

ЛАБОРАТОРНА РОБОТА

Тема: Основні операції в системі контролю версій Git.

Мета: Ознайомитись та навчитись використовувати основні команди системи контролю версій Git.

Теоретичні відомості

Під час розробки програмного забезпечення дуже важливо робити резервну копію рішення (solution), щоб у надзвичайних випадках можна було відновити та продовжити роботу.

Рішення представляє собою інформаційну систему, яка може складатись з модулів (проектів), які вирішують певну проблему.

Найпростішим способом створення резервної копії є **архівування** файлів рішення, та копіювання архіву в інше місце (наприклад, інший жорсткий диск або флеш-накопичувач).

Підхід з архівуванням має декілька недоліків:

- 1.) Кожного разу потрібно заново архівувати все рішення, що може займати багато часу;
- 2.) Управління резервними копіями відбувається вручну;
- 3.) Архів може бути випадково видаленим;
- 4.) У випадку середніх та великих проектів, багато резервних копій будуть займати багато місця в постійній пам'яті.

Для вирішення перелічених проблем, були створені **системи контролю версій (Control Version System)**.

Версія (version) — зафіксований (збережений) стан певного об'єкта. В контексті розробки програмного забезпечення, версією є збережена, працююча функціональність. По відношенню до файлів, версією є стан файла (опис та вміст), після його збереження.

Система контролю версій (Control Version System, CVS) — спеціалізоване програмне забезпечення, призначене для автоматизованого ведення обліку даних в **репозиторії**. Система контролю версій веде облік з використанням бази даних, в якій фіксуються всі зміни в репозиторії.

Репозиторій — місце, в якому зберігаються та підтримуються дані. По відношенню до програмного забезпечення, в якості даних репозиторію можуть виступати:

- 1.) Бібліотеки та файли вихідного коду;
- 2.) Файли ресурсів (зображення, аудіовідеофайли і т.д.);
- 3.) Файли налаштувань.

Репозиторії за місцем розташування можна поділити на:

1. **Локальні (CVS або Revision Control System)** — всі дані зберігаються автономно на одному пристрої, до якого є прямий доступ. Топологію локальної системи контролю версій зображено на Рис.1;

2. **Віддалені (Remote CVS)** — всі дані зберігаються на одному (**централізовані, Centralized CVS**) або декількох пристроях (**розподілені, Distributed CVS**), до якого **немає** прямого доступу. Зазвичай, віддалені системи контролю версій розміщуються на серверах, доступ до яких забезпечується через комп'ютерні мережі (локальні, глобальні). Топологію централізованої та розподіленої систем контролю версій зображено на Рис.2 та Рис.3 відповідно.

При централізованій топології виділяється сервер на якому зберігається єдина база даних системи контролю версій. Якщо вона буде пошкоджена — це призведе до зупинки процесу розробки, до тих пір поки вона не буде відновлена. Серед переваг такої топології — база даних завжди в актуальному стані, тобто всі користувачі працюють завжди з останньою версією рішення.

При розподіленій топології кожен новий учасник процесу розробки отримує повну копію бази даних, що призводить до проблеми синхронізації баз даних між учасниками процесу розробки. В свою чергу, розподілена топологія дозволяє відновити більшу частину бази даних, у випадку пошкодження у одного з учасників розробки.



Рисунок 1 — Схема топології локальної системи контролю версій



Рисунок 2 — Схема топології централізованої системи контролю версій

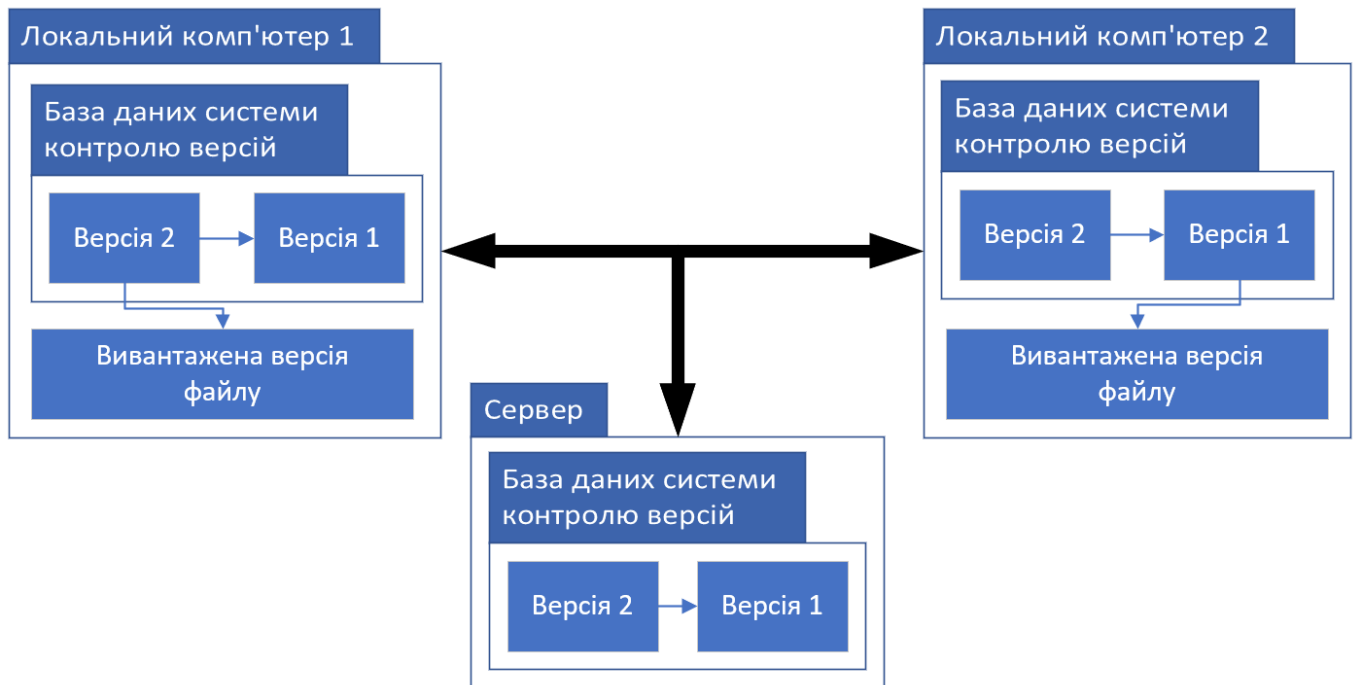


Рисунок 3 — Схема топології розподіленої системи контролю версій

Git, GitHub, GitLab

Популярною системою контролю версій є безкоштовна **розподілена система Git**. Git поставляється в складі деяких дистрибутивів UNIX-подібних операційних систем, для всіх інших операційних систем її потрібно окремо встановлювати. Git має консольний (Рис.4) та графічний інтерфейс.

```
C:\>git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
          [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
          [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
          [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
          [--super-prefix=<path>] [--config-env=<name>=<envvar>]
          <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  restore    Restore working tree files
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  diff       Show changes between commits, commit and working tree, etc
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status

grow, mark and tweak your common history
  branch     List, create, or delete branches
  commit     Record changes to the repository
  merge      Join two or more development histories together
```

Рисунок 4 — Консольний інтерфейс системи контролю версій Git

Для віддаленого зберігання репозиторіїв існує хостинг **GitHub**. Володіє набором інструментів для редагування коду в браузері, також надаються інструменти для забезпечення безпеки (контроль доступу).

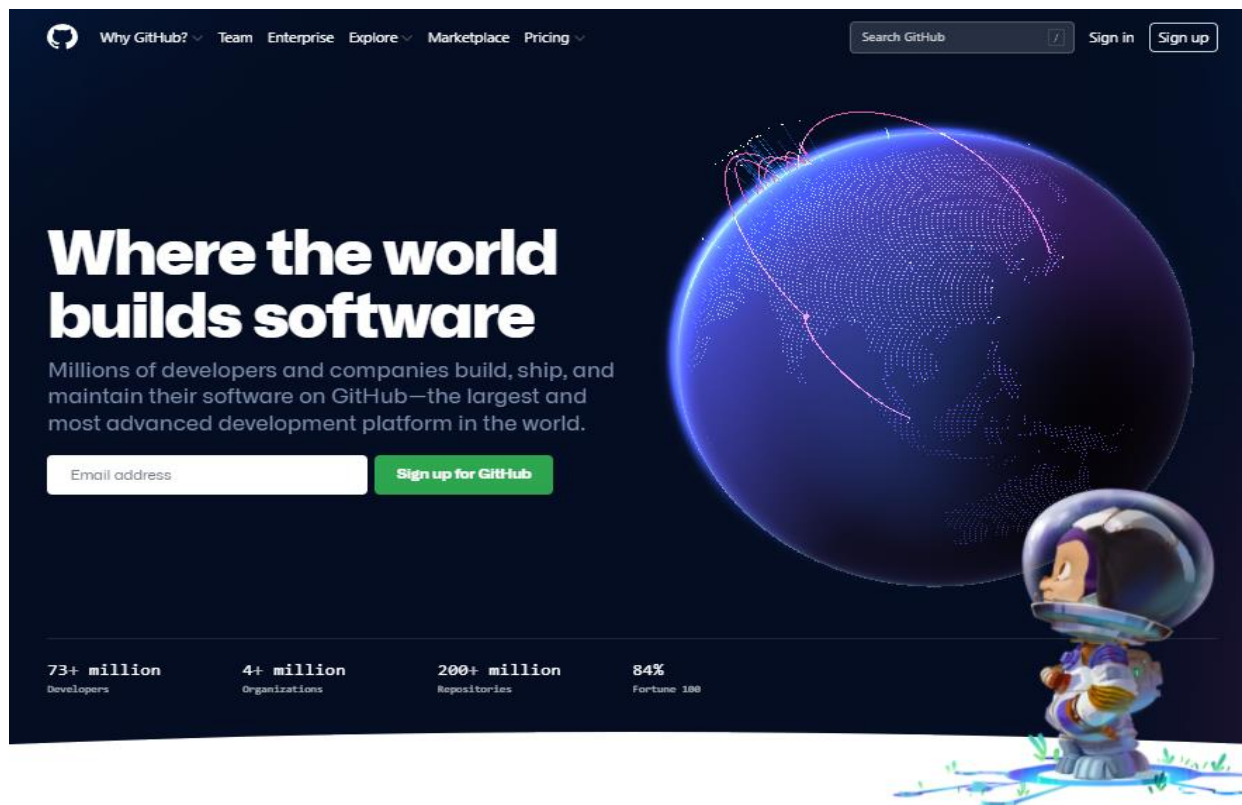


Рисунок 5 — Хостинг проектів GitHub

Коли виникає потреба в автоматизації процесу розробки та тестування, можна налаштувати «конвеєр». **GitLab** — відкрита платформа для налаштування «конвеєра», зображено на Рис.6. «Конвеєр» розробки складається з основних компонентів:

1. **«Постійна збірка» (Continuous Integration)** — автоматизований процес постійної побудови (build) та тестування рішення;
2. **«Постійна поставка» (Continuous Delivery)** — розгортання (deploy) результату «постійної збірки» зацікавленій стороні.

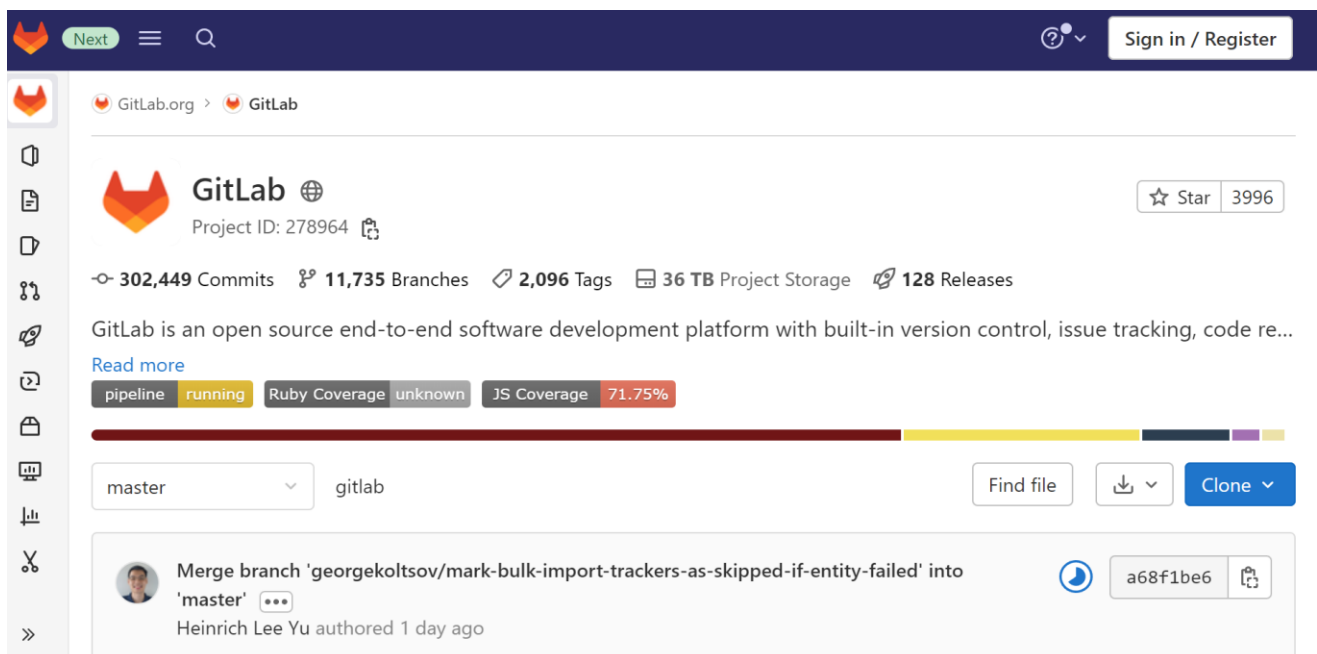


Рисунок 6 — Платформа GitLab

Робота з системою Git

В загальному випадку, робота з системою контролю версій Git складається з етапів (Рис.7):

1. **Створення та ініціалізація репозиторію** — створення та налаштування репозиторію, щоб можна було відслідковувати зміни в рішенні;
2. **Створення «гілки» (branch)** — є різні підходи до організації контролю версій, один з яких передбачає, що під час розробки рішення весь перевірений функціонал потрапляє в основну гілку (main чи master branch), а кожен окремий модуль (проект) розробляється в окремій гілці. Гілку можна розглядати як альтернативний варіант розвитку проекту. Кожна гілка будується на основі попередньої (наприклад, якщо створюється гілка на основі головної, то нова гілка, на момент створення, буде містити в собі всі зафіксовані зміни з головної гілки);
3. **Виявлення файлів для відслідковування** — після додавання файлів в каталог репозиторію, вони будуть відображені як невідслідковувані (untracked), тобто система знайшла нові файли, які ще не внесені в базу даних для відслідковування;
4. **Внесення файлів для відслідковування (індексація)** — процес внесення файлів в базу даних системи контролю версій, для відслідковування, називається індексацією.
5. **Збереження (фіксація) змін** — файли вносяться в базу даних у вигляді знімків (snapshot), по аналогії зі знімками файлової системи, а не зберігаються у вигляді записів змін. Для виявлення змін порівнюються хеш-суми файлів. Для файлів в яких не було ніяких змін, залишається посилання на старий знімок;
6. Якщо роботу в поточній гілці завершено — зміни з поточної гілки вносяться в іншу гілку, тобто відбувається злиття гілок за допомогою команди **git merge**, після чого зміни фіксуються в репозиторії. Якщо репозиторій віддалений, то зміни спочатку надсилаються у віддалений репозиторій за допомогою команди **git push**. Потім необхідно створити запит на оновлення (**pull request**), та тільки після підтвердження (approve) можна виконати злиття гілок.
7. Якщо потрібно продовжити роботу з іншою гілкою, то необхідно її переключити. В іншому випадку, достатньо внести необхідні зміни, та повторити необхідні кроки для збереження результатів роботи.

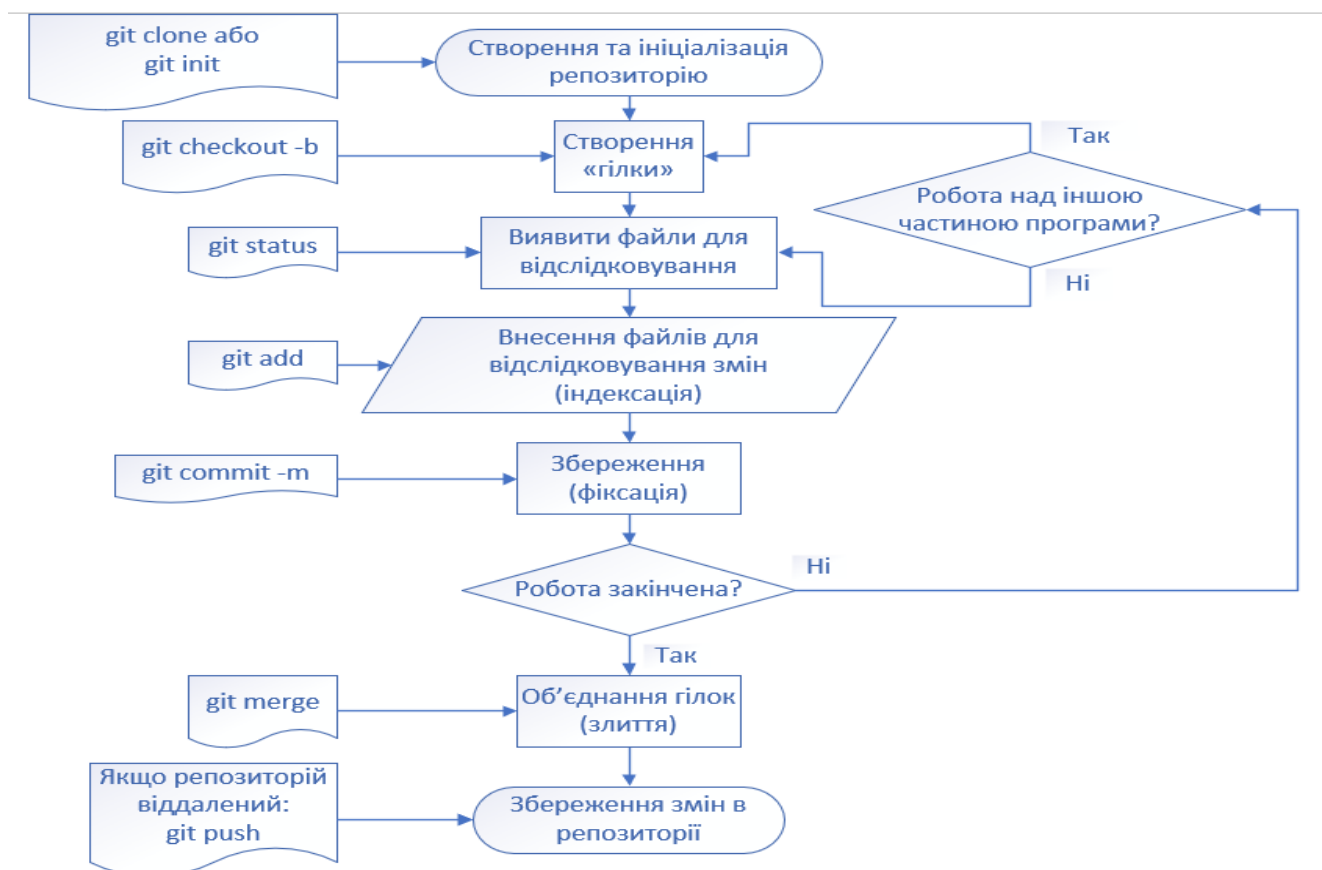


Рисунок 7 — Загальна схема роботи з Git

Для забезпечення колективної роботи над проектом, зручно створити репозиторій на хостингу GitHub. Головною перевагою використання віддаленого репозиторію є **підтримка проекту в актуальному стані для всіх учасників**.

Створення та ініціалізація репозиторію

Для створення віддаленого репозиторію потрібно авторизуватись на сайті github.com та на головній сторінці натиснути кнопку «New» (Рис.8).

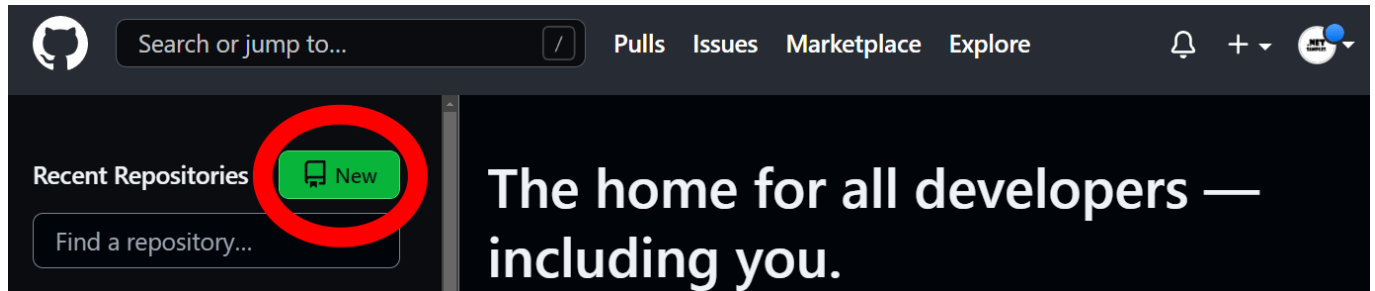


Рисунок 8 — Кнопка створення віддаленого репозиторію

На веб-сторінці, що буде відкрита, потрібно заповнити необхідні поля. Спочатку необхідно задати назву та короткий опис репозиторію (Рис.9).

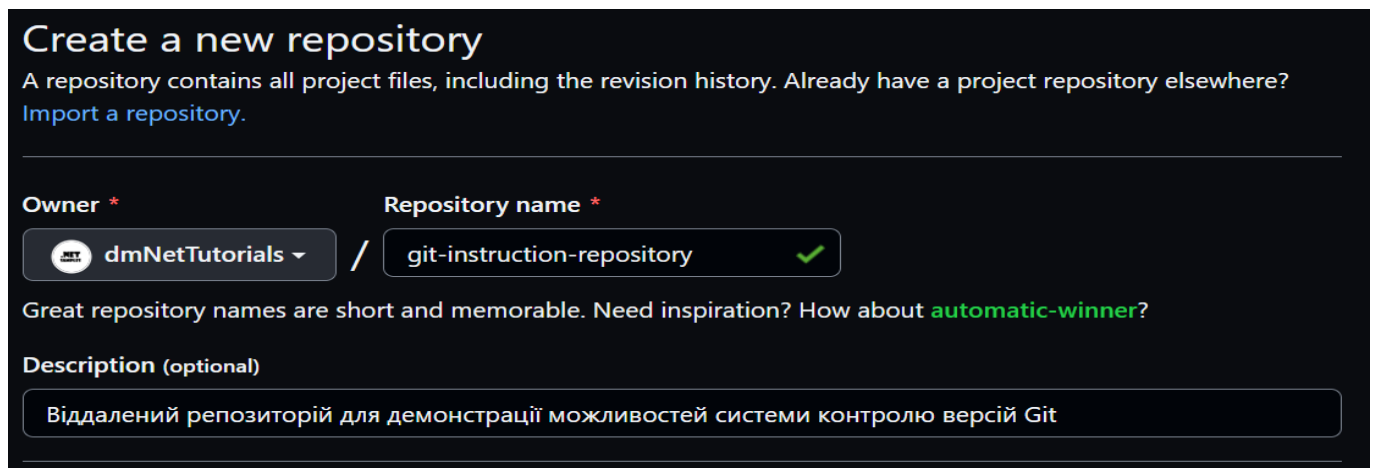


Рисунок 9 — Основні відомості про віддалений репозиторій

Для віддаленого репозиторію необхідно обрати тип доступу (Рис.10). Типи доступу можуть бути:

1. **Відкритими (public)** — будь-хто може переглядати вміст та надсилати запити на зміни віддаленого репозиторію. Пошукові системи (наприклад Google) можуть видавати віддалений репозиторій в результатах пошуку.
2. **Закритими (private)** — вміст та можливість надсилання запитів на зміни надається тільки учасникам, переліченим в налаштуваннях віддаленого репозиторію. Пошукові системи не знаходять закриті репозиторії.

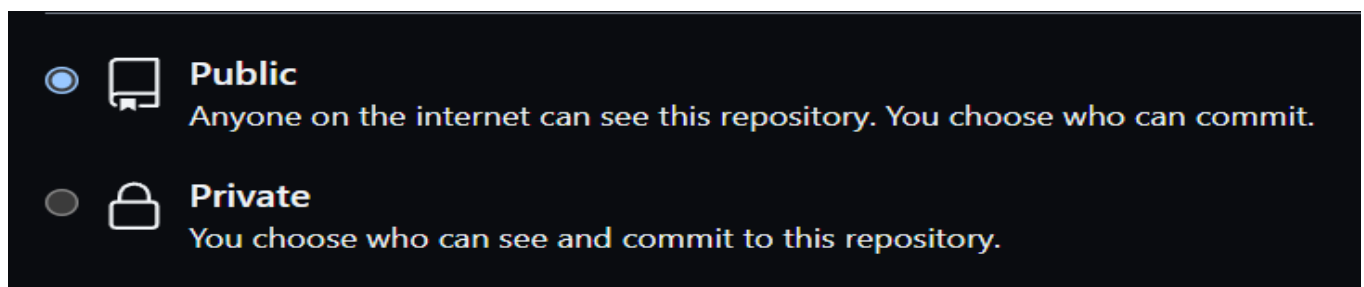


Рисунок 10 — Тип доступу до віддаленого репозиторію

Можна задати додаткові параметри віддаленого репозиторію, які можна в будь-який момент змінити:

1. Додати **README.md** файл — спеціальний файл розмітки, в якому можна вказати повний опис проекту. Використання мови розмітки, дозволяє використовувати широкі можливості форматування тексту та вставку зображень (Рис.11).

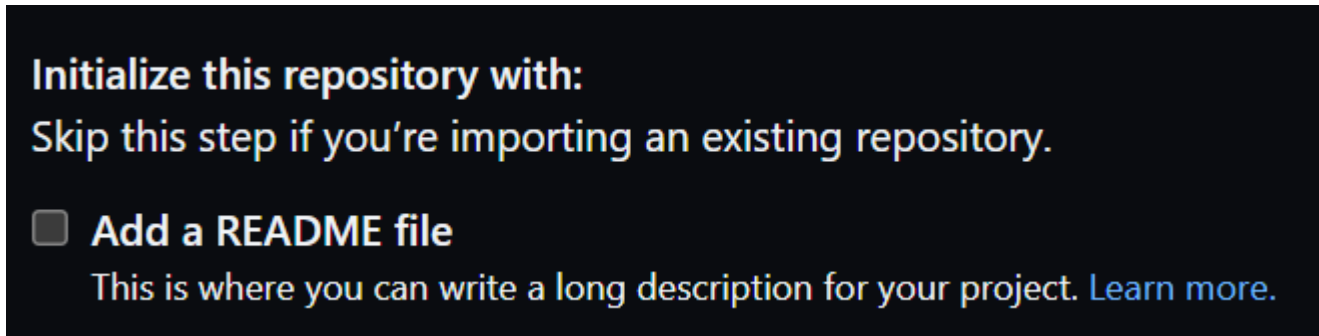


Рисунок 11 — Додавання файлу опису README.md у віддалений репозиторій

2. Додати файл **.gitignore** — в даному файлі зазначається перелік файлів, які не потрібно надсилати у віддалений репозиторій. Щоб не формувати цей файл вручну, в Github є можливість обрати заздалегідь визначені шаблони файлу **.gitignore**, наприклад, щоб не надсилати папки **bin** та **obj** проектів Visual Studio, можна підключити відповідний шаблон (Рис.12).

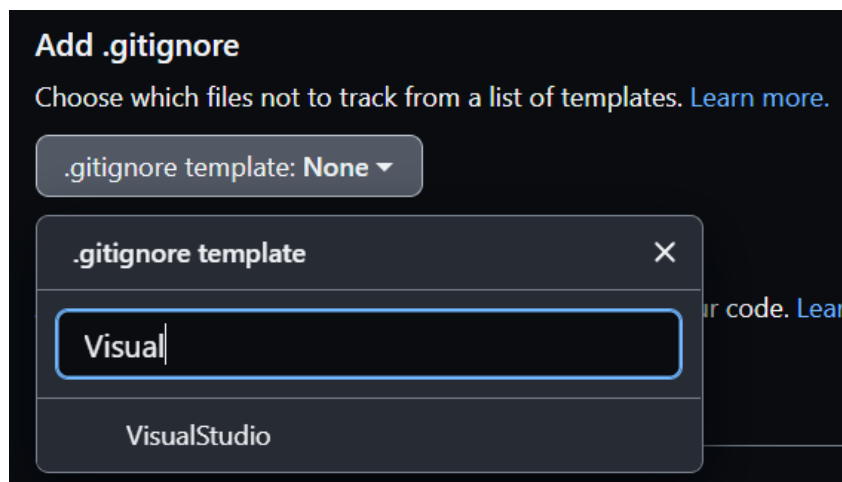


Рисунок 12 — Додавання шаблону файлу .gitignore для проектів Visual Studio

3. Обрати ліцензію — визначити за якими правилами інші користувачі можуть використовувати код розташований у віддаленому репозиторії, тобто передбачають права інтелектуальної власності на код (Рис.13). Кожна окрема ліцензія передбачає різні правила, тому перед обранням ліцензії потрібно детально з ними ознайомитись. Щоб змінити ліцензію — необхідно внести зміни до файлу **LICENSE**.

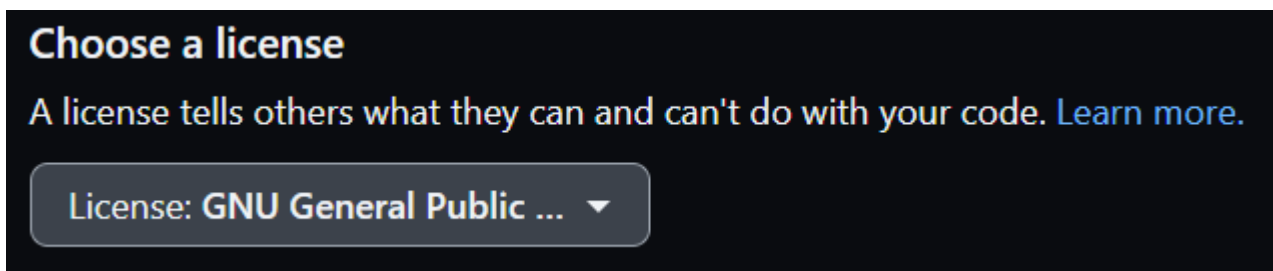


Рисунок 13 — Репозиторій з вказаною ліцензією на код

Після успішного заповнення всіх необхідних полів, буде створено віддалений репозиторій з заданими налаштуваннями (Рис.14). Після чого вже можна працювати з віддаленим репозиторієм.

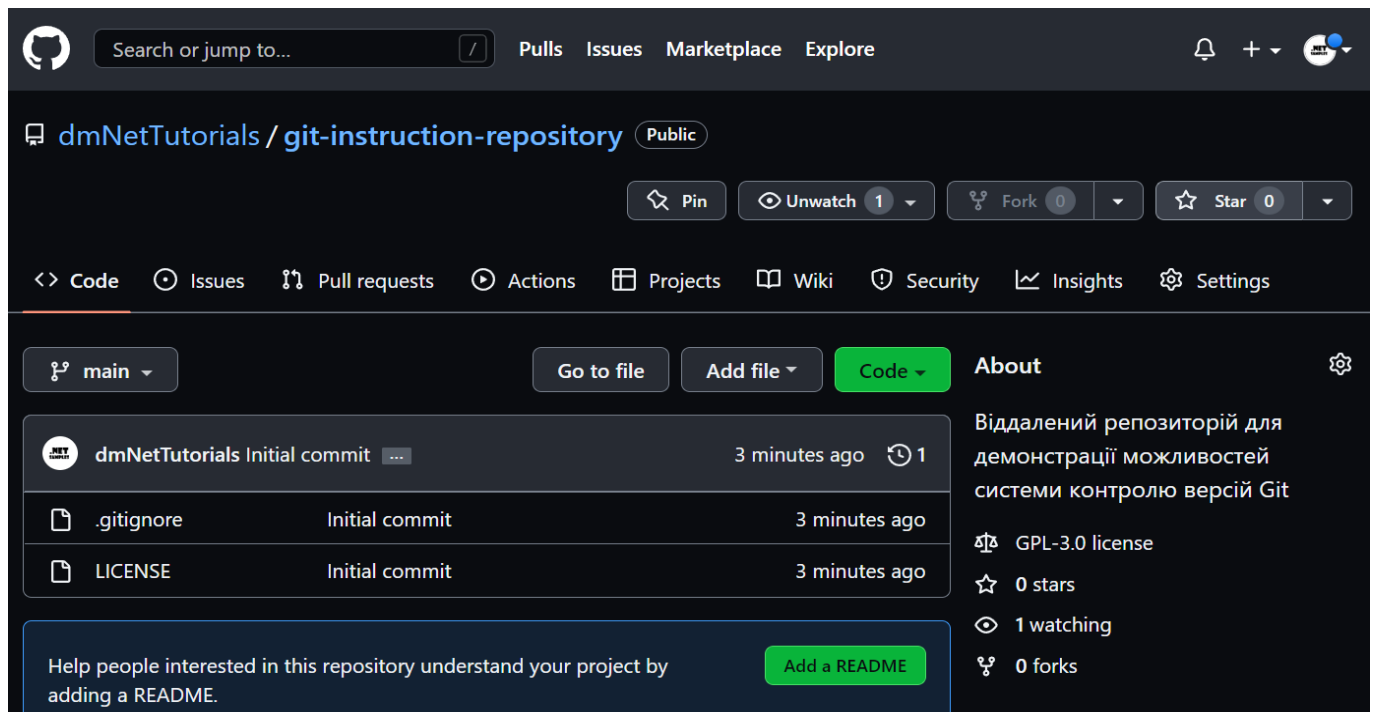


Рисунок 14 — Створений віддалений репозиторій

Якщо віддалений репозиторій було створено без додаткових налаштувань, то буде виведено веб-сторінку з інструкцією, в якій описано послідовність команд необхідних для створення та налаштування віддаленого репозиторію (Рис.15).

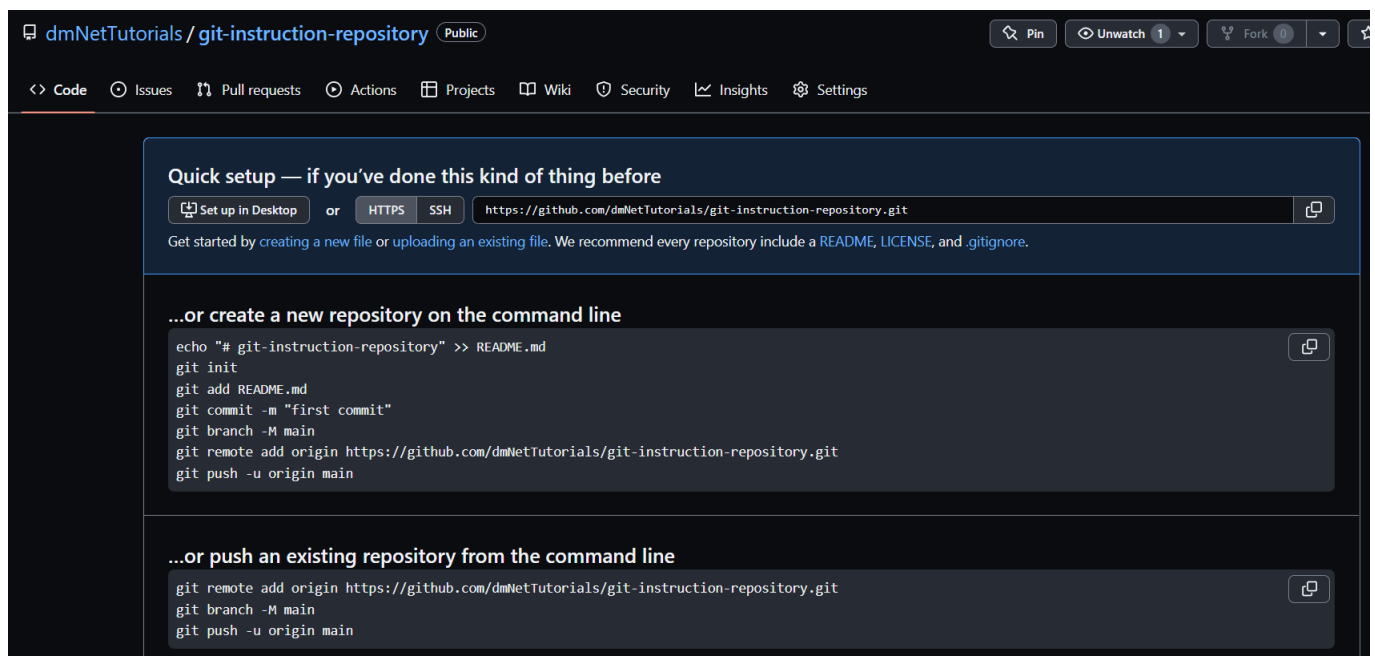


Рисунок 15 — Альтернативний спосіб створення віддаленого репозиторію

Після створення віддаленого репозиторію, необхідно створити локальний репозиторій, в якому буде відбуватись основна робота над проектом.

Натиснувши комбінацію клавіш Win+R, ввести назву програми — «cmd» та натиснути Enter.

Потрібно обрати місце на локальному комп'ютері, де буде зберігатись репозиторій, та за допомогою відповідної команди консолі перейти туди. Після чого виконати команду **git clone**, для завантаження («клонування») віддаленого налаштованого репозиторію. В якості параметра команди **git clone** необхідно вказати адресу репозиторію, яку можна отримати за допомогою відповідного меню на сайті github.com (Рис.16).

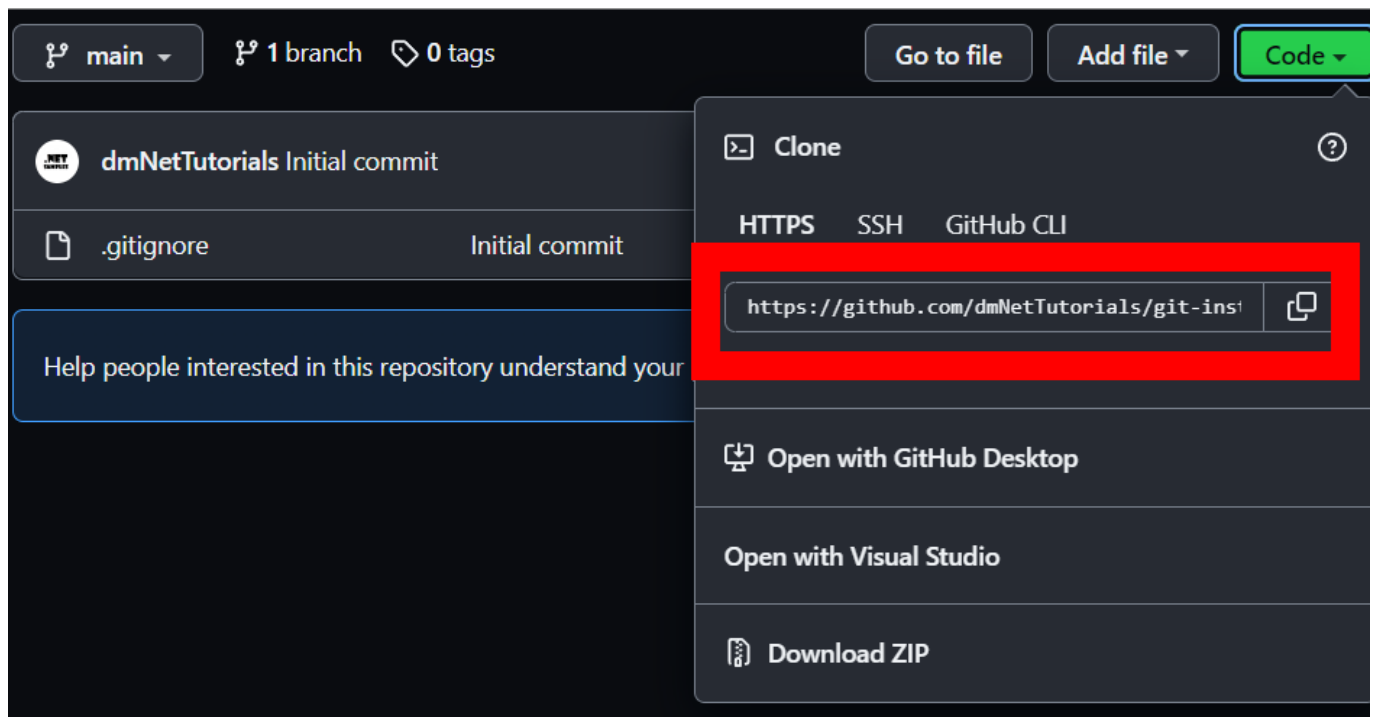


Рисунок 16 — Меню «клонування» віддаленого репозиторію на сайті Github

Порядок команд «клонування» віддаленого репозиторію на локальний комп'ютер зображено на Рис.17.

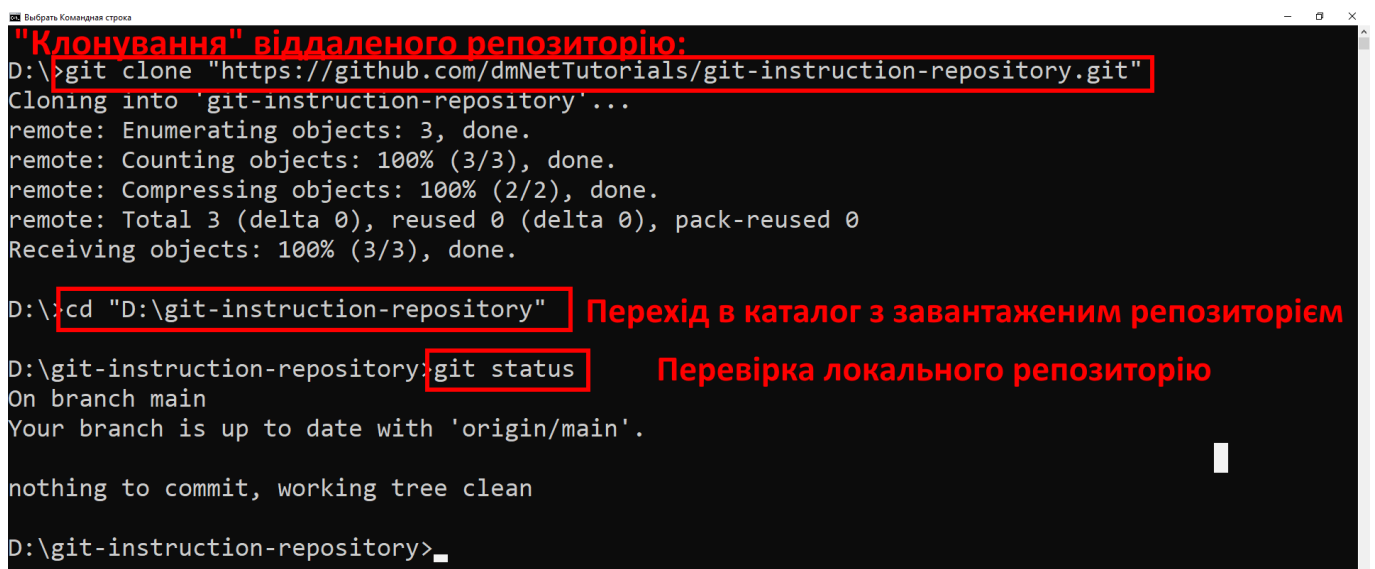


Рисунок 17 — «Клонування» віддаленого репозиторію

Каталог в якому зберігається локальний репозиторій, містить прихований каталог з назвою «.git», в якому зберігається база даних зі змінами в репозиторії, та інші об'єкти необхідні для роботи з системою контролю версій Git.

Щоб перевірити наявність даного каталогу необхідно виконати команди:

Для ОС Windows:

```
?> DIR /A:H /W
```

Для UNIX-подібних ОС:

```
?> ls -a
```

```

D:\git-instruction-repository>DIR /A:H /W
Том в устройстве D имеет метку DM-SSD
Серийный номер тома: 8ED7-7B00

Содержимое папки D:\git-instruction-repository

[.git] ← Прихований каталог з налаштуваннями
          0 файлов                      0 байт
          1 папок   12 439 040 000 байт свободно

D:\git-instruction-repository>

```

Рисунок 18 — Перевірка наявності каталогу з налаштуваннями репозиторію в ОС Windows

Створення «гілок»

Процес створення інформаційної системи можна розглядати як сукупність підпроцесів, які характеризуються власними *потоками (flow) або робочий процес*.

Потік (flow) або робочий процес — в контексті процесу розробки програмного забезпечення, означає умови та послідовність дій, що потенційно повинні привести до очікуваного результату. Під час процесу розробки потік може змінюватись відповідно до нових умов. Потік є частиною *життєвого циклу (lifecycle) процесу*.

Під час розробки інформаційної системи є мінімум один, головний потік (**main flow**). Результатом виконання головного потоку є **готова версія (release version)** інформаційної системи.

В системі контролю версій Git для представлення потоку використовуються **«гілки» (branch)**. В них фіксуються певні зміни для конкретного потоку, які по завершенню додаються (**«зливаються», merge**) в головну гілку, яка представляє головний потік розробки. Схематичне зображення гілок наведено на Рис.

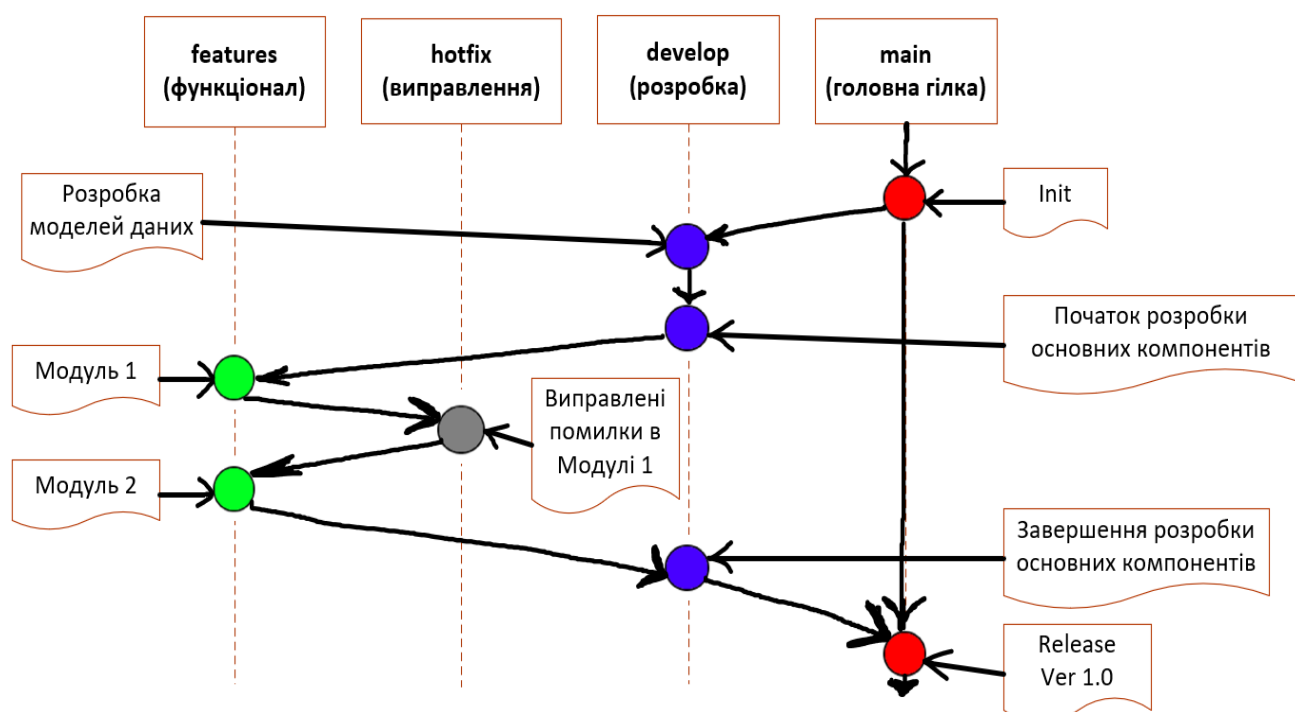


Рисунок 19 — Схематичне зображення гілок в системі контролю версій Git

Під час розробки виникають ситуації, коли є декілька альтернативних підходи для вирішення задачі. Для кожного з таких підходів можна створити окрему гілку, а потім порівняти підходи та визначити найкращий підхід.

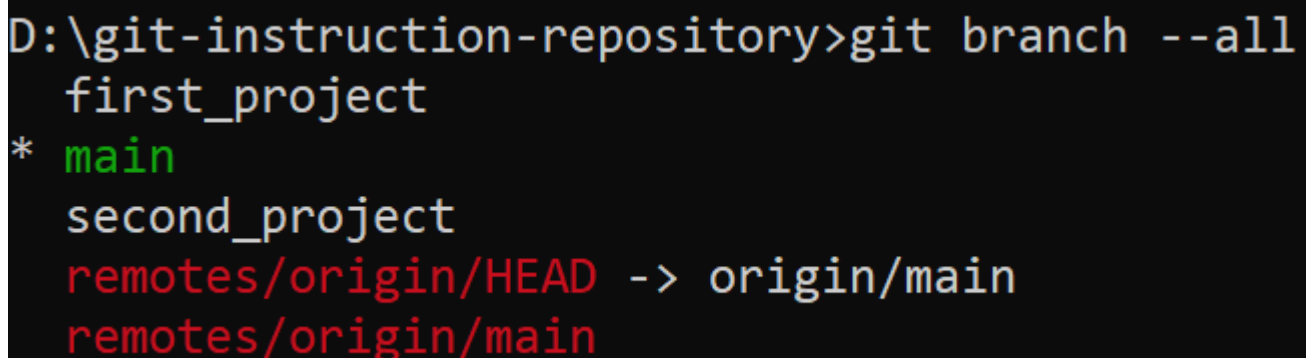
Існує декілька стратегій ведення гілок в системі контролю версій Git:

1. Розробка на основі нововведень (FBD, Feature Based Development) — підхід, при якому розробляється ціла функціональна частина. Даний підхід зустрічається в проектах з відкритим вихідним кодом (open-source), за рахунок того, що у розробників може бути різний рівень підготовки та кваліфікація. Налаштування CI/CD-«конвеєра» більш складне, за рахунок того, що функціонал може бути різним (як за об'ємом коду, так і за структурою). При такому підході нові версії програми випускаються нечасто.

2. Магістральна розробка (TBD, Trunk Based Development) — підхід, при якому вносяться невеликі зміни до коду, але дуже часто. За рахунок частого внесення змін, можна організувати оперативну поставку програми замовнику. Налаштування CI/CD-«конвеєра» відносно просте, за рахунок того, що зміни можна швидко перевірити та внести в основну гілку розробки (тому цей підхід називається магістральним). Нові версії програм випускаються часто, але вимагають від команди розробників високої кваліфікації (як в написанні коду, так і тестів до коду).

Дізнатись перелік всіх гілок в репозиторії, можна за допомогою команди (гілка в якій зараз ведеться розробка буде помічена знаком «*»). Результат виконання команди зображено на Рис.20:

```
?> git branch --all
```



```
D:\git-instruction-repository>git branch --all
first_project
* main
second_project
remotes/origin/HEAD -> origin/main
remotes/origin/main
```

Рисунок 20 — Перелік всіх гілок в поточному репозиторії

Для створення нової гілки використовується команда:

```
?> git branch "назва гілки"
```

Кожна нова гілка будується на основі поточної. Наприклад, якщо створити гілку на основі main, то нова гілка буде включати в себе зміни, які були зроблені в гілці main на момент створення нової гілки.

Для переходу на іншу гілку використовується команда:

```
?> git checkout "назва гілки"
```

Існує можливість створити та переключитись на створену гілку за допомогою однієї команди:

```
?> git checkout -b "назва гілки"
```

Назви гілок формуються на англійській мові, та повинні описувати призначення гілки. Якщо назва гілки складається з декількох слів, їх можна розділити символом нижнього підкреслення «_». Наприклад, гілку можна назвати `development_mobile`, де з назви зрозуміло, що гілка призначена для розробки мобільної версії програми.

Типові імена для гілок:

1. `main` (`master` в старих версіях Git) — головна гілка. В дану гілку потрапляє весь протестований та затверджений код. На основі даної гілки випускаються версії інформаційної системи (релізи, `release`);
2. `develop` — гілка розробки, в якій відбувається основний процес розробки. Може існувати декілька таких гілок, окрема для кожної платформи, наприклад, для мобільної версії — `develop_mobile`, для веб-версії — `develop_web_app`;
3. `feature` — конкретна функціональність програми. Наприклад, модуль взаємодії з мережею. Таких гілок може бути дуже багато;
4. `hotfix` — гілка для виправлень проблем в коді. Для кожної проблеми створюється окрема гілка. Наприклад, гілка може називатись `hotfix_123_web_app`, «123» в назві означає номер помилки, а «`web_app`» — помилка в веб-застосуванні.

Виявлення файлів для відслідковування

Після того, як створено репозиторій та відповідні гілки, можна починати розробку інформаційної системи. Для того, щоб система контролю версій Git відслідковувала зміни в рішенні, необхідно:

1. Розмістити рішення з усіма його проектами **на одному рівні** (робочий каталог) з прихованим каталогом `“.git”`, як зображено на Рис.21.

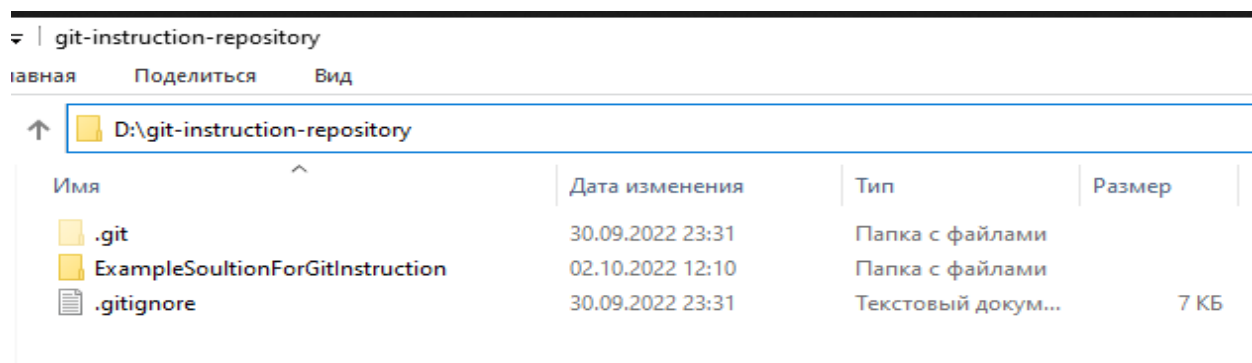


Рисунок 21 — Розташування рішення для відслідковування

2. За допомогою команди, перевірити чи знаходить система контролю версій нові файли та рішення (Рис.22). Нові файли та рішення будуть відображені червоним кольором (невідстежуваних):

`?> git status`

```
D:\git-instruction-repository>git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ExampleSoultionForGitInstruction/

nothing added to commit but untracked files present (use "git add" to track)
```

Рисунок 22 — Результат виконання команди `git status`

За допомогою команди `git status`, можна перевіряти зміни в репозиторії. Інформація про проіндексовані файли зберігається в файлі `.git\index`.

Внесення файлів для відстеження змін (індексація)

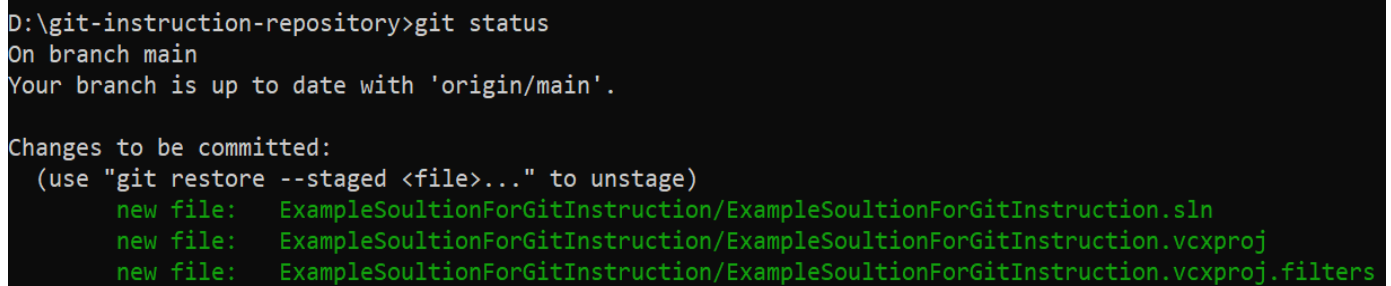
Індексація (stage) — процес внесення невідстежуваних файлів в локальну базу даних системи контролю версій, для відслідковування змін. Індексація файлів відбувається за допомогою команди:

```
?> git add "назва файлу або каталогу"
```

Якщо файлів для індексації дуже багато, можна виконати команду:

```
?> git add --all
```

Після виконання індексації, можна перевірити, які файли були проіндексовані (виділені зеленим кольором), за допомогою команди `git status` (Рис.23).



```
D:\git-instruction-repository>git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   ExampleSoultionForGitInstruction/ExampleSoultionForGitInstruction.sln
    new file:   ExampleSoultionForGitInstruction/ExampleSoultionForGitInstruction.vcxproj
    new file:   ExampleSoultionForGitInstruction/ExampleSoultionForGitInstruction.vcxproj.filters
```

Рисунок 23 — Перелік проіндексованих файлів

Якщо потрібно відмінити індексацію деякого файла, потрібно виконати команду :

```
?> git restore --staged "Назва файлу"
```

Коли потрібно відмінити індексацію всіх файлів, потрібно виконати команду:

```
?> git restore --staged *
```

Збереження індексованих файлів (commit)

Збереженням індексованих файлів є їх **знімок (snapshot)**, тобто фіксація стану файлів на момент збереження. В системі контролю версій зберігається посилання на такі знімки. Для файлів, в які не були внесені зміни, залишається старе посилання, а не створюється нове[1]. Перелік знімків в локальному репозиторії, зберігаються в прихованому каталозі — `".git/objects"`.

Будь-яка гілка складається з послідовності фіксацій. Послідовність фіксацій формує **історію фіксацій**. Схематичне зображення історії фіксацій наведено на Рис.24.

Кожна фіксація однозначно ідентифікується за допомогою хеш-суми. Також за допомогою хеш-сум, система контролю версій Git визначає, чи був змінений файл. Якщо файл був змінений — буде підрахована інша хеш-сума файла.

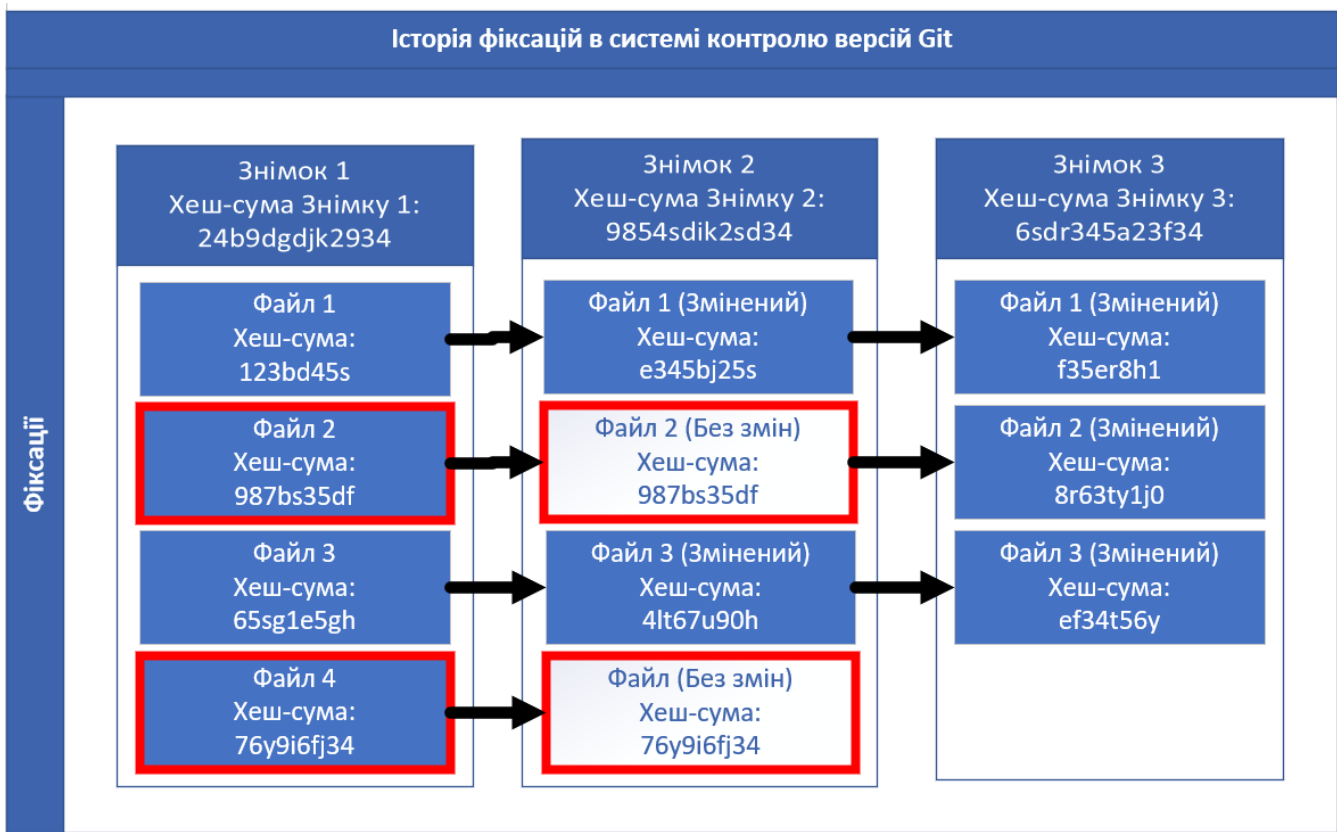


Рисунок 24 — Схематичне зображення історії фіксацій в Git

Фіксація відбувається за допомогою команди:

?> git commit -m "Назва фіксації"

Назви для фіксації потрібні для того, щоб можна було швидко зорієнтуватись що було зафіксовано. Такий підхід зручний для людей, оскільки хеш-сума не дає чіткого розуміння, що було зафіксовано. Тому важливо обирати імена таким чином, щоб можна було зрозуміти, що було зафіксовано.

У випадку, коли на одному комп'ютері використовуються два облікових записи github, система контролю версій Git повідомить про помилку (Рис.25).

```
D:\git-instruction-repository>git commit -m "First version of project"
Author identity unknown

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'DM@DESKTOP-L55FP41.(none)')
```

Рисунок 25 — Одночасне використання декількох облікових записів github

Для усунення проблеми з конфліктами облікових записів, можна додатково налаштувати локальний репозиторій за допомогою команди **git config**. Команда `git config` дозволяє налаштовувати систему контролю версій Git, як на рівні системи (параметр **--global**), так і на рівні окремого репозиторію. Після налаштувань потрібно знову виконати спробу фіксації змін (Рис.26).

Для уточнення, який обліковий запис необхідно використовувати потрібно:

?> git config user.email "вказати необхідну ел.пошту, яку потрібно використовувати"

?> git config user.name "Вказати назву облікового запису в GitHub"

```
D:\git-instruction-repository>git config user.email "d.o.sulyma@npu.edu.ua"

D:\git-instruction-repository>git config user.name "dmNetTutorials"

D:\git-instruction-repository>git commit -m "First version of project"
[develop 8bfacd5] First version of project
 3 files changed, 186 insertions(+)
 create mode 100644 ExampleSoultionForGitInstruction/ExampleSoultionForGitInstruction.sln
 create mode 100644 ExampleSoultionForGitInstruction/ExampleSoultionForGitInstruction.vcxproj
 create mode 100644 ExampleSoultionForGitInstruction/ExampleSoultionForGitInstruction.vcxproj.filters

D:\git-instruction-repository>git status
On branch develop
nothing to commit, working tree clean
```

Рисунок 26 — Налаштування локального репозиторію та фіксація змін

Якщо виникла необхідність відмінити останню фіксацію, то потрібно виконати команду:

?> git commit --amend

Щоб переглянути історію фіксацій потрібно виконати команду:

?> git log

```
D:\git-instruction-repository>git log
commit 8bfacd50c715f42faa31c0ff8d1d0e3842af6cad (HEAD -> develop)
Author: dmNetTutorials <d.o.sulyma@npu.edu.ua>
Date:   Sun Oct 2 14:30:02 2022 +0300

    First version of project

commit 09b315ec9681745628d7cc7c975008208b2f3083 (origin/main, origin/HEAD, second_project, main)
Author: dmNetTutorials <95587345+dmNetTutorials@users.noreply.github.com>
Date:   Fri Sep 30 23:14:18 2022 +0300

    Initial commit
```

Рисунок 27 — Історія фіксацій

Якщо потрібно перейти на конкретну фіксацію, необхідно знати її хеш-суму, та виконати команду:

?> git checkout "хеш-сума фіксації"

Надсилання змін в віддалений репозиторій

Щоб надіслати зміни поточної гілки у віддалений репозиторій, необхідно виконати команду:

?> git push

В результаті чого буде виведено повідомлення (Рис.28), в якому вказана наступна команда:

```
D:\git-instruction-repository>git push
fatal: The current branch develop has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin develop

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

Рисунок 28 — Спроба надіслати зміни у віддалений репозиторій

Після виконання попередньої команди, фіксації будуть надіслані у віддалений репозиторій у вигляді **запиту на зміну (pull request)**. Для цього необхідно перейти за посиланням, яке повідомить система контролю версій Git (Рис.29).

```
D:\git-instruction-repository>git push --set-upstream origin develop
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 2.37 KiB | 2.37 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/dmNetTutorials/git-instruction-repository/pull/new/develop
remote:
To https://github.com/dmNetTutorials/git-instruction-repository.git
 * [new branch]      develop -> develop
branch 'develop' set up to track 'origin/develop'.
```

Рисунок 29 — Посилання сформоване системою контролю версій Git

Перейшовши за посиланням, буде відкрита форма запиту на зміни (Рис.30).

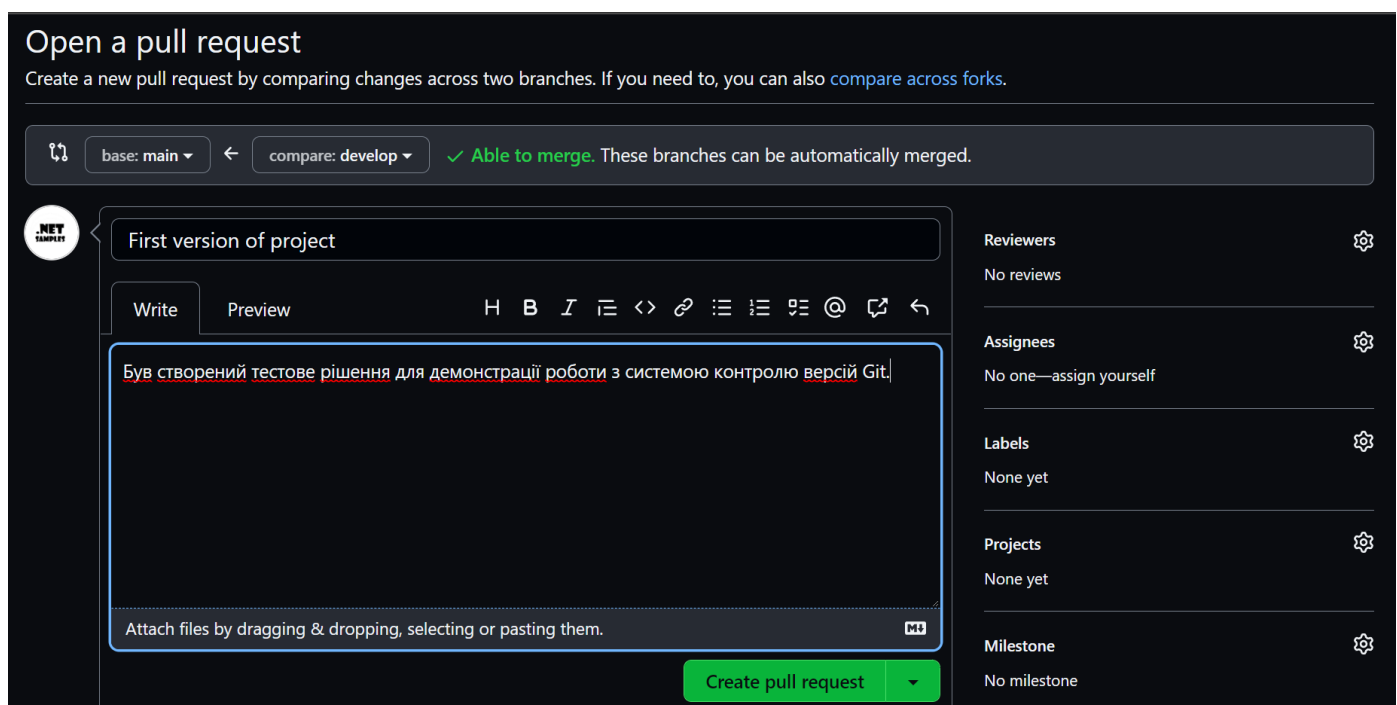


Рисунок 30 — Форма відкриття запиту на зміни

Натиснувши зелену кнопку з надписом «Create Pull Request», буде створено запит, який можна переглянути у вкладці «Pull Requests» на сторінці репозиторію (Рис.31). Всі користувачі, у яких є доступ до репозиторію, можуть надсилати власні запити на зміни.

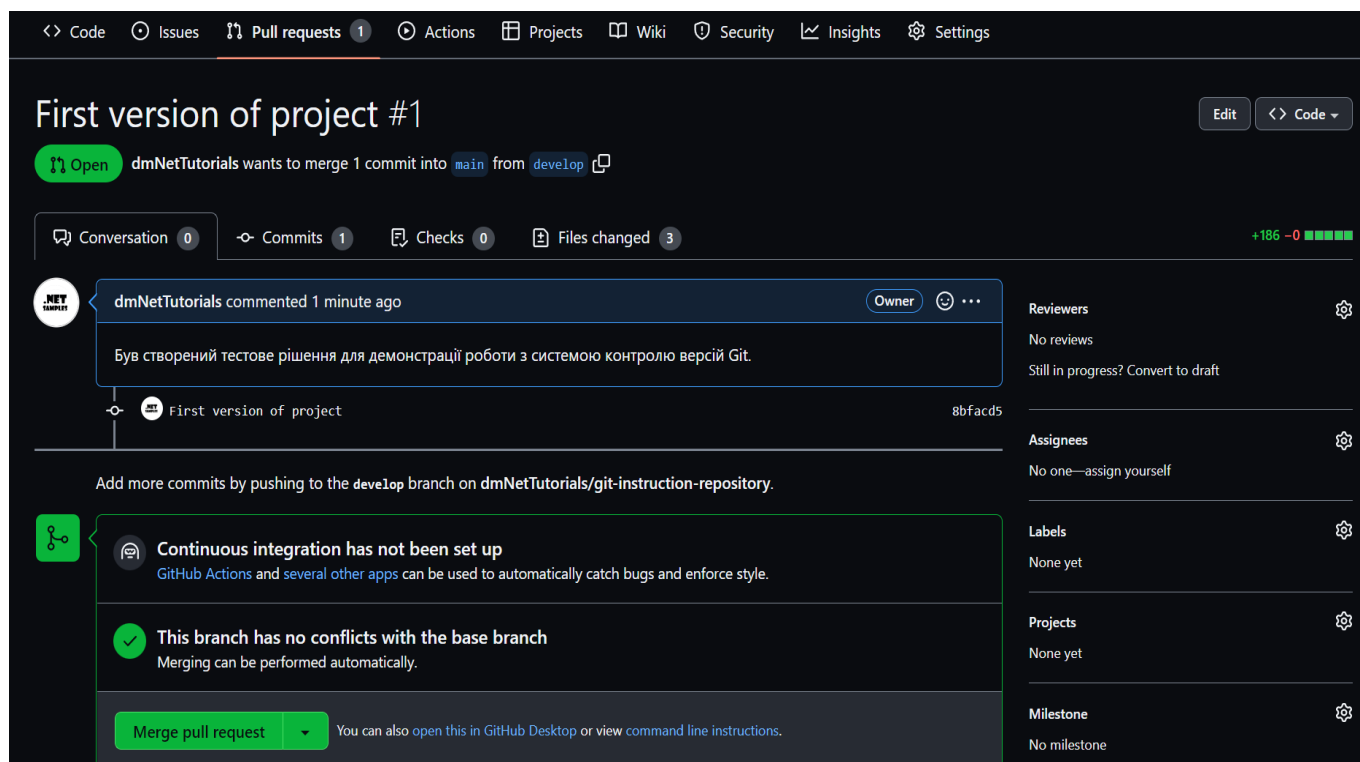


Рисунок 31 — Відображення відомостей про запит на зміни у віддаленому репозиторії

Якщо зміни надіслані за допомогою запиту на зміни не суперечать (конфліктують) з існуючим кодом, то можна виконати **злиття (merge)**, тобто перенести надісланий код в головну гілку. Для цього потрібно натиснути кнопку з надписом «Merge pull request», та підтвердити дію (Рис.32).

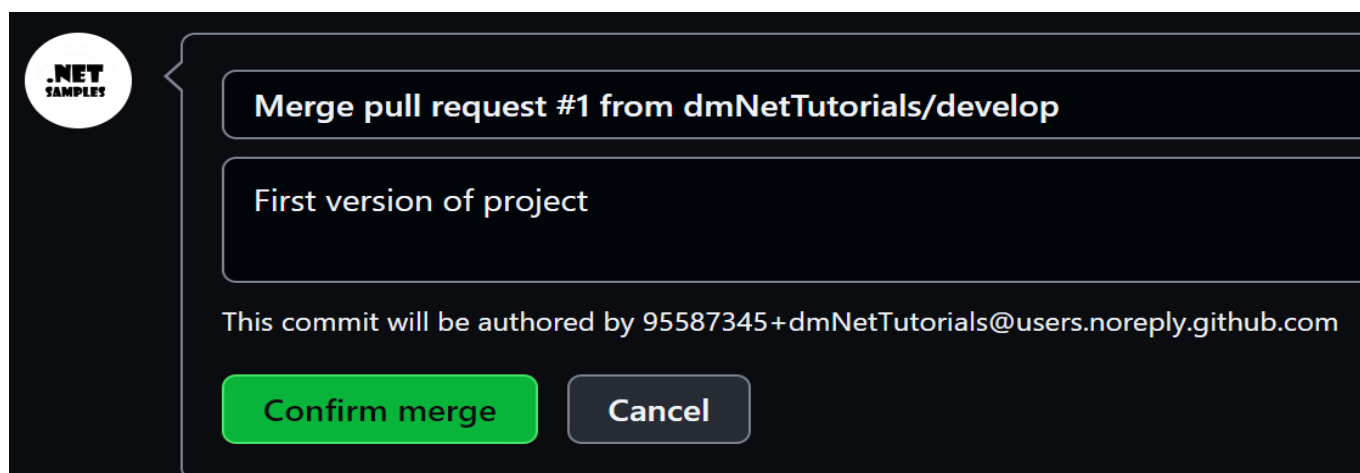


Рисунок 32 — Підтвердження «злиття» коду з головною гілкою.

Після успішного поєднання коду, буде запропоновано видалити гілку (Рис.33), якщо в майбутньому передбачається внесення змін до цієї гілки, то її не потрібно видалити.

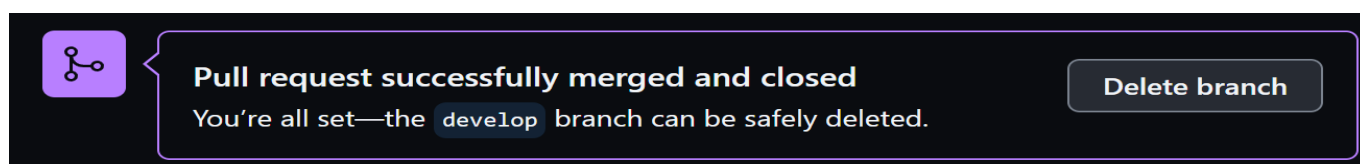


Рисунок 33 — Успішне «злиття» коду в головну гілку

Коли відбулось «злиття» коду з головною гілкою, то всі учасники з доступом до репозиторію можуть побачити (Рис.34) та завантажити нові зміни. Перевірити наявність змін в віддаленому репозиторії можна за допомогою команди:

?> git fetch

Для завантаження змін з віддаленого репозиторію, потрібно виконати команду:

?> git pull

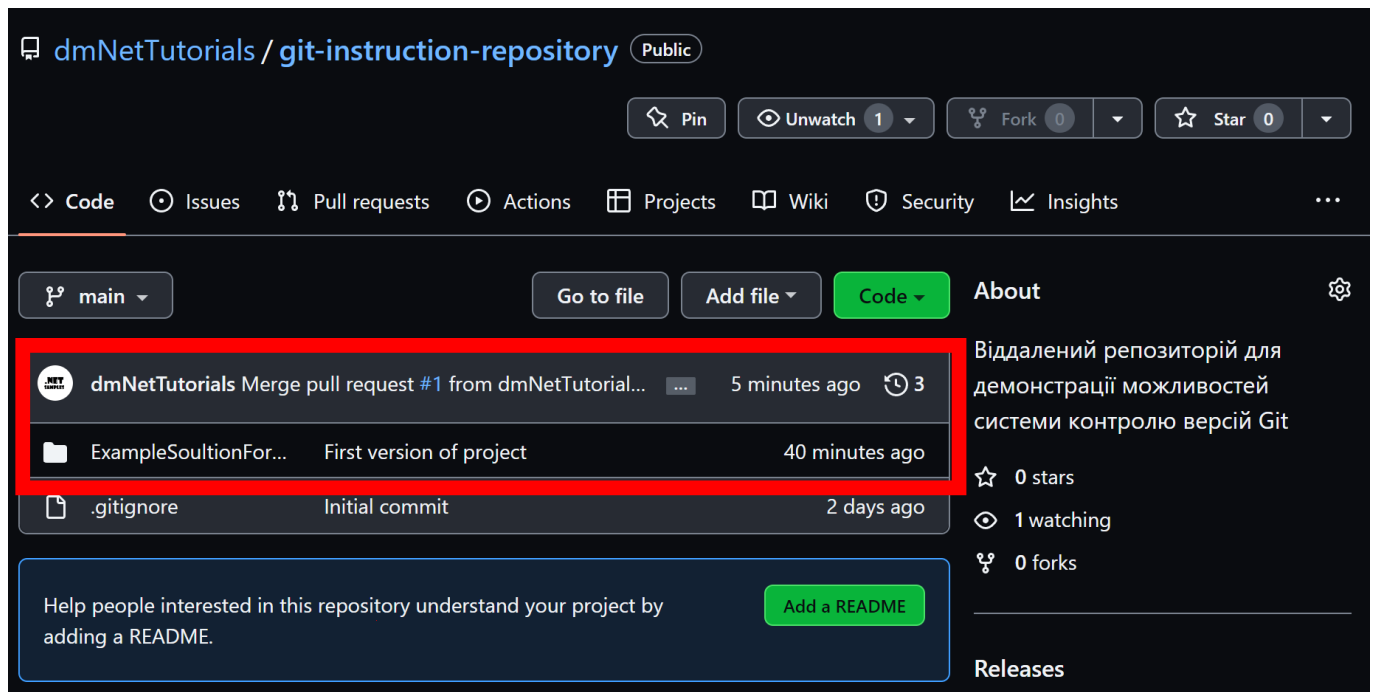


Рисунок 34 — Відображення змін у віддаленому репозиторії

Під час роботи з системою контролю версій Git, ще є важливі аспекти, які потрібно розглядати більш детально щоб зрозуміти, як вони працюють:

1. **Робота з історією фіксацій** (команди git rebase, git merge, git log тощо);
2. **Вирішення конфліктів** (встановлення інструментів для вирішення конфліктів та дослідження їх основних команд);

Детальний розгляд аспектів для роботи з історією та вирішенням конфліктів буде розглянуто в наступних роботах.

Загальні завдання

1. Ознайомитися з теоретичною частиною.
2. Створити обліковий запис в GitHub за посиланням вказаному в джерелі [8];
3. Встановити систему контролю версій Git (бажано Git CMD або Git Bash);
4. Створити відкритий (public) репозиторій на хостингу GitHub;
5. Клонувати на локальний комп'ютер (використовуючи команду git clone);
6. Створити в локальному репозиторії гілку (команда git checkout -b);
7. Додати в репозиторій файли з кодом або створити тестове рішення (можна використовувати будь-яку IDE);
8. Індексувати файли або рішення в локальному репозиторії (команда git add);
9. Зафіксувати зміни (команда git commit);
10. Надіслати запит на зміни у віддалений репозиторій (команда git push);
11. Поєднати код з запиту на зміни з головною гілкою (merge pull request);
12. Надіслати викладачу посилання на створений репозиторій в GitHub.

Контрольні запитання

1. Що таке системи контролю версій та їх призначення
2. Призначення хостингів проектів
3. Що таке версія?
4. Що таке репозиторій та як поділяються за місцем розташування?
5. Різниця між Git, GitHub, GitLab?
6. Яким чином можна створити репозиторій?
7. Для чого призначені гілки?
8. Стратегії створення та ведення гілок
9. Основні команди управління гілками
10. Виявлення файлів для індексації
11. Фіксація змін
12. Запит на зміни у віддалений репозиторій

Інформаційні джерела

1. Чакон С., Штрауб Б. Git для професійного програміста. — СПб.: Питер, 2016. — 496 с.: ил.
2. GitHub. — 2021. — Mode of access: <https://github.com/> Date of Access: 15.10.2021.
3. Git Documentation. — 2021. — Mode of access <https://git-scm.com/doc> Date of Access: 15.10.2021
4. GitHub Documentation. — 2021. — Mode of access <https://docs.github.com/en> Date of Access: 15.10.2021
5. Изучаем ветвление в Git. — 2021. — Mode of access: https://learngitbranching.js.org/?locale=ru_RU Date of Access: 15.10.2021.
6. Как использовать Git правильно: обучающие материалы, новости и советы. — 2021. — Mode of access: <https://www.atlassian.com/ru/git> Date of Access: 15.10.2021.