***Кафедра комп’ютерної та програмної інженерії***

**ЛАБОРАТОРНА РОБОТА № 2**

**з курсу**

**«Архітектура програмної платформи .NET»**

**Студент: Бондарєва Є.Ю.**

**Група: № 42 ІПЗ**

**Факультет: Математики, інформатики**

**та фізики**

**Перевірив: Січкаренко В.О.**

Тема: Основні операції в системі контролю версій Git.

Мета: Ознайомитись та підготувати інструменти під власні потреби, які використовуються для розробки програмного забезпечення (ПЗ).

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 Git

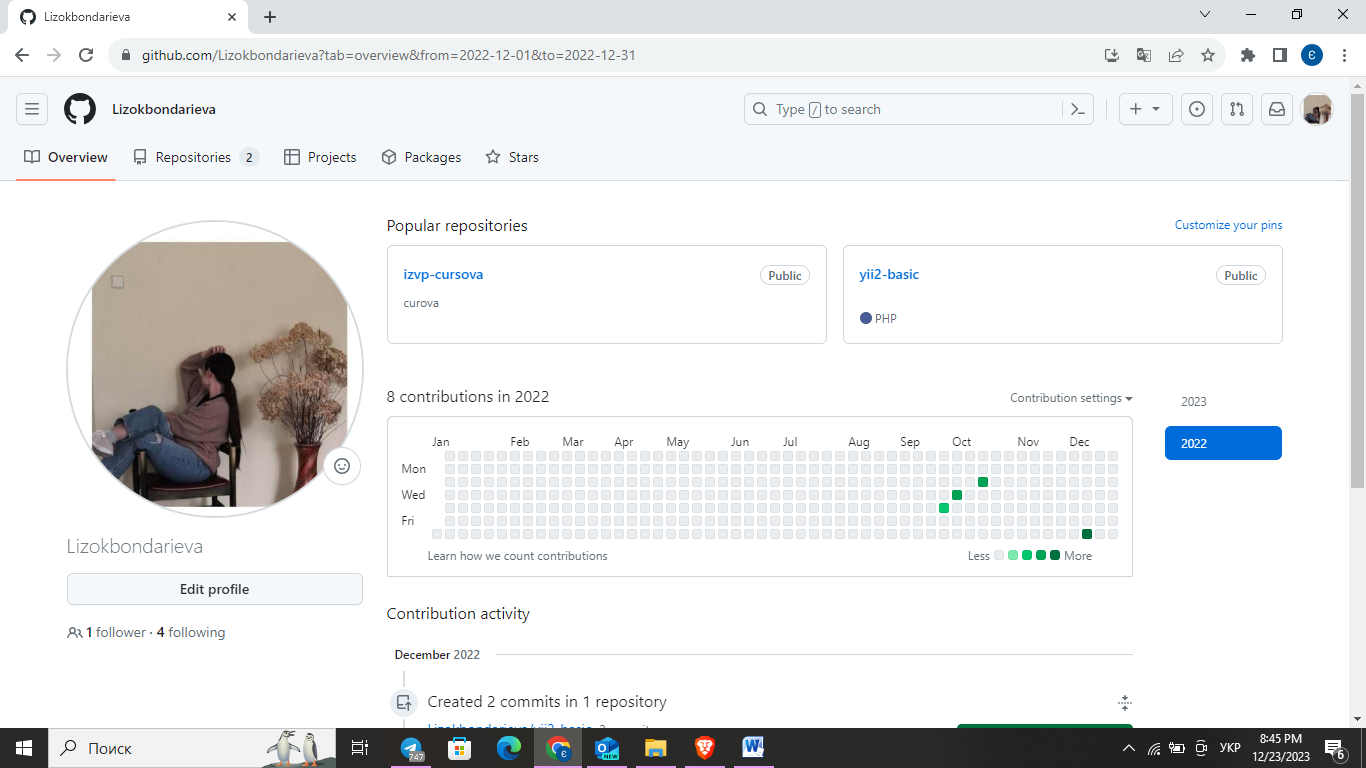


Рисунок 1 - Сторінка облікового запису в GitHub

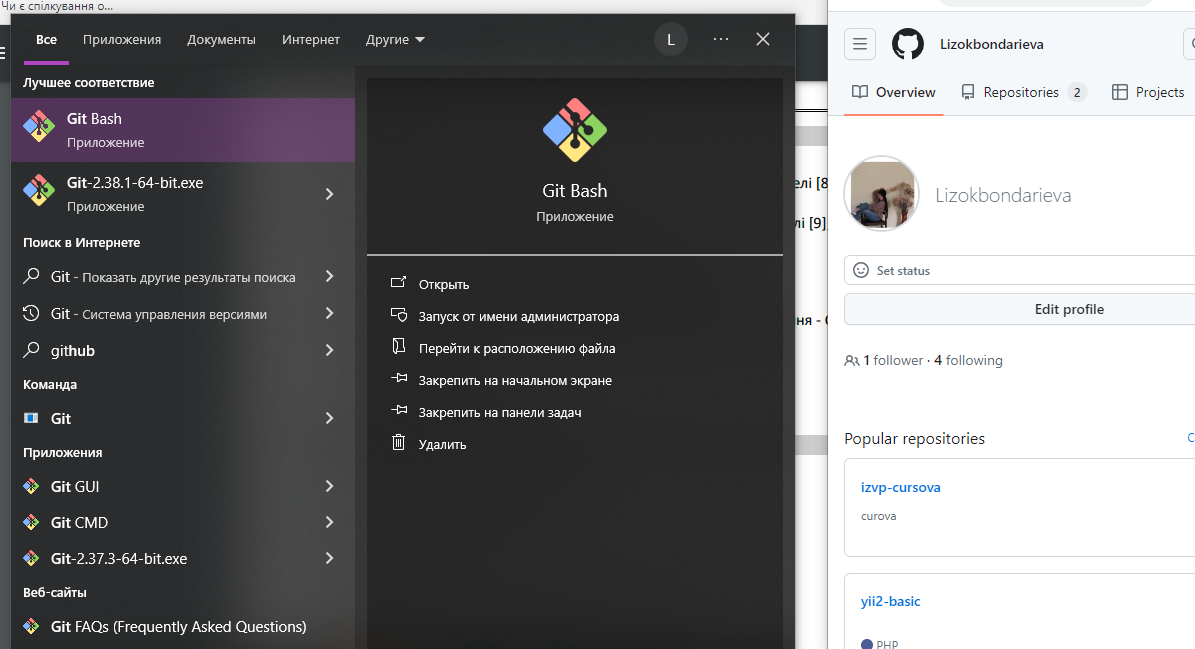


Рисунок 2 - Встановлена система контролю версій Git

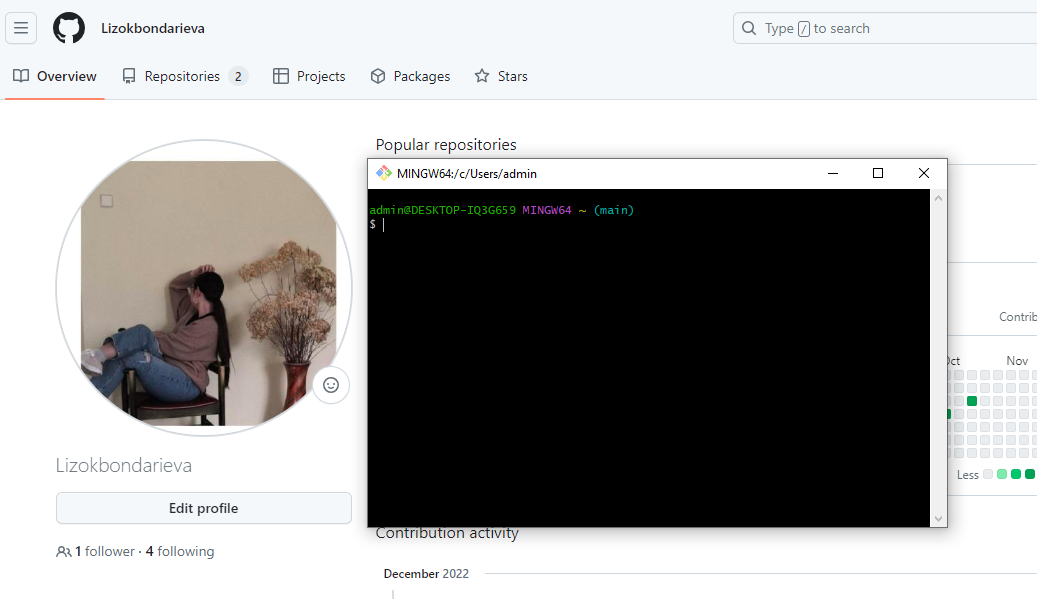


Рисунок 3 – Вікно встановленої системи

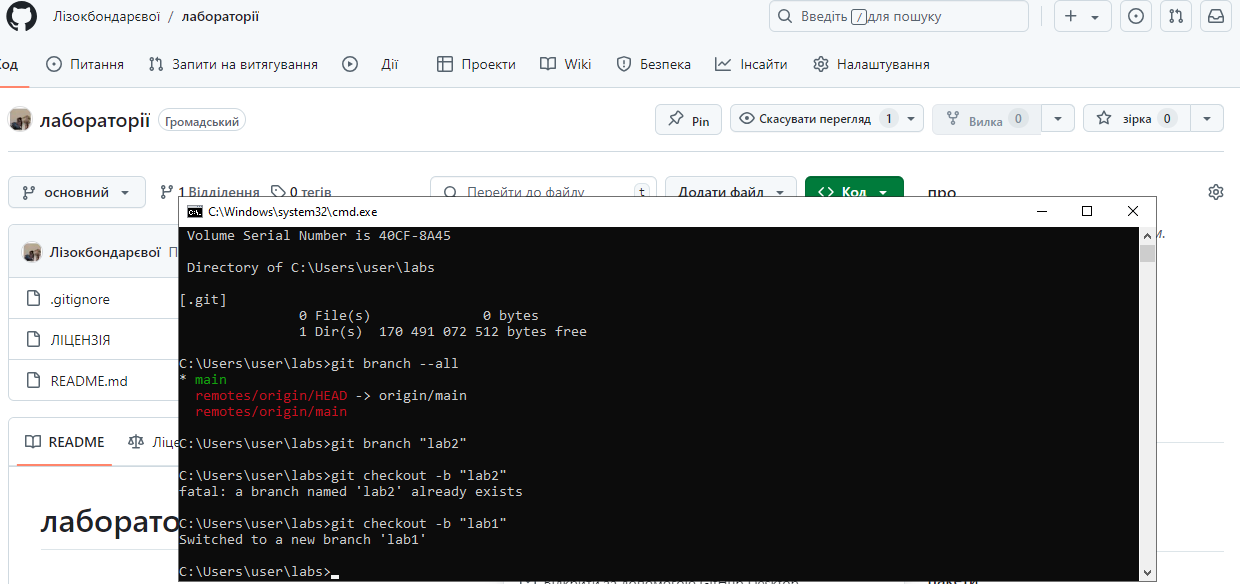
Створюю відкритий (public) репозиторій на хостингу GitHub(рис.4).

A screenshot of a computer

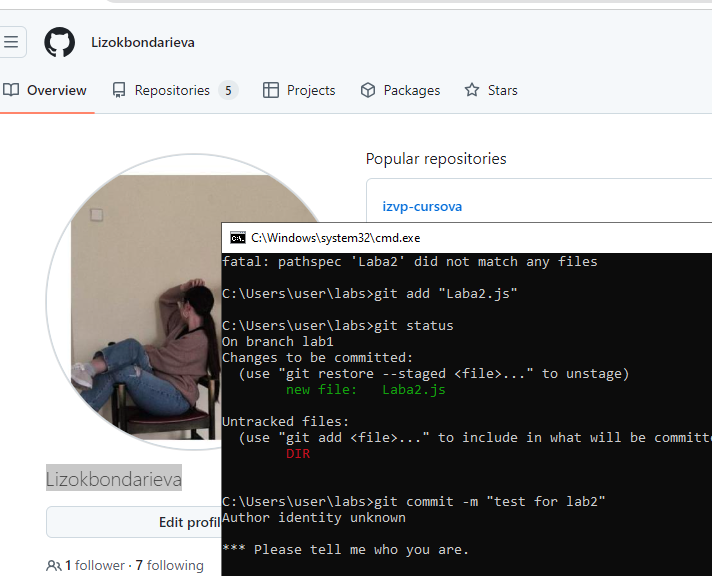
Description automatically generated

Рисунок 4 – Новий репозиторій labs

Клоную створений мною репозиторій на локальний комп’ютер (рис.5). І створюємо нову гілку testing.

Рисунок 5 – Клонований репозиторій і створення нової гілки

Додаю до репозиторія на створену гілку файл з кодом і фіксування змін(рис.6).

Рисунок 6 – Додавання файлу

A screenshot of a computer

Description automatically generated

Рисунок 7 – Фіксація змін

Перегляд доданого файлу на нову створену гілку(рис.8).

A screenshot of a computer

Description automatically generated

Рисунок 8 – Доданий файл на новій гілці

Поєднуємо код з запиту на зміни з головною гілкою(рис.9).

A screenshot of a computer

Description automatically generated

Рисунок 9 - Поєднання код з запиту на зміни з головною гілкою

A screenshot of a chat

Description automatically generated

Рисунок 9 - Виконання команди git merge

A screenshot of a computer

Description automatically generatedРисунок 10 – Пропозиція видалення гілки

Надіслати викладачу посилання на створений репозиторій в Git:

<https://github.com/Lizokbondarieva/labs>

Висновки

В результаті виконання лабораторної роботи мною було створено новий відкритий (публічний) репозиторій створено на ньому нову гілку та додано новий файл. Потім поєднали код з запиту на зміни з головною гілкою.

В ході виконання лабораторної роботи помилок та труднощів не виникло та було досягнуто поставленого результату.

Контрольні запитання

1. Що таке системи контролю версій та їх призначення

Система контролю версій (Version Control System, VCS) — це програмне забезпечення, яке використовується для відстеження змін у файлах та деревах каталогів в процесі розробки програмного забезпечення. Головне призначення систем контролю версій — забезпечення ефективного та організованого спільного редагування та зберігання коду, а також відстеження історії змін. Основні завдання систем контролю версій включають:

1. Відстеження Змін:

- Зберігання історії змін для кожного файлу, включаючи додавання, видалення та модифікації.

2. Спільна Робота:

- Можливість кількох розробників працювати одночасно над спільним проектом, конфлікти якого легко вирішити.

3. Версіонування:

- Зберігання різних версій коду, щоб можна було повернутися до попередніх станів або порівняти різні версії.

4. Ідентифікація Авторства:

- Запис в історії змін інформації про автора та дату внесення кожної зміни.

5. Гілкування та Злиття:

- Можливість створювати відокремлені гілки розвитку для роботи над конкретними функціями чи виправленнями, а потім злити їх разом.

6. Ізоляція Версій:

- Забезпечення ізоляції різних версій проекту одна від одної, щоб зміни в одній гілці не впливали на інші.

7. Тестування та Відновлення:

- Можливість створювати тестові гілки та відновлювати систему до попередніх станів для тестування та відладки.

8. Історія та Аналіз Змін:

- Забезпечення повного журналу всіх змін, що дозволяє аналізувати розвиток проекту та визначати причини помилок.

Системи контролю версій є важливим інструментом у сфері розробки програмного забезпечення, сприяючи спільній роботі розробників, забезпечуючи стабільність та відстеження змін у коді. Вони можуть бути централізованими (наприклад, SVN) або розподіленими (наприклад, Git).

1. Призначення хостингів проектів

Хостинг проектів в інформаційних технологіях вказує на забезпечення інфраструктури та сервісів для зберігання, управління та взаємодії з кодовою базою проекту. Призначення хостингів проектів включає в себе декілька аспектів:

1. Зберігання Коду:

- Хостинг проектів надає місце для зберігання кодової бази. Це дозволяє розробникам спільно працювати над проектом та взаємодіяти з кодом через інтерфейс хостингу.

2. Керування Версіями (Version Control):

- Більшість хостингів проектів використовують системи контролю версій (наприклад, Git), що дозволяє відстежувати та керувати версіями коду.

3. Спільна Робота та Колаборація:

- Забезпечення можливості спільної роботи розробників над проектом, обговорення змін, виправлення помилок та прийняття змін (pull requests).

4. Проектні Функції:

- Багато хостингів надають додаткові функції для керування проектом, такі як ведення завдань, відслідковування помилок, ведення документації тощо.

5. Інтеграція Засобів Розробки:

- Підтримка інтеграції з популярними засобами розробки, такими як IDE (Integrated Development Environment), CI/CD (Continuous Integration/Continuous Deployment) системами тощо.

6. Безпека та Доступ:

- Забезпечення захисту кодової бази, обмеження доступу, можливість налаштовувати права доступу до проекту.

7. Хмарне Зберігання:

- Багато хостингів пропонують хмарне зберігання, що дозволяє розробникам отримати доступ до свого коду з будь-якого місця та працювати над проектом в розподілених командах.

8. Автоматичні Збірки та Доставка (CI/CD):

- Підтримка автоматизованих процесів збірки та доставки для швидкої розробки та релізів.

Популярні хостинги проектів включають такі платформи, як GitHub, GitLab, Bitbucket, а також інші. Вони грають важливу роль у розробці програмного забезпечення, дозволяючи розробникам ефективно співпрацювати та керувати кодовою базою проекту.

1. Що таке версія?

У контексті програмного забезпечення та систем контролю версій, термін "версія" вказує на конкретний стан або випуск програми чи проекту в певний момент часу. Кожна версія має свій унікальний ідентифікатор (наприклад, номер версії чи хеш коміту в системі контролю версій), що визначає конкретний набір коду та інших ресурсів, які складають програму на даний момент.

Версії використовуються для керування та відстеження розвитку програмного забезпечення. Вони дозволяють розробникам та користувачам:

1. Відстежувати та Зберігати Історію Змін:

- Запам'ятовувати, як змінюється код та функціонал з плином часу. Кожна версія представляє собою стан проекту в певний момент.

2. Повертатися до Попередніх Версій:

- Відновлювати попередні версії коду чи проекту, якщо потрібно виправити помилки чи повернутися до попереднього функціоналу.

3. Розробляти Новий Функціонал:

- Впроваджувати новий функціонал та зміни у відокремлених версіях для уникнення конфліктів та збереження стабільності.

4. Управляти Процесом Релізів:

- Визначати та маркувати версії для публічного випуску. Номер версії може вказувати на важливість або зміст змін.

Номери версій можуть використовувати різні схеми, такі як "мажорна.мінорна.патч" (наприклад, 1.2.3), дата, алфавітні теги, або інші системи ідентифікації. У контексті систем контролю версій, такі як Git, версії також пов'язані з комітами або знімками коду, які можуть бути використані для відновлення стану проекту на певний момент часу.

1. Що таке репозиторій та як поділяються за місцем розташування?

Репозиторій (Repository):

У термінах систем контролю версій, репозиторій - це місце для зберігання та управління версіями коду та інших ресурсів проекту. Він містить повну історію змін, гілки розвитку, конфігураційні файли, документацію та інші елементи проекту. Репозиторій дозволяє розробникам спільно працювати над проектом, відстежувати зміни та керувати версіями коду.

Розділення Репозиторіїв за Місцем Розташування:

1. Локальний Репозиторій:

- Розташований безпосередньо на комп'ютері розробника. Тут розробник може вносити зміни, експериментувати та комітувати їх перед відправленням на центральний сервер або інший віддалений репозиторій.

2. Центральний Репозиторій:

- Розташований на центральному сервері або хостингу. Це централізоване місце для обміну та спільної роботи розробників. Тут зберігається основна версія проекту.

3. Віддалений Репозиторій:

- Репозиторій, який розташований на іншому сервері або хостингу, відмінному від центрального сервера. Розробники можуть використовувати віддалені репозиторії для обміну своїми змінами та спільної роботи над груповими проектами.

4. Гітуб, Гітлаб, Бітбакет (GitHub, GitLab, Bitbucket):

- Онлайн-платформи для зберігання та управління репозиторіями. Вони надають інтерфейс для роботи з Git, інструменти для спільної роботи над проектами, відстежування задач, CI/CD та інші функції.

5. Децентралізовані Репозиторії:

- Системи контролю версій, які дозволяють розробникам працювати з різних копій репозиторію, навіть якщо немає доступу до централізованого сервера. Приклади: Git з можливістю використання локальних та віддалених репозиторіїв.

Комбінування цих різновидів репозиторіїв дозволяє розробникам ефективно працювати над проектами в розподілених та колективних середовищах.

1. Різниця між Git, GitHub, GitLab?

Git:

- Опис: Git - це розподілена система контролю версій (VCS), розроблена Лінусом Торвальдсом. Вона призначена для відстеження змін у файловій системі та спільної роботи розробників над проектами.

- Розташування: Локальний на комп'ютері кожного розробника.

GitHub:

- Опис: GitHub - це хостинг-платформа для Git-репозиторіїв. Вона надає інтерфейс для зберігання, управління та спільної роботи над проектами, використовуючи Git.

- Розташування: Онлайн, це веб-сервіс, на якому можна зберігати та спільно працювати з Git-репозиторіями.

GitLab:

- Опис: GitLab - це інша хостинг-платформа для Git-репозиторіїв, аналогічна GitHub. Проте, GitLab також надає інші функції, такі як власна система CI/CD, управління задачами, а також можливість встановлення на власний сервер (GitLab Community Edition).

- Розташування: Може бути використано онлайн (GitLab.com) або самостійно встановлено на власному сервері.

Основні різниці:

1. Git - це система контролю версій, яка використовується для відстеження змін в коді та спільної роботи розробників локально.

2. GitHub та GitLab - це хостинг-платформи для Git-репозиторіїв, які дозволяють розробникам зберігати, управляти та спільно працювати над своїми проектами онлайн. GitLab також може бути встановлено на власному сервері.

Обидва GitHub та GitLab надають інтерфейси для взаємодії з Git-репозиторіями, але GitLab також має додаткові функції та можливість встановлення на власному сервері, що може бути корисним для підприємств або команд, які шукають більше контролю та конфіденційності.

1. Яким чином можна створити репозиторій?

Створення репозиторію може відбуватися на різних платформах, таких як GitHub, GitLab, Bitbucket або навіть на локальному комп'ютері. Нижче наведені загальні кроки для створення репозиторію на GitHub та GitLab, двох популярних хостинг-платформах для Git-репозиторіїв.

На GitHub:

1. Увійдіть або Зареєструйтеся:

- Увійдіть на свій обліковий запис на GitHub або зареєструйтеся, якщо у вас його немає.

2. Створення Репозиторію:

- Натисніть на кнопку "New" у верхньому правому куті сторінки.

- Введіть ім'я для свого репозиторію та заповніть інші налаштування.

- Оберіть опцію для створення `Public` або `Private` репозиторію.

- Натисніть кнопку "Create repository".

3. Копіювання URL Репозиторію:

- Після створення репозиторію скопіюйте URL-адресу, яка виглядає приблизно так: `https://github.com/username/repository.git`.

На GitLab:

1. Увійдіть або Зареєструйтеся:

- Увійдіть на свій обліковий запис на GitLab або зареєструйтеся, якщо у вас його немає.

2. Створення Проекту:

- Клацніть на плюс (+) у верхньому меню та оберіть "New project".

- Оберіть опцію "Create blank project".

- Введіть ім'я для свого проекту та заповніть інші налаштування.

- Натисніть кнопку "Create project".

3. Копіювання URL Репозиторію:

- Після створення проекту скопіюйте URL-адресу, яка виглядає приблизно так: `https://gitlab.com/username/project.git`.

Зауважте, що після створення репозиторію вам може бути запропоновано додати початковий `README.md` файл, вибрати ліцензію чи інші налаштування. Після створення репозиторію ви можете почати додавати та комітувати свій код.

1. Для чого призначені гілки?

Гілки (branches) в системах контролю версій, таких як Git, використовуються для розділення розвитку програмного проекту на ізольовані лінії розробки. Головна гілка, яка створюється за замовчуванням, часто називається `master` (або `main` в більш сучасних практиках), і вона представляє стабільний стан проекту. Гілки використовуються для таких цілей:

1. Розділення Розвитку:

- Гілки дозволяють розробникам працювати над різними функціями, властивостями чи виправленнями в ізольованих середовищах, не впливаючи один на одного.

2. Експерименти та Функціональність:

- Розробники можуть створювати гілки для впровадження нової функціональності чи проведення експериментів без порушення стабільності основної гілки.

3. Виправлення Помилок:

- Для виправлення помилок можна створити окрему гілку від головної, внести зміни та злити їх назад, щоб не впливати на роботу інших розробників.

4. Ізоляція Версій:

- Гілки дозволяють ізолювати версії проекту, наприклад, для підтримки стабільності поточної версії, тоді як на іншій гілці можуть відбуватися розробка нових функцій.

5. Спільна Робота:

- Кілька розробників може спільно працювати над однією гілкою, об'єднуючи свої зміни пізніше.

6. Тестування та Релізи:

- Створення гілок для тестування нових функцій чи підготовки до релізу.

7. Історія та Відстеження Змін:

- Гілки зберігають історію змін та дозволяють аналізувати розвиток проекту з різних точок зору.

Гілки використовуються для підтримки організованого та ефективного розробницького процесу, де різні частини проекту можуть розвиватися паралельно, а зміни легко і безпечно вносити в основний код.

1. Стратегії створення та ведення гілок

Стратегії створення та ведення гілок в системах контролю версій, таких як Git, визначають, як розробники організовують та управляють гілками у своїх проектах. Нижче подано кілька поширених стратегій:

1. [Git Flow](https://nvie.com/posts/a-successful-git-branching-model/):

- Мета: Ця стратегія має на меті забезпечити стабільний та організований розвиток проекту. Вона використовує різні гілки для розвитку нових функцій, виправлення помилок та підготовки до релізів.

- Гілки:

- `master` (або `main`): Стабільна гілка.

- `develop`: Гілка розробки.

- `feature/`: Гілки для реалізації нових функцій.

- `bugfix/`: Гілки для виправлення помилок.

- `release/`: Гілки для підготовки до релізу.

- `hotfix/`: Гілки для виправлення критичних помилок на `master` чи `develop`.

2. [GitHub Flow](https://guides.github.com/introduction/flow/):

- Мета: Проста та ефективна стратегія для розвитку, яка підходить для невеликих та швидкорозвиваючих проектів.

- Гілки:

- `main`: Основна гілка для стабільного коду.

- `feature/`: Гілки для реалізації нових функцій.

3. [GitLab Flow](https://docs.gitlab.com/ee/topics/gitlab\_flow.html):

- Мета: Проста стратегія, яка підходить для проектів різного розміру та типу. Використовується одна основна гілка для стабільного коду та роботи над новими функціями.

- Гілки:

- `main`: Основна гілка для стабільного коду.

- `feature/`: Гілки для реалізації нових функцій.

4. [GitLab Topic Branches](https://about.gitlab.com/blog/2014/09/29/gitlab-flow/):

- Мета: Використання невеликих гілок для реалізації конкретної функції чи виправлення помилки.

- Гілки:

- Різні гілки, які створюються для роботи над конкретною задачею.

5. [Trunk-Based Development](https://trunkbaseddevelopment.com/):

- Мета: Стратегія роботи без довгострокових гілок, що спрощує процес розгалуження та ведення коду.

- Гілки:

- Основна робоча гілка (зазвичай, `main` чи `trunk`), до якої надходять всі зміни.

- Можливо використовувати функціональні флаги (feature flags) для введення нових функцій.

Кожна стратегія має свої переваги та недоліки, і вибір конкретної залежить від специфіки проекту, розміру команди, швидкості розробки та інших факторів. Важливо обговорити та прийняти стратегію роботи з гілками, яка найкраще відповідає потребам вашого проекту.

1. Основні команди управління гілками

У Git і інших системах контролю версій існують команди, які дозволяють розробникам створювати, переглядати, змінювати та об'єднувати гілки. Ось деякі основні команди управління гілками в Git:

1. Створення Гілки:

- Для створення нової гілки використовується команда `git branch`:

```bash

git branch <ім'я\_гілки>

```

- Або одразу перейти на нову гілку:

```bash

git checkout -b <ім'я\_гілки>

```

2. Перегляд Гілок:

- Команда `git branch` виводить список всіх гілок, позначаючи поточну гілку зірочкою:

```bash

git branch

```

3. Перехід до Іншої Гілки:

- Для переходу на іншу гілку використовується команда `git checkout`:

```bash

git checkout <ім'я\_гілки>

```

4. Створення та Перехід до Нової Гілки (одним кроком):

- Використовуйте `-b` з командою `git checkout` для створення нової гілки та переходу на неї:

```bash

git checkout -b <ім'я\_гілки>

```

5. Видалення Гілки:

- Команда `git branch -d` видаляє вказану гілку:

```bash

git branch -d <ім'я\_гілки>

```

- Для видалення гілки без перевірки на об'єднані зміни використовуйте `-D`:

```bash

git branch -D <ім'я\_гілки>

```

6. Об'єднання Гілок (Merge):

- Команда `git merge` об'єднує зміни з іншої гілки в поточну:

```bash

git merge <ім'я\_гілки>

```

7. Перейменування Гілки:

- Використовуйте команду `git branch -m` для перейменування гілки:

```bash

git branch -m <нове\_ім'я\_гілки>

```

8. Перегляд Історії Гілок:

- Команда `git log` з параметром `--graph` виводить історію гілок у графічному форматі:

```bash

git log --graph --oneline --all

```

Ці команди дозволяють розробникам ефективно керувати гілками, використовуючи Git для організації та управління розвитком проекту.

1. Виявлення файлів для індексації

В Git процес індексації (staging) включає в себе вибір та підготовку файлів для подальшого коміту. Щоб вибрати файли для індексації, вам слід використовувати команди, такі як `git add` та `git rm`. Нижче наведено кілька ключових команд для роботи з індексацією:

1. Додавання Файлів до Індексу:

- Додайте всі змінені файли до індексу:

```bash

git add .

```

- Додайте конкретний файл до індексу:

```bash

git add <ім'я\_файлу>

```

2. Додавання Змінених Файлів та Видалення Видалених Файлів:

- Додайте всі змінені файли, включаючи видалені файли:

```bash

git add -A

```

3. Додавання Змінених та Нових Файлів:

- Додайте всі нові та змінені файли (не враховуючи видалених):

```bash

git add --all

```

4. Додавання Змінених Файлів у Зазначений Коміт:

- Додайте змінені файли до певного коміту (підготуйте їх для подальшого коміту):

```bash

git add --patch <ім'я\_файлу>

```

5. Видалення Файлу з Індексу (Unstage):

- Видаліть файл з індексу, але залиште його в робочій директорії:

```bash

git reset <ім'я\_файлу>

```

6. Видалення Файлу з Робочої Директорії та Індексу:

- Видаліть файл з робочої директорії та індексу:

```bash

git rm <ім'я\_файлу>

```

7. Відміна Змін у Робочій Директорії та Індексі:

- Скасуйте всі зміни у робочій директорії та індексі (поверніться до стану останнього коміту):

```bash

git reset --hard

```

Ці команди допомагають керувати тим, які файли включаються у наступний коміт та які ігноруються. За допомогою цих команд ви можете ефективно підготувати ваші зміни для коміту, враховуючи лише ті файли, які ви хочете включити в історію проекту.

1. Фіксація змін

В Git, фіксація змін відбувається за допомогою команди `git commit`. Коміт — це моментальний запис змін у репозиторії, який включає в себе зміни, внесені до індексу. Ось кілька базових прикладів використання команди `git commit`:

Фіксація всіх змін, які додані до індексу:

```bash

git commit -m "Ваш коментар до коміту"

```

Цей приклад фіксує всі зміни, які вже додані до індексу, з коротким коментарем, що пояснює природу змін.

Фіксація змін і вказання файлів у команді:

```bash

git commit -m "Ваш коментар до коміту" <ім'я\_файлу1> <ім'я\_файлу2> ...

```

Якщо ви хочете вказати конкретні файли для коміту, вказуйте їх після коментаря.

Вказання коміта якісніший інтерфейс:

```bash

git commit

```

Ця команда відкриє редактор (зазвичай, це vim або інший встановлений за замовчуванням текстовий редактор), щоб ви могли ввести більш докладний коментар до коміту.

Фіксація та Додавання Змін у Коментарі:

```bash

git commit -am "Ваш коментар до коміту"

```

Цей підхід автоматично додає всі зміни до індексу та фіксує їх у новому коміті з вказаним коментарем.

Кожен коміт має свій унікальний хеш-код, який служить для ідентифікації коміту в історії репозиторію. Важливо давати змістовні та лаконічні коментарі до комітів, щоб інші розробники могли легко розуміти, які зміни внесені в кожному коміті.

1. Запит на зміни у віддалений репозиторій

Запит на зміни у віддалений репозиторій (зазвичай це використовується для об'єднання змін із гілки в основну гілку) використовується за допомогою концепції Pull Request або Merge Request, в залежності від платформи контролю версій, такої як GitHub, GitLab або Bitbucket. Основна ідея полягає в тому, щоб пропонувати власні зміни для обговорення та об'єднання з основною гілкою репозиторію.

Нижче наведено загальні кроки для створення запиту на зміни:

За допомогою GitHub:

1. Створіть нову гілку:

- Спочатку створіть та перейдіть на нову гілку для вашого нового функціоналу чи виправлення.

2. Зробіть та зафіксуйте зміни:

- Внесіть та зафіксуйте свої зміни на цій гілці.

3. Перейдіть на GitHub:

- Зайдіть на веб-сайт GitHub та відкрийте ваш репозиторій.

4. Створіть Pull Request:

- Натисніть на кнопку "New pull request" біля вашої нової гілки.

5. Оберіть Гілку для Об'єднання:

- Виберіть основну гілку, в яку ви хочете злити свої зміни.

6. Створіть Pull Request:

- Надайте опис змін, коментарі, відстежуйте обговорення та натисніть "Create pull request".

За допомогою GitLab:

1. Створіть нову гілку та Зафіксуйте Зміни:

- Створіть та перейдіть на нову гілку, потім зафіксуйте свої зміни.

2. Перейдіть на GitLab:

- Зайдіть на веб-сайт GitLab та відкрийте ваш репозиторій.

3. Створіть Merge Request:

- Натисніть на кнопку "Merge Request" біля вашої нової гілки.

4. Оберіть Гілку для Об'єднання:

- Оберіть основну гілку, в яку ви хочете злити свої зміни.

5. Створіть Merge Request:

- Надайте опис змін, додайте коментарі та натисніть "Submit merge request".

Після цього ваш запит на зміни буде розглядатися іншими розробниками, і вони можуть внести коментарі, висловити зауваження чи злити ваші зміни у основну гілку. Цей процес забезпечує можливість обговорення та рецензування змін перед їх об'єднанням із основним кодом.

**Використані джерела інформації:**

1. Git - Установка Git. Git. URL: <https://git-scm.com/book/ru/v2/Введение-Установка-Git>
2. Git - Downloading Package. Git. URL: <https://git-scm.com/download/win>