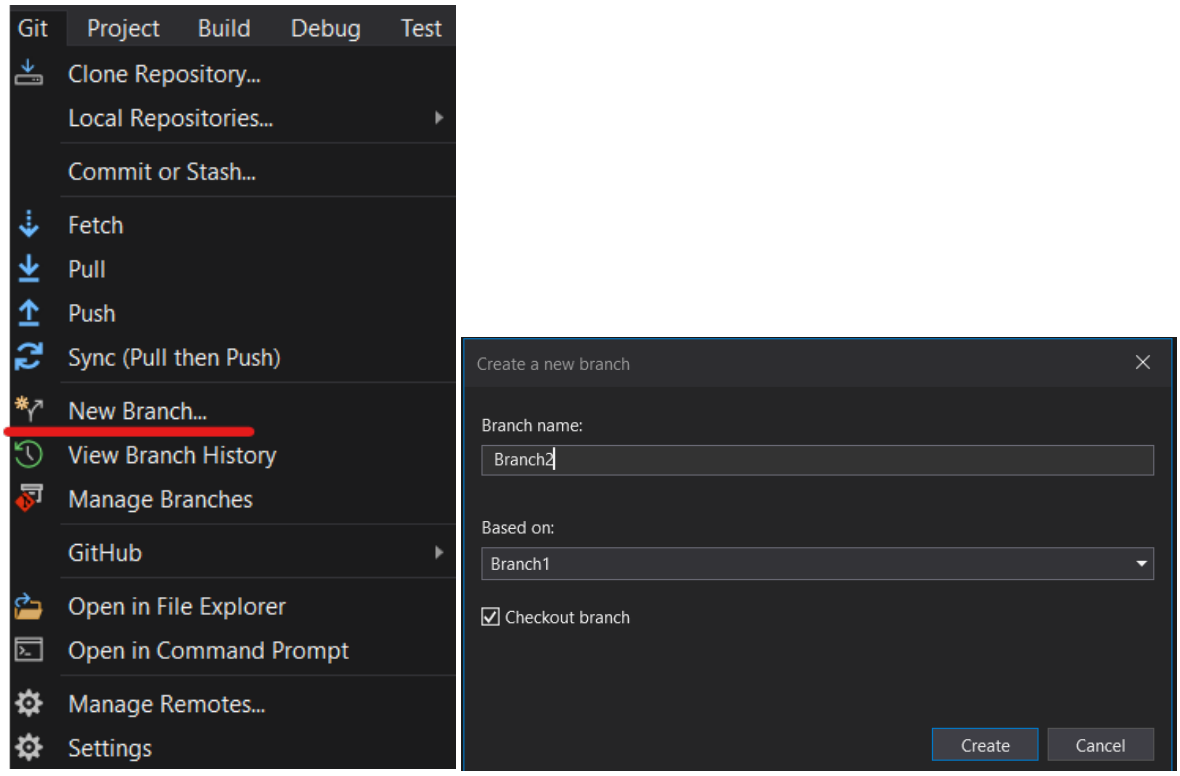


Рис. 33. Створення гілки Git у Visual Studio.

У розділі «Based on:» можна вибрати, гілку, на базі якої буде створено нову. Прапорець «Checkout branch», який увімкнено за замовчуванням, автоматично перемикається на щойно створену гілку. Потрібно встановити цей прапорець, якщо потрібно залишитися в поточній гілці.

Після створення нової гілки та перемикання на неї можна розпочати роботу, змінивши існуючі файли або додавши нові, а потім зафіксувавши свою роботу в репозиторії.

Створення фіксації Git у Visual Studio. Основна частина будь-якого робочого процесу Git полягає у зміні файлів та фіксації змін у цих файлах. Git відстежує зміни файлів у репозиторії в процесі роботи та поділяє файли на три категорії.

Файли без змін: ці файли не були змінені після останньої фіксації.

Змінені файли: ці файли змінилися з моменту останньої фіксації, але ще не були підготовлені до наступної фіксації.

Проміжні файли: ці файли містять зміни, які будуть додані до наступної фіксації.

Під час роботи Visual Studio відстежує зміни у файлах проекту у розділі «Commit or Stash…» вікна Git Changes.

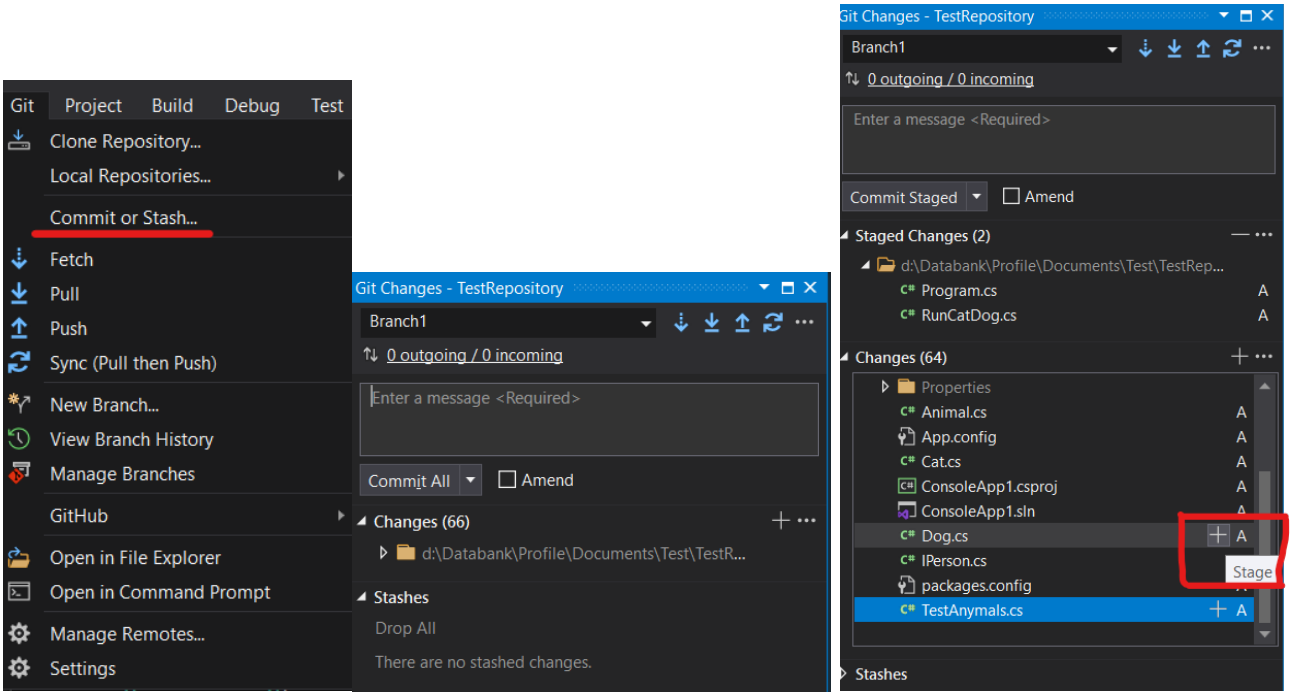


Рис. 34. Вікно відстеження змін у файлах проєкту

Щоб підготувати зміни, натисніть кнопку + (плюс) для кожного файлу, який потрібно підготувати, або виберіть правою кнопкою миші файл та виберіть «Stage». Можна також підготувати всі змінені файли одним клацанням миші, використовуючи кнопку «+» у верхній частині розділу «Changes».

Під час підготовки змін Visual Studio створює розділ «Staged Changes». Тільки зміни в розділі «Staged Changes» додаються до наступної фіксації, яку можна виконати, вибравши команду «Commit Staged».

Також можна скасувати підготовку змін, натиснувши кнопку – (мінус).

Крім того, можна відмовитись від підготовки змінених файлів, пропустивши область підготовки. У цьому випадку Visual Studio дозволяє зафіксувати зміни безпосередньо без попередньої підготовки. Просто введіть повідомлення про фіксацію та виберіть «Commit All».

Visual Studio також дозволяє виконати фіксацію та синхронізацію одним клацанням миші за допомогою ярликів «Commit Staged and Push» та «Commit Staged and Suns».

Якщо двічі клацнути будь-який файл у розділі «Changes» або «Stashes», можна порівняти змінену версію файлу з незміненою (рис. 35).

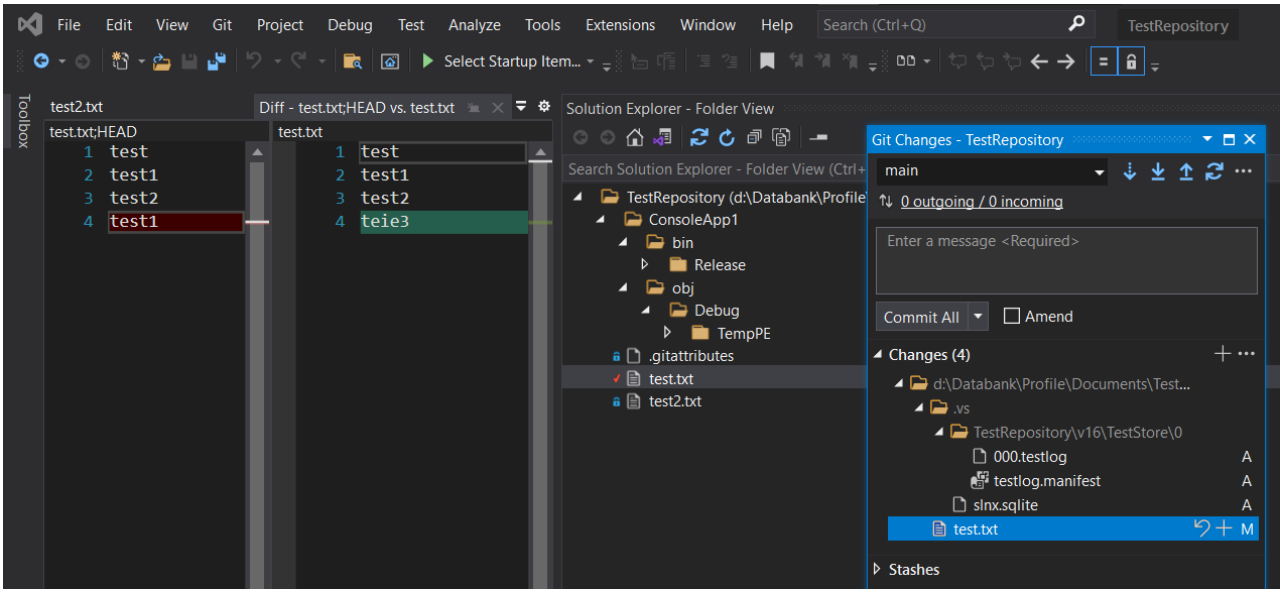


Рис. 35. Вікно порівняння змін у файлі.

Відправлення змін з Visual Studio у віддалену гілку. У вікні «Git Changes» знаходиться текст посилання, що включає кількість вхідних та вихідних підтверджених змін. На рис. 36 текст посилання зчитує 1 вихідний / 0 вхідний (1 outgoing/0 incoming).

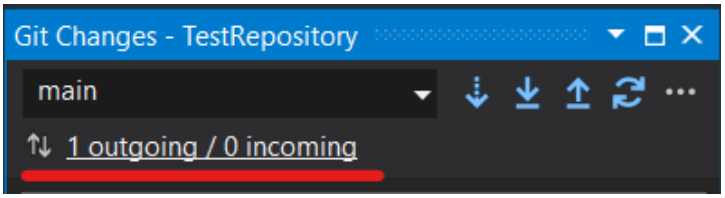


Рис. 36. Текст посилання, що включає кількість вхідних та вихідних підтверджених змін

Вихідний текст представляє кількість підтверджених змін, які ще не були відправлені у віддалений репозиторій, а текст вхідний представляє кількість підтверджених змін, які були отримані, але ще не вилучені з віддаленого репозиторію. Якщо натиснути на текст посилання, то відкриється вікно

відомостей про фіксацію внесених змін (рис. 37). Якщо двічі клацнути фіксацію, у Visual Studio відкриються відомості про неї в окремому вікні інструментів. Тут можна скасувати фіксацію, скинути її, виправити повідомлення про фіксацію або створити тег для неї.

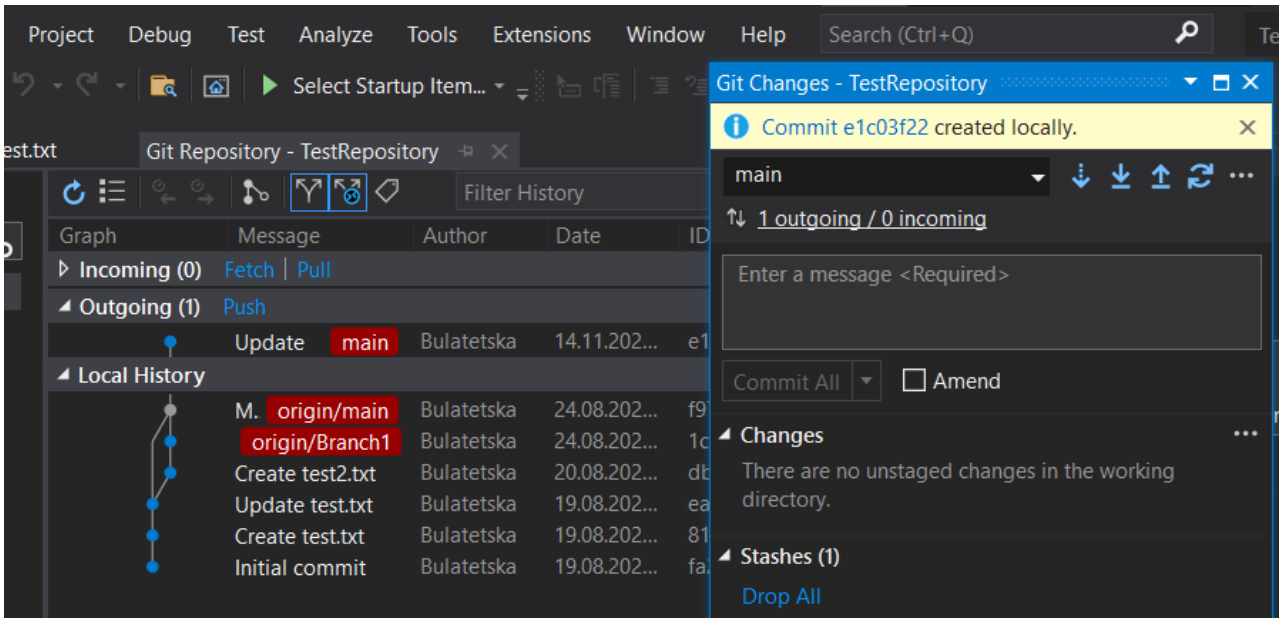


Рис. 36. Відомості про фіксацію змін внесених у файл.

Git – це розподілена система управління версіями, тобто всі зміни, які були вже внесені, є локальними. Щоб внести ці зміни до віддаленого репозиторію, необхідно надіслати локальні підтверджені зміни на віддалений вузол.

Visual Studio допомагає синхронізувати локальну гілку з віддаленою гілкою за допомогою операцій скачування (отримання та вилучення) та передачі (надсилання). Щоб надіслати у віддалений репозиторій, потрібно вибрати «Push» в меню Git. Щоб отримати зміни віддаленого репозиторію, потрібно обрати «Fetch». Щоб вилучити зміни на локальний репозиторій, потрібно обрати «Pull».

Для виконання цих операцій можна також використовувати елементи керування кнопкою у вікні змін Git (рис. 38).

Зліва направо елементи керування кнопкою включають отримання (Fetch), вилучення (Pull), відправлення (Push) та синхронізацію (Sync).

Крім того, існує також елемент керування три крапки (**...**) для додаткових операцій (рис. 39). При виборі відображається контекстне меню. Його можна використовувати для точного налаштування операцій отримання, вилучення, надсилання та синхронізації.

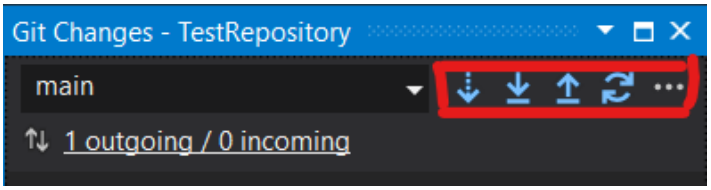


Рис. 38. Елементи керування у вікні змін Git.

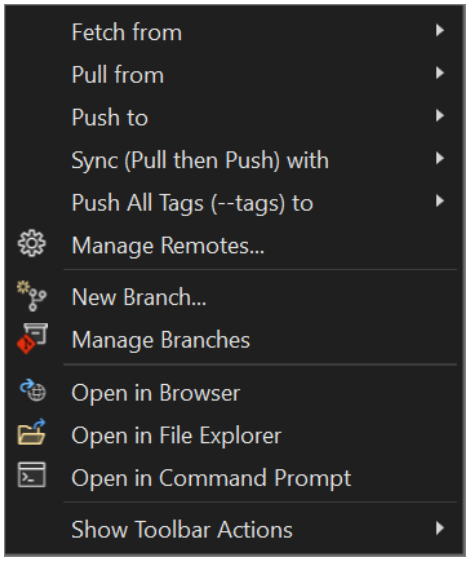


Рис. 39. Контекстне меню елемента керування трьома крапками (**...**) для додаткових операцій.

Запитання для самоконтролю

1. Що таке система контролю версій?
2. Що таке локальні системи контролю версій?
3. Які недоліки локальних систем контролю версій?
4. Що таке централізовані системи контролю версій?
5. Де зберігаються файли при використанні централізованої системи контролю версій?
6. Які переваги дає використання централізованої системи контролю версій?
7. Які недоліки має використання централізовані системи контролю версій ?
8. Що розуміється під керуванням доступом до файлів в централізованій системі контролю версій?
9. Що таке розподілена система контролю версій?
10. Де зберігаються файли при використанні розподіленої системи контролю версій?
11. Які переваги дає використання розподіленої системи контролю версій?
12. Які недоліки має використання розподіленої системи контролю версій?
13. Навіщо потрібні розподілені системи контролю версій?
14. До якої системи контролю відноситься Git?
15. Що являє собою репозиторій?
16. Що таке гілка в розподіленій системі контролю версій?
17. Чим відрізняється клон (Clone) репозиторію від копії (Fork)?

Список використаних джерел

1. Version Control Systems - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/version-control-systems/ (дата звернення: 14.09.2022).
2. What Is Git | Explore A Distributed Version Control Tool | Edureka. *Edureka*. URL: https://www.edureka.co/blog/what-is-git/ (дата звернення: 14.09.2022).
3. How to Git(Hub) Sponge 8.0.0. *Sponge Documentation*. URL: https://docs.spongepowered.org/stable/ru/contributing/howtogit.html (дата звернення: 14.09.2022).
4. Підручник з робочого столу GitHub - Співпрацюйте з GitHub зі свого робочого столу - Інший. *Огляди, Ігри, Розваги, Листопад 2022*. URL: https://uk.myservername.com/github-desktop-tutorial-collaborate-with-github-from-your-desktop (дата звернення: 14.09.2022).
5. Що таке Git та Github – керівництво для початківців. *SebWeo*. URL: https://sebweo.com/scho-take-git-ta-github-kerivnitstvo-dlya-pochatkivtsiv/ (дата звернення: 14.09.2022).
6. The Git experience in Visual Studio. *Microsoft Learn: Build skills that open doors in your career*. URL: https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2022 (дата звернення: 14.09.2022).

УДК 004.655

Електронне мережне навчально-методичне видання

Булатецька Леся Віталіївна

Булатецький Віталій Вікторович

Система контролю версіями Git
методичні рекомендації до організації виконання лабораторних робіт з освітнього компонента «Технології платформи .Net».