Як розгорнути проєкт та працювати з репозиторієм (GitHub)1) [встановлення LTS версії Node.js](#5m7uyj2bt892)

2) [створення репозиторія для командного проєкту](#5aydobx9kedw)   
 3) [клонування репозиторію](#d5j7i9ocgk8v)

4) [налаштування репозиторію](#im36eziybyu9)  
 5) [основні гілки проєкту](#fd1p0a8fuoq)

6) [розробка фіч або багфіксів](#2vyyrlr4uqwy)  
 7) [видалення гілок](#148jn2vlej8o)

8) [прийняття Pull Requests](#hdfd0d65kvur)

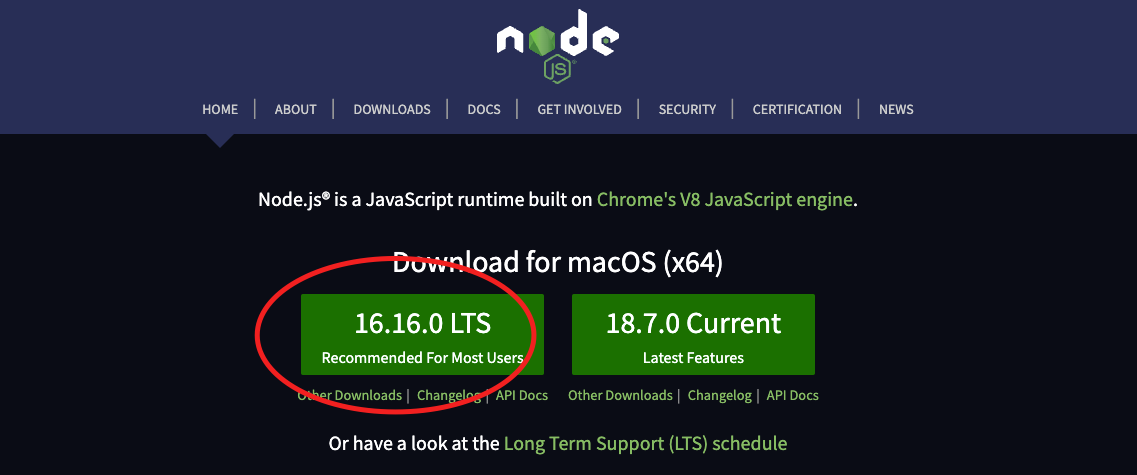
9) [вирішення git-конфліктів](#t0dsaji6omsk)

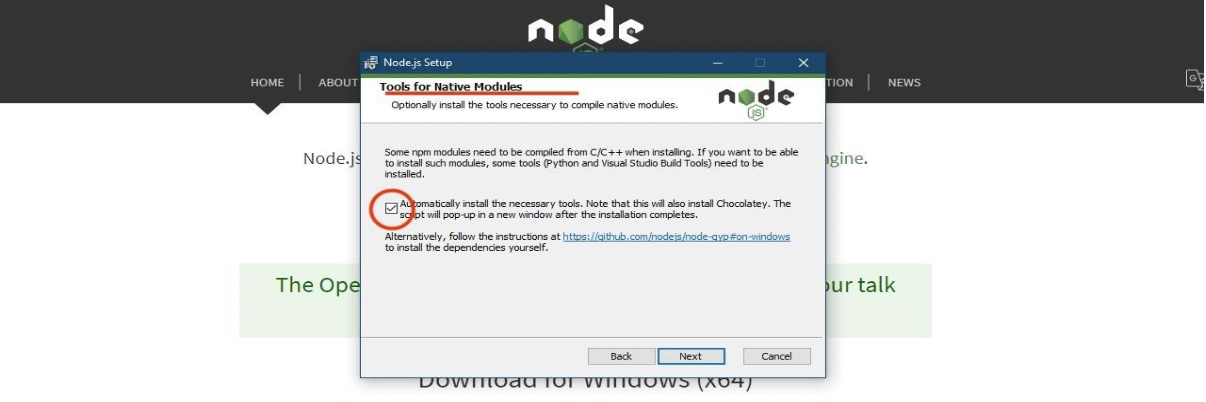
10) [список команд для роботи з терміналом](#e7oa40o63e)

Як Ви пам’ятаєте, **Git** – це програмне забезпечення для контролю версій, яке зробить співпрацю з колегами по команді дуже простою.

Тому розпочнемо саме з

1. **встановлення LTS версії Node.js усіма членами команди, а саме:**

* перейти до <https://nodejs.org/en/> і встановити останню стабільну версію Node.js (LTS) ****
* почати установку, обов'язково проставивши чек поряд з Automatically install the necessary tools



* якщо попередній етап був пропущений, користувачам Windows інструменти потрібно встановити через консоль вручну

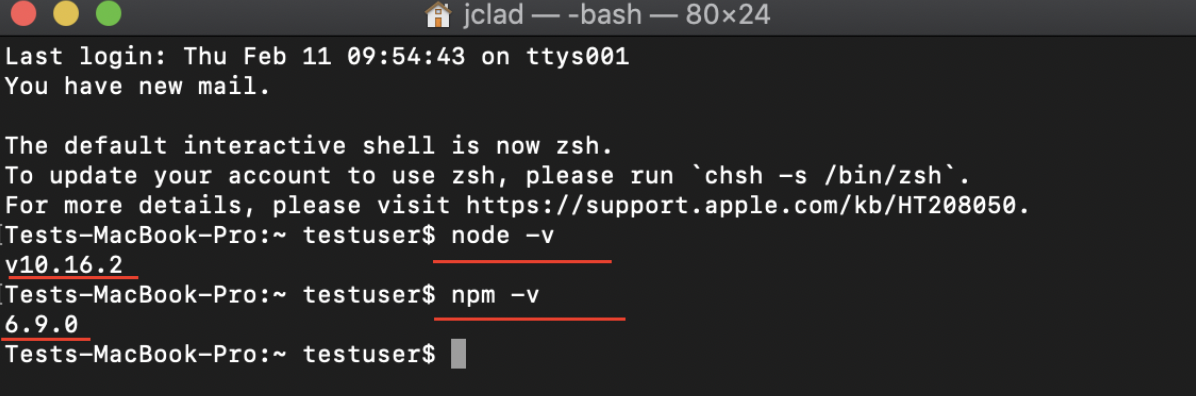
\* для запуска Powershell від імені адміністратора введіть в терміналі

*npm install --global --production windows-build-tools*

\*щоб перевірити, чи встановлено Node.js, введіть у консоль команди

*npm node -v* та  
 *npm -v*

У відповідь на команду відобразиться встановлена ​​версія Node та npm



**ВАЖЛИВО!!!** Користувачам Windows потрібно, щоб була версія Windows вище 7-ї.

1. **створення репозиторія для командного проєкту**

**Тімлід** створює **пустий публічний** репозиторій для проєкту **на своєму** GitHub-акаунті

1. **клонування репозиторію**

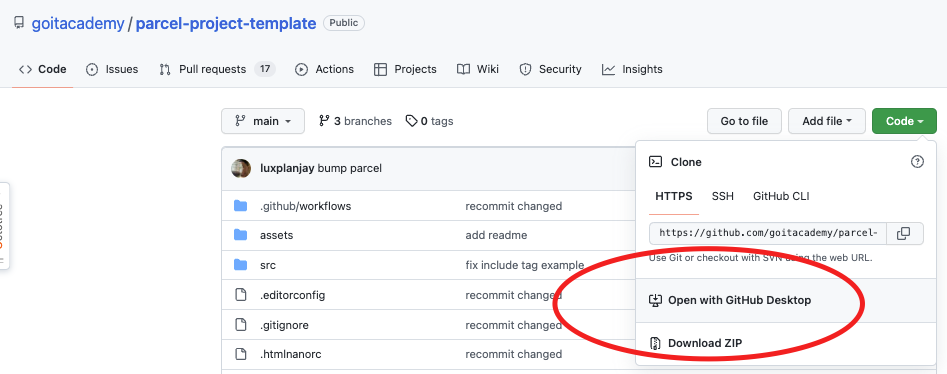
Наразі, існує декілька способів взаємодії з Git:

1. **взаємодія через GitHub Desktop** (для початківців)

Скрінкаст: <https://www.youtube.com/watch?v=sxeW_z5-sq4>

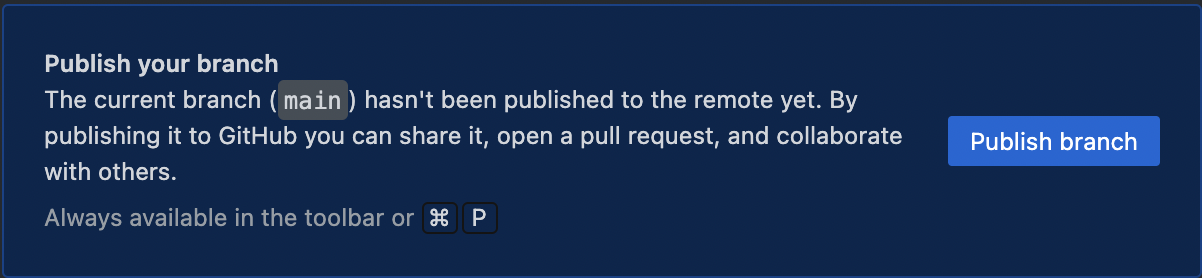
* **Тімлід** клонує репозиторій собі на машину <https://github.com/goitacademy/parcel-project-template> (етап HTML/CSS/JS)

<https://github.com/goitacademy/react-homework-template> (eтап React/React+Node.js)

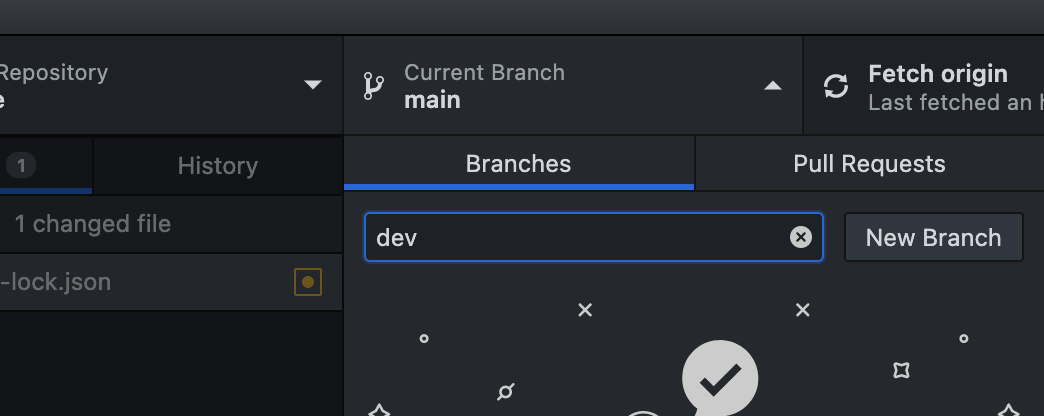


* **Тімлід** змінює ім’я папки (наприклад, з `parcel-project-template`) на особисте ім’я проєкту
* **Тімлід** зв’язує склонований репозиторій з репозиторієм, створеним перед цим для командного проєкту

[https://docs.github.com/en/get-started/getting-started-with-git/managing-remote-repositories#changing-a-remote-repositories-url](https://docs.github.com/en/get-started/getting-started-with-git/managing-remote-repositories#changing-a-remote-repositorys-url)

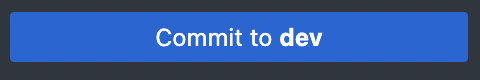
* **Тімлід** переміщає папку проєкта в робоче вікно GitHub Desktop, після чого публікує гілку main  
  

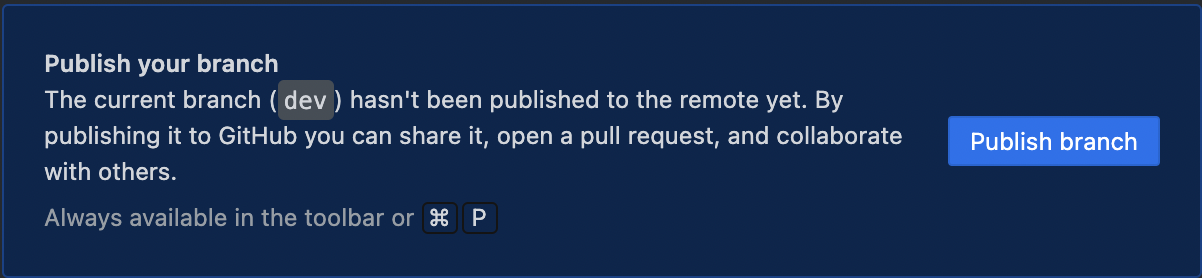
та створює гілку dev



* **Тімлід** через термінал **VSCode** встановлює залежності проєкта в терміналі командою *npm install* тазапускає режим розробки за допомогою команди *npm start*
* **Тімлід** змінює значення у файлі **package.json**

"homepage": "<https://your_username.github.io/your_repo_name/>"  
"scripts": {"build": "parcel build src/\*.html --public-url /your\_repo\_name/"}

* **Тімлід** comitить зміни та заливає на репозиторій  
  



1. **взаємодія через термінал**(найбільш розповсюджений спосіб взаємодії в реальних IT-компаніях)

Скрінкаст: <https://www.youtube.com/watch?v=wFY5HVuQBgw>

* **Тімлід** відкриває VSCode => New Window => New File=> зверху натискає Terminal => New Terminal та за допомогою команд *ls* (показує складові папки) та *cd* (переходить на один крок глибше в древі папок) обирає місце для збереження проекту
* **Тімлід** клонує репозиторій

*git clone* [*https://github.com/goitacademy/parcel-project-template.git*](https://github.com/goitacademy/parcel-project-template.git)(етап HTML/CSS/JS)

*git clone* <https://github.com/goitacademy/react-homework-template> (eтап React/React+Node.js)

* **Тімлід** змінює ім’я папки (наприклад, з `parcel-project-template`) на особисте ім’я проєктута за допомогою команди *cd* в терміналі заходить в папку зклонованого репозиторію
* **Тімлід** зв’язує особистий репозиторій зі зклонованим

*git remote -v*

*git remote set-url origin https://github.com/USERNAME/REPOSITORY.git*

та перевіряє чи все вийшло

​​*git remote -v*

* **Тімлід** за допомогою команди *git push* публікує гілку main
* **Тімлід** створює гілку dev та публікує її

*git checkout -b dev*

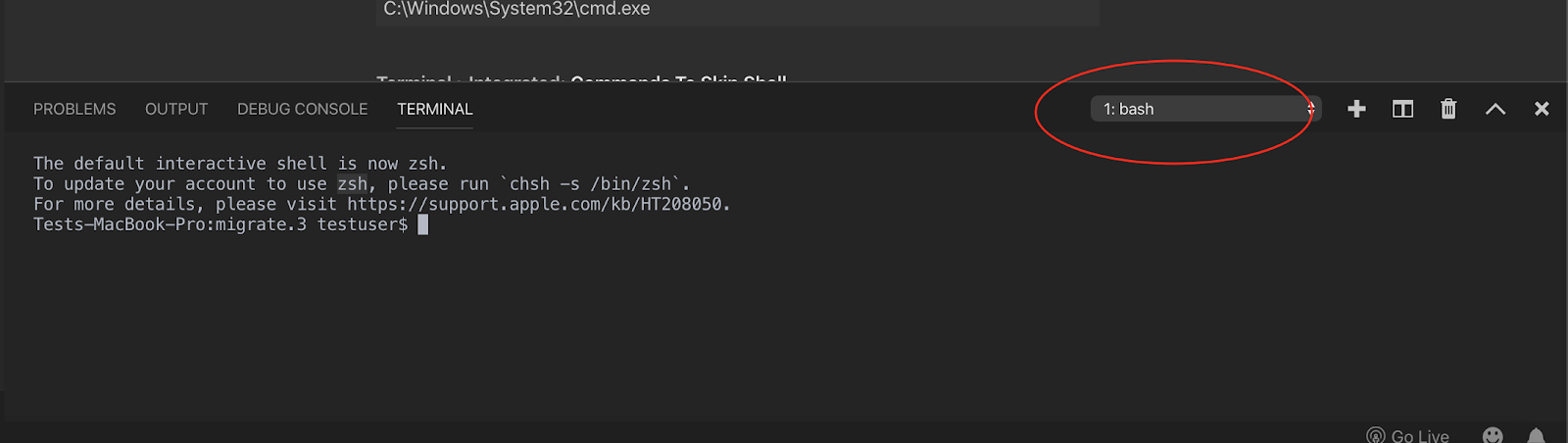
*git push origin dev*

* **Тімлід** встановлює залежності проєкта в терміналі командою

*npm install*

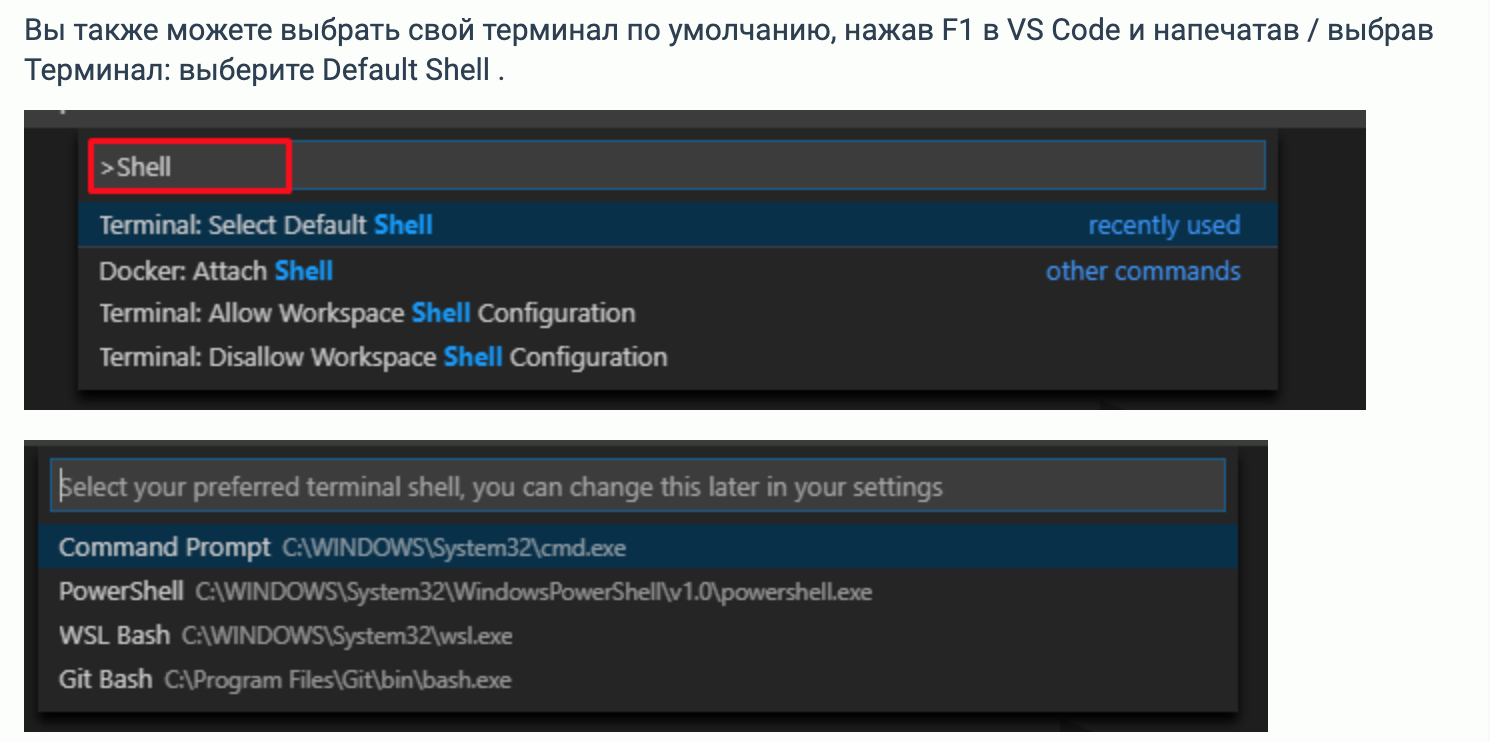
якщо встановлення пройшло успішно, у папці проекту з'явиться нова папка *node\_modules*

Примітка: Переконайтеся, що ви використовуєте bash або zsh термінал.

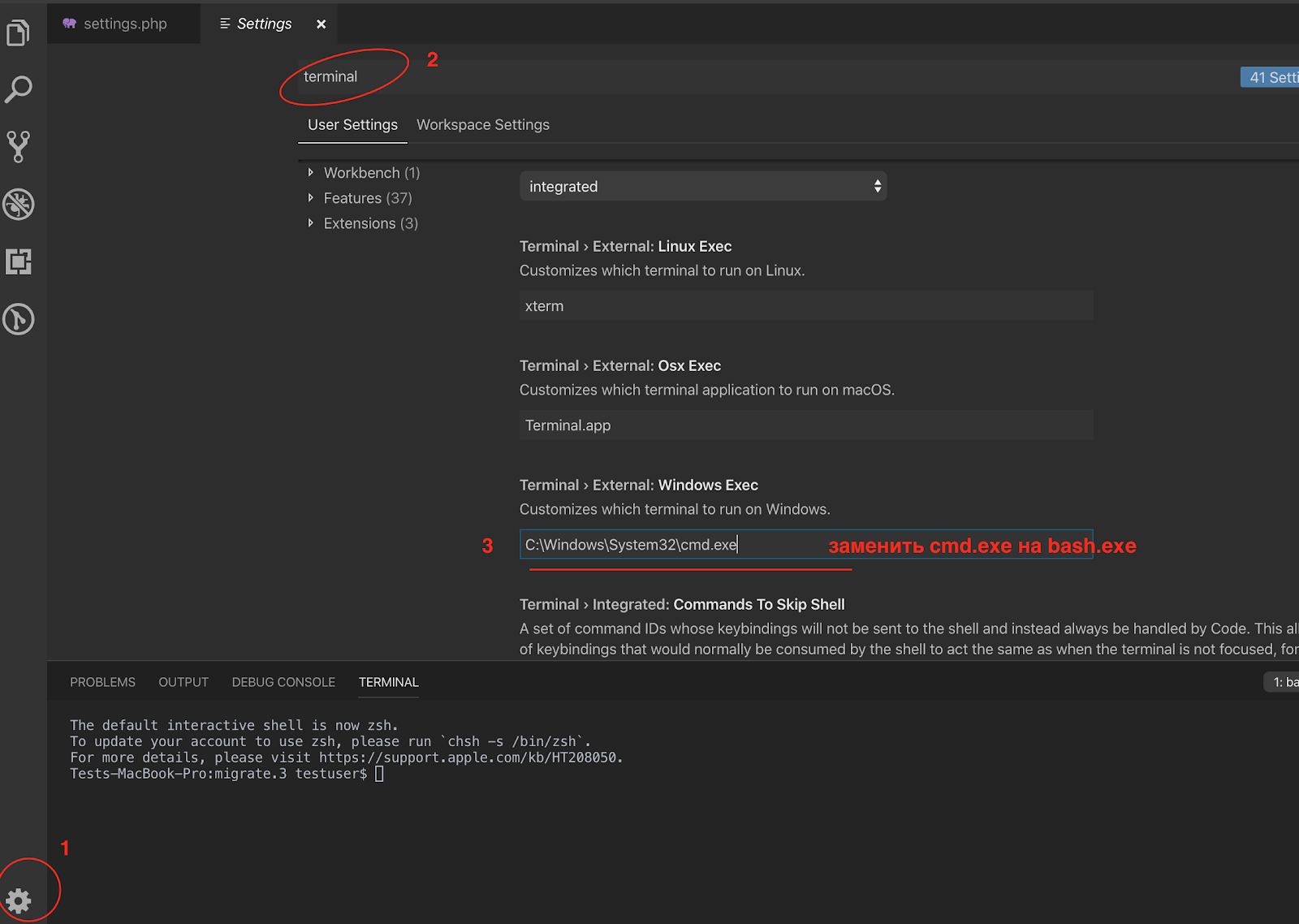


Увага! При спробі встановлення npm-пакетів термінал Power Shell може видавати помилку. Не використовуйте його.

Для того, щоб змінити термінал за замовчуванням

**Варіант 1** 

**Варіант 2**



* **Тімлід** запускає режим розробки за допомогою команди

*npm start*

* **Тімлід** змінює значення у файлі **package.json**

"homepage": "https://your\_username.github.io/your\_repo\_name/",

"scripts": {"build": "parcel build src/\*.html --public-url /your\_repo\_name/"}

* **Тімлід** comitить зміни та заливає на репозиторій

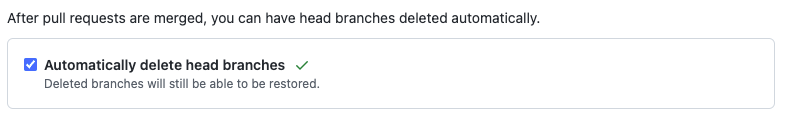
git add .

git commit -m ‘назва-коміту’

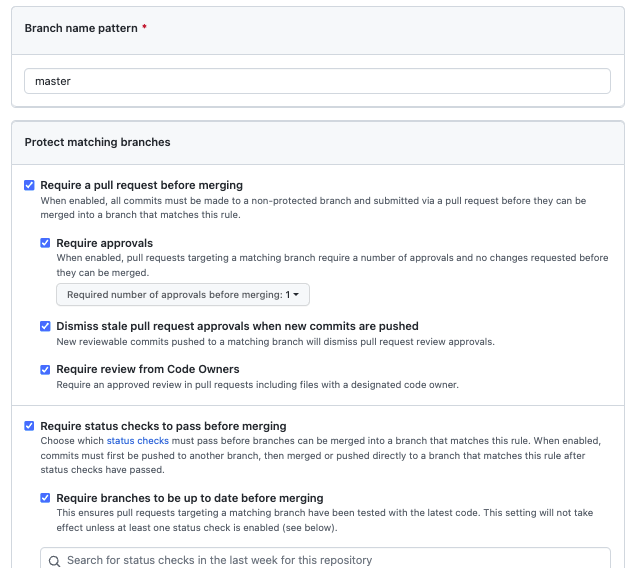
git push origin dev

1. **налаштування репозиторію**  
   **a.** На сторінці вже створеного репозиторію **Тімлід** заходить до **Settings**  та вносить наступні зміни

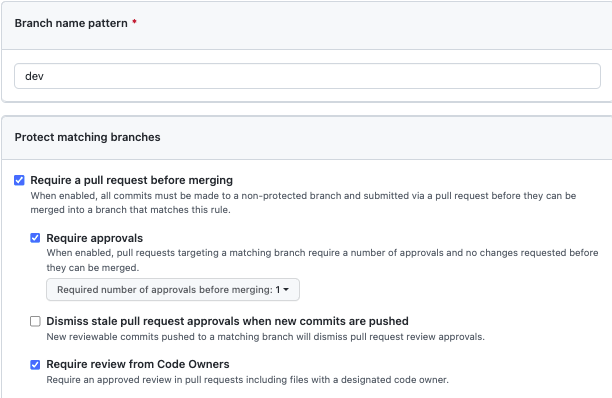
* на вкладці **General** у секції **Pull Requests** ставить чек поряд з **Automatically delete head branches**

****

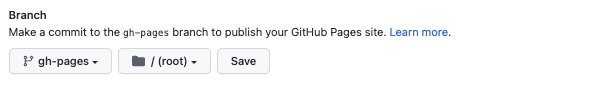
* у вкладці **Branches** у розділі **Branch protection rules** встановлює правила для гілки **master**

****

та правила для гілки **dev**



* на вкладці **Pages** у секції **GitHub Pages** налаштовує хостинг проекту з гілки **gh-pages**



* **Тімлід** у вкладці **Collaborators** додає членів команди в репозиторій. Кожному буде надіслано інвайт на пошту.
* після чого усі розробники з команди клонують репозиторій, створений Тімлідомта виконують в терміналі команду *npm .*

**5) осно****вні гілки проєкту**

### **Гілка main**

У гілці main зберігається робочий вихідний код готового проєкту

**Гілка dev (опціонально)**

Для роботи над проєктом тимлід створює гілку dev. У цю гілку будуть зливатися всі зміни щодо проєкту.

### **Гілка gh-pages**

Вже є у збиранні проєкту. Самостійно створювати її не потрібно. У гілці **gh-pages** зберігається готовий проєкт, який відображатиметься на статичній сторінці **gh-pages**. Файли в цю гілку вивантажуються автоматично, кожного разу, коли оновлюється гілка **main.** Наприклад, після прямого пуша або прийнятого пул-реквесту. Для цього необхідно у файлі **package.json** відредагувати поле **homepage** та скрипт **build**, замінивши ім'я\_користувача та ім'я\_репозиторію на свої.  
  
"homepage": "https://**ім’я\_користувача**.github.io/**ім’я\_репозиторія**",

"scripts": {"build": "parcel build src/\*.html --public-url /**ім’я\_репозиторія**/"},

*Нова гілка створюється під кожне завдання. Після того, як гілка була смерджена, її потрібно видалити локально і на віддаленому репозиторії.*

**6) р****озробка фіч або багфіксів**

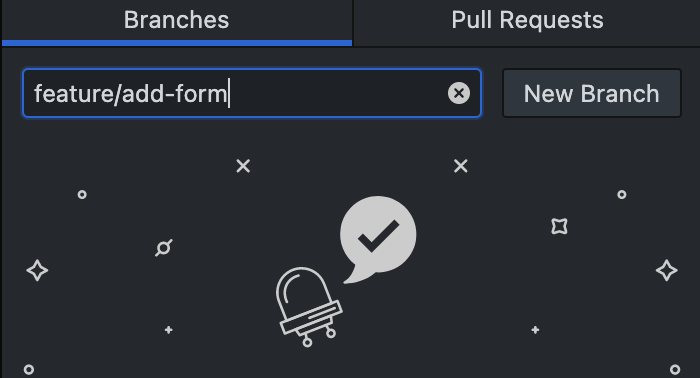
* кожна фіча або багфікс виконується в окремій гілці
* перед початком роботи над новою фічею або багфіксом оновлюємо локальний **dev**, щоб він був ідентичний **dev на GitHub.** В залежності від обраного способу взаємодії з Git

1. переходимо в **GitHub** **Desktop** на гілку **dev** і робимо **fetch+pull,** щоб синхронізувати локальну гілку з тією, що на **GitHub** чи
2. **в терміналі**

*git checkout dev*

*git pull*

* робимо нову гілку з **dev.** Рекомендується давати назву гілки за шаблоном **feature/назва-фічі** та **bugfix/назва-бага**  
  a. в **GitHub** **Desktop**



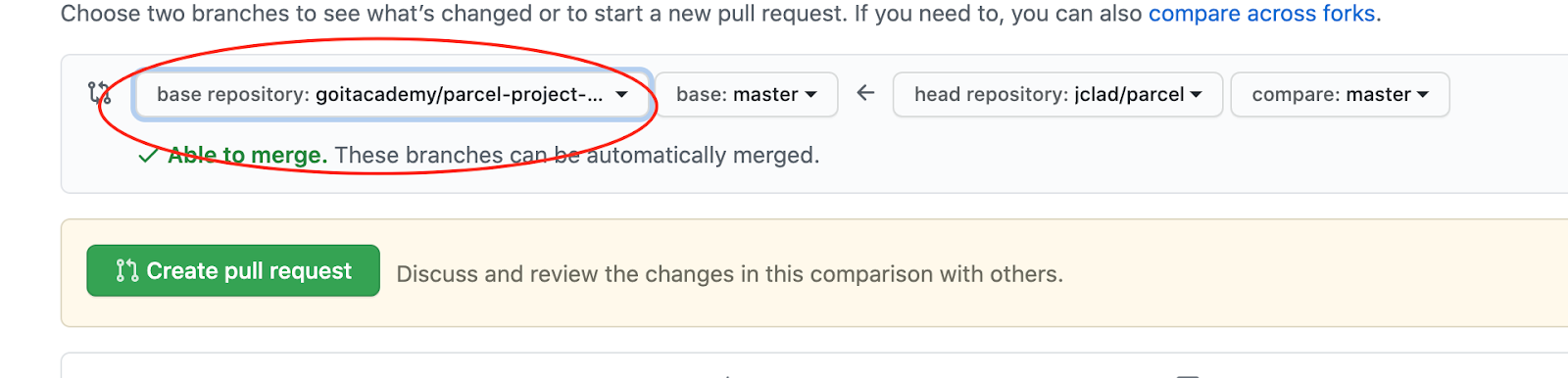
чи

**b. в терміналі**

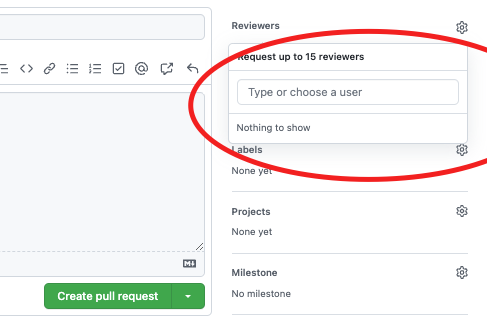
*git checkout -b ‘feature/назва-фічі’*

Працюємо у цій гілці та вносимо необхідні зміни. Комітимо зміни за потребою.

* після того, як фіча готова, відправляємо історію (комміти) локальної гілки на GitHub і відкриваємо **Pull request**

У GitHub, що відкрився, уважно виберіть репозиторій вашого **Tімліда** і натисніть **Create pull request**

* призначаємо мінімум одного **Reviewer,** щоб зробив ревью коду



* якщо необхідні редагування, вносимо їх у тій самій гілці і відправляємо зміни на **GitHub.** Поточний **Pull request** оновиться автоматично, відкривати новий не треба
* після того як **Pull request**  пройшов **review** коду, **Тімлід** приймає його (**merge** у гілку **dev**).

- після того, як **Pull request** був прийнятий, розробник

1. у **GitHub Desktop:** переходить на гілку **dev** і робить **fetch+pull** щоб синхронізувати локальну гілку з тієї що на **GitHub**
2. **в терміналі**

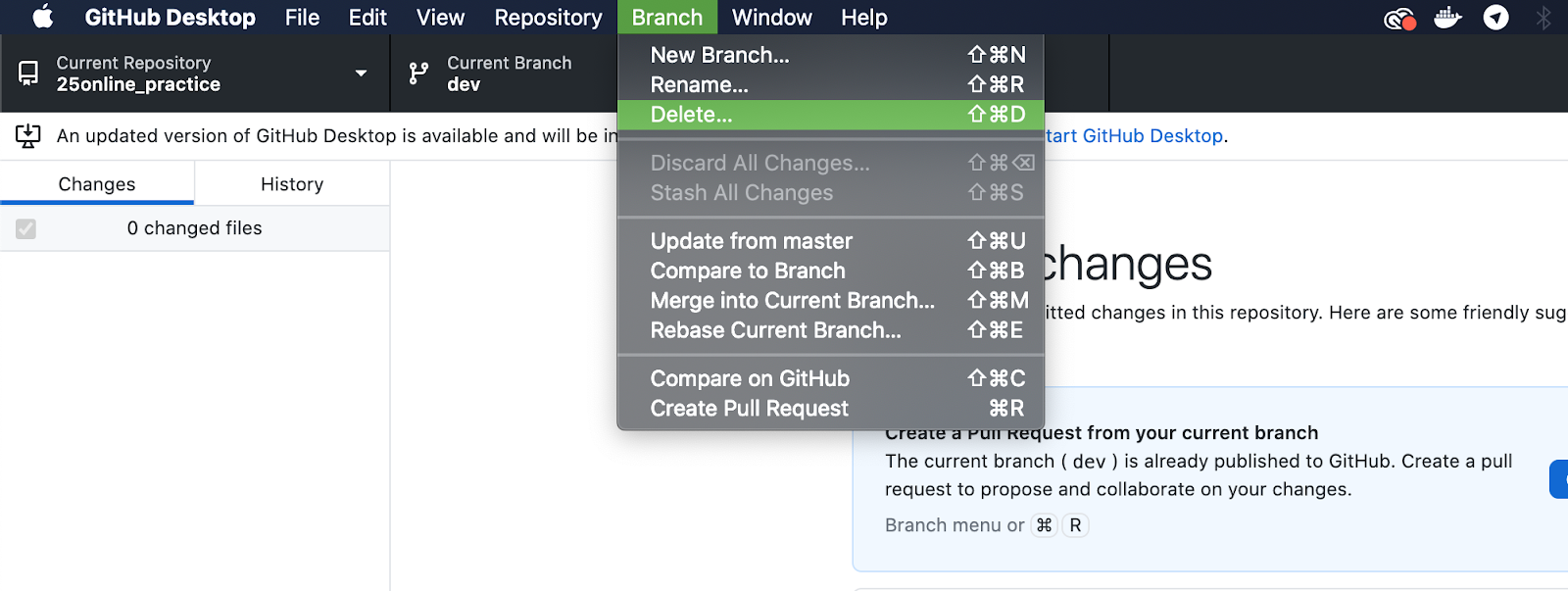
*git checkout dev*

*git pull*

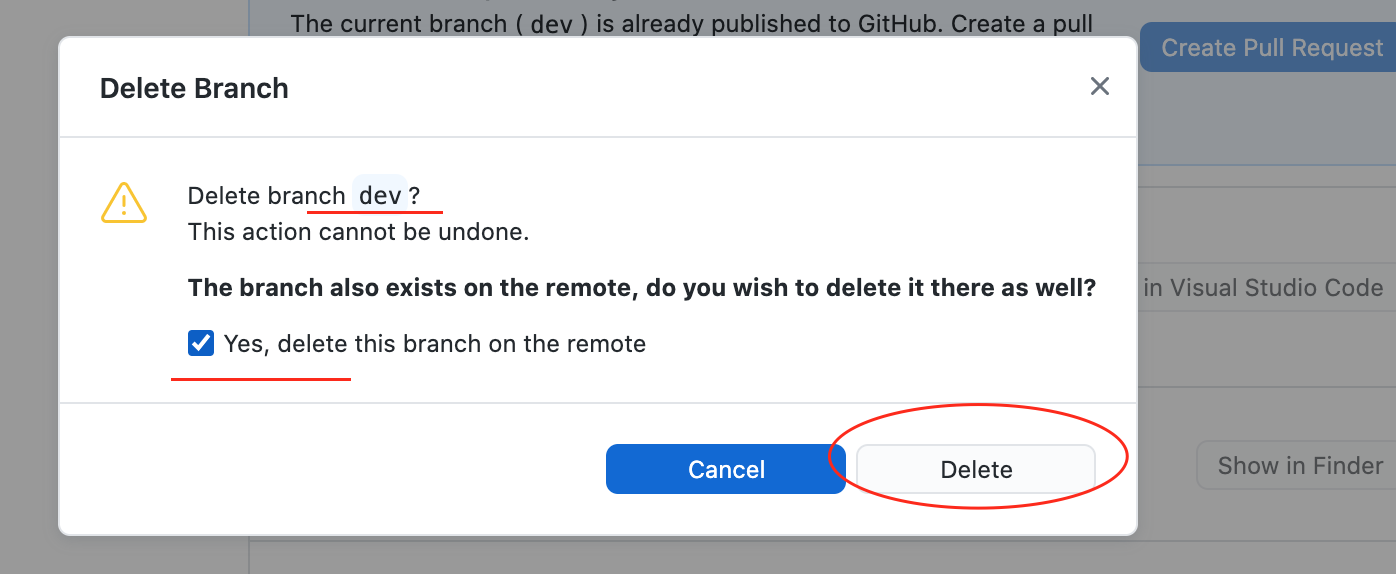
* після синхронізації гілки **dev** розробник видаляє свою локальну гілку фічі/баги

**7)** **видалення гілок**

1. щоб видалити гілку через **GitHub Desktop**, перейдіть до меню Branch > Delete



* переконайтеся, що ви обрали необхідну Вам гілку
* поставте чек, якщо гілку потрібно видалити не лише локально, а й у віддаленому репозиторії
* натисніть Delete

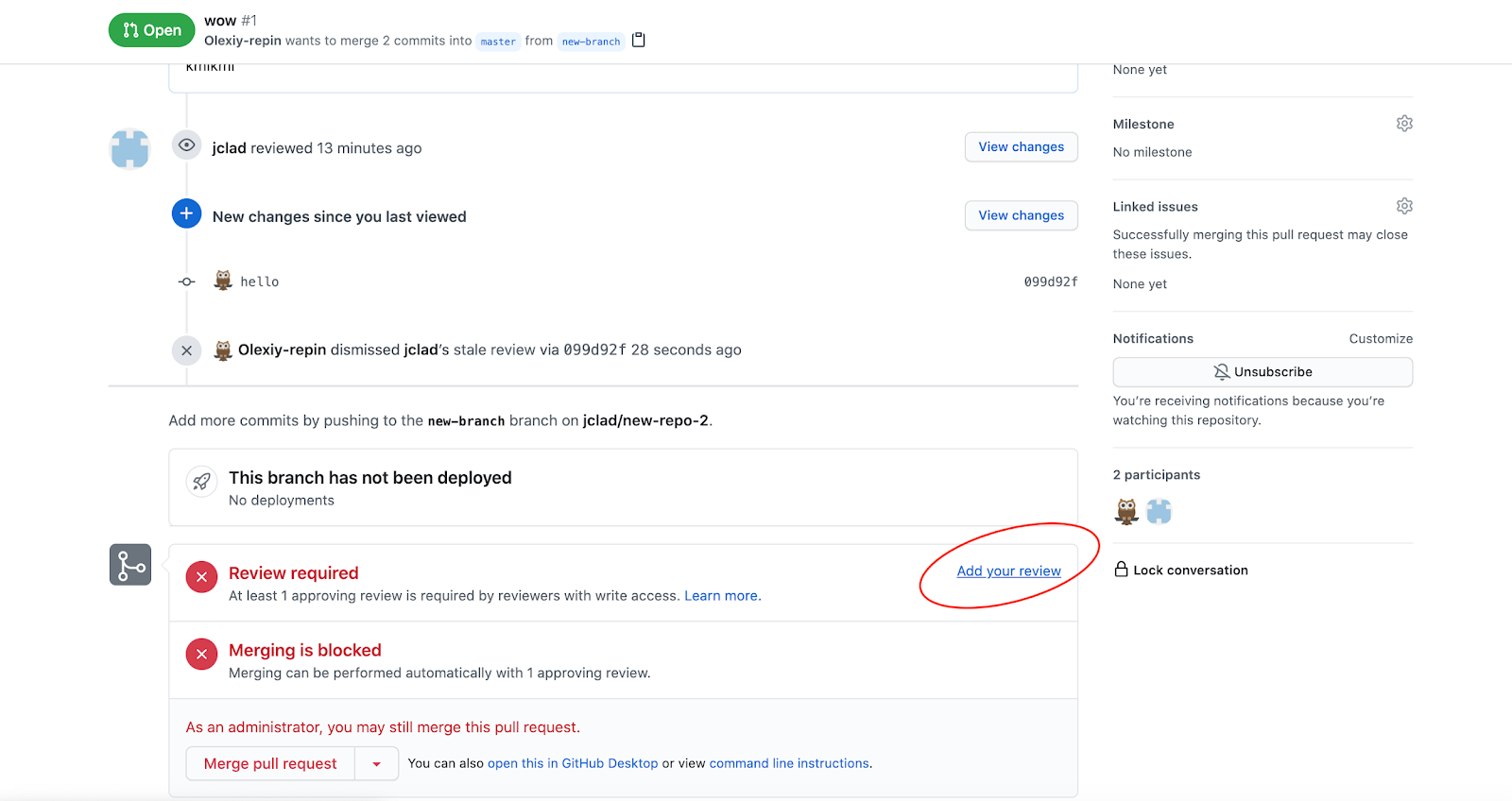


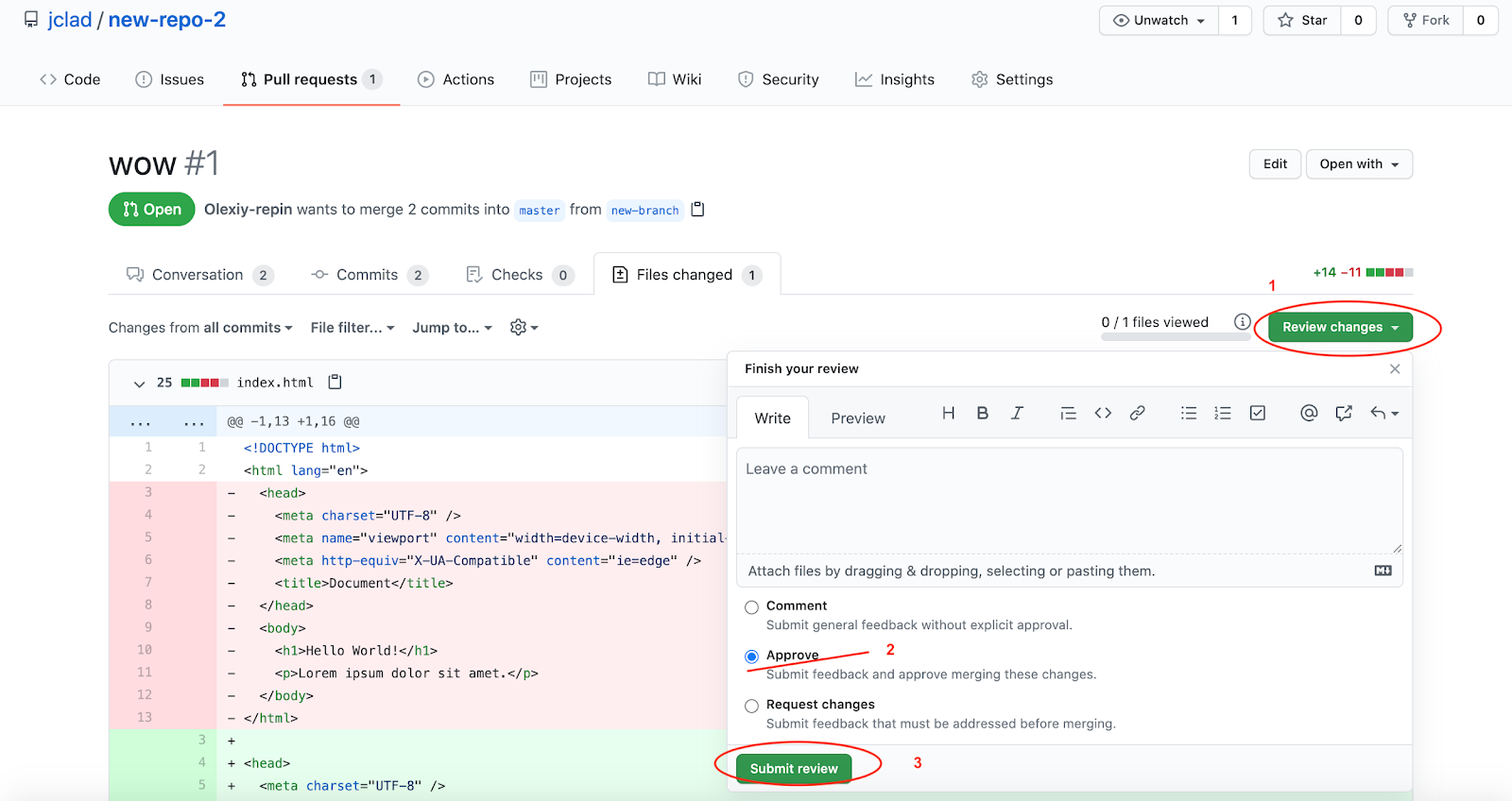
1. щоб видалити гілку через **термінал**

*git branch -d <назва\_гілки>* - видалити гілку

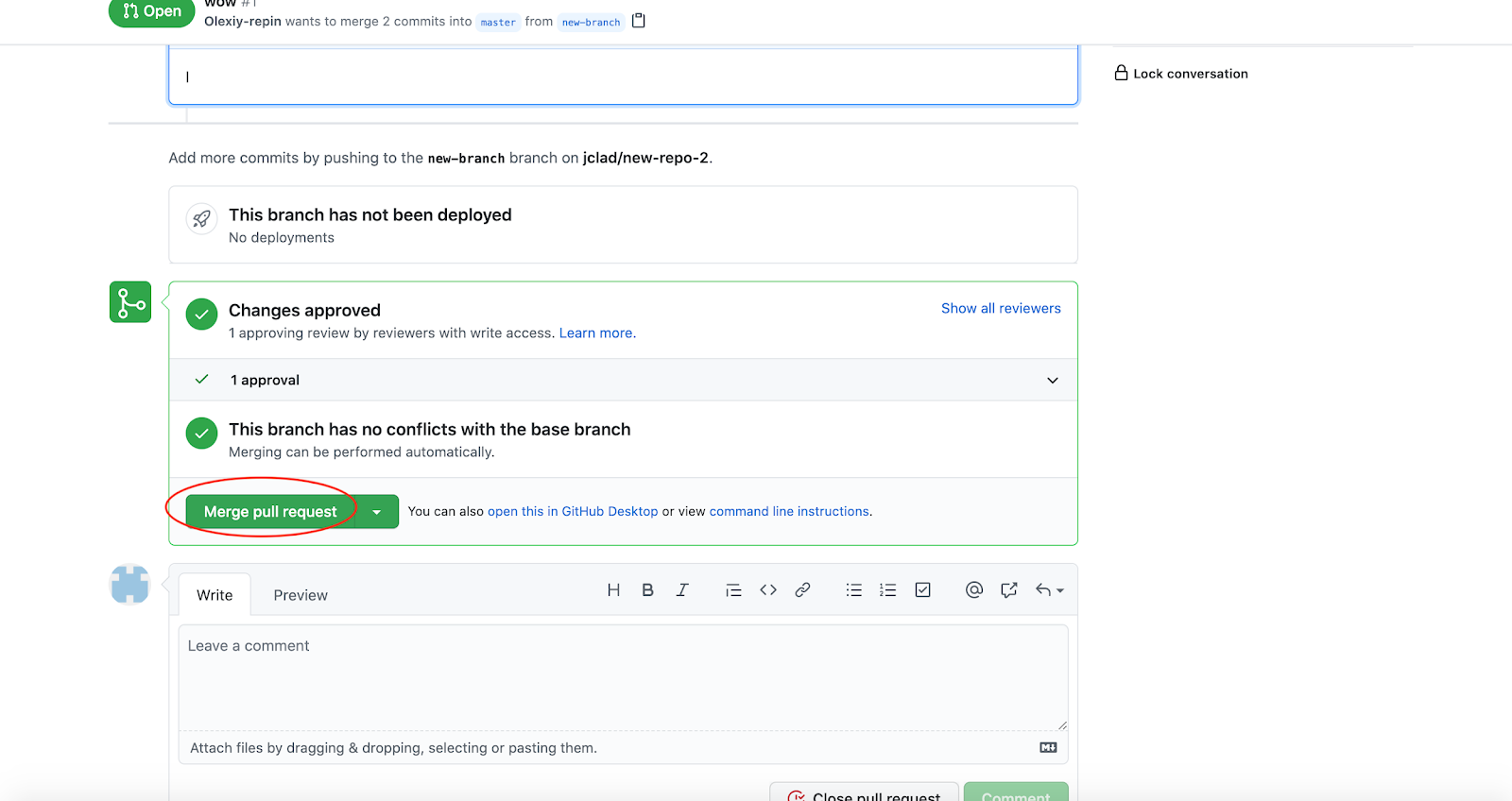
*git push origin --delete <назва\_гілки>* - видалити гілку на віддаленому репозиторії

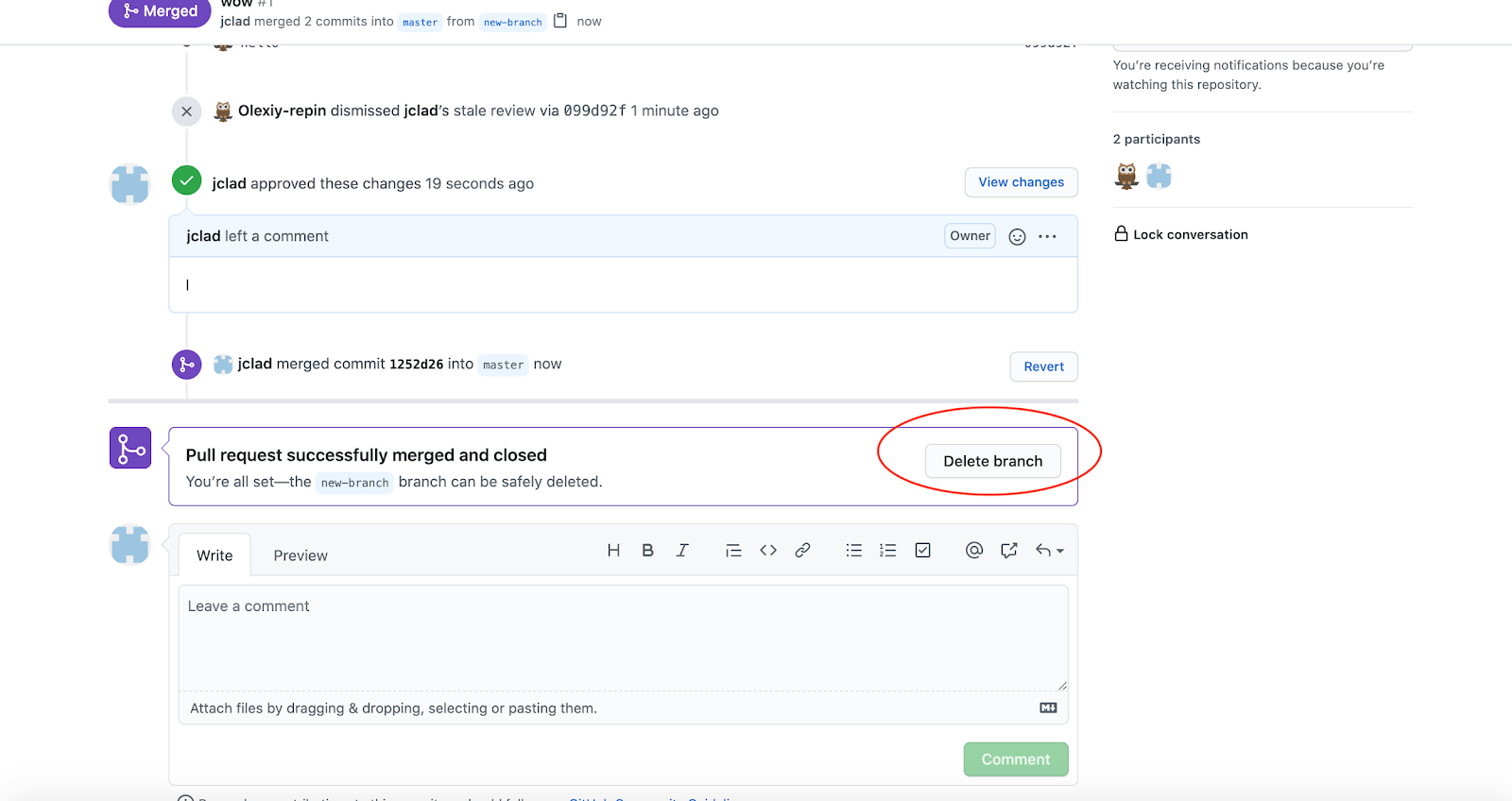
**8)** **прийняття Pull Requests**

* перейдіть до вкладки **Pull Requests**. Виберіть **Add your review**.
* перегляньте код на наявність зауважень та натисніть **Review changes**. Якщо все добре, натисніть **Approve**.



Тепер можна с**merg**ити зміни та прийняти **Pull Request**



Не забувайте видаляти гілки після мерджу!

**9)** **вирішення git-конфліктів**

Конфлікти відбуваються коли два розробники одночасно змінюють один і той же рядок в одному тому самому файлі. Виникають під час **Pull Request**, через що він не може бути успішно cmergeний у базову гілку. Конфлікти бажано вирішувати локально, а не на GitHub. Для цього

1. у **GitHub Desktop** переходимо в гілку **dev** і робимо **fetch + pull**, щоб синхронізувати її з гілкою на GitHub, в якій є всі зміни. Переходимо у гілку фічі/багфікса і зливаємо (merge) до неї гілку dev.
2. у **терміналі***git checkout dev*

*git pull*

*git checkout ‘назва-вашої-гілки’*

*git merge dev*

Вирішуємо конфлікт і робимо коміт. Після чого відправляємо зміни на GitHub, щоб оновити **Pull Request**.

**10)** **список команд для роботи з терміналом**

**git clone https://посилання\_на\_репозиторій** клонуєте репозиторій

**git checkout ‘назва-гілки’** перехід на гілку ‘назва-гілки’

**git pull origin ‘назва-гілки’** стягує останні змінигілки ‘назва-гілки’

**git checkout -b ‘назва-нової-гілки’** створює нову гілку

**git add .** та **git commit -m “назва-коміта”** записує зміни до локальної гілки

**git push origin ‘назва-вашої-гілки’** публікує ваші зміни на гілці віддаленого репозиторію

**git merge ‘назва-гілки’** залиття коду з гілки ‘назва-гілки’ на поточну гілку

**git branch -d ‘назва-гілки’** видалення гілки ‘назва-гілки’

**git fetch** стягує всі зміни з віддаленого репозиторію