**ПРАКТИЧНА РОБОТА 3**

**Основи командної роботи з віддаленим репозиторієм**

**Система оцінювання**

| **№** | **Тема** | **Оцінка** |
| --- | --- | --- |
| 1. | *Завдання 1. Основи роботи з GitHub* | 0,5 |
| 2. | *Завдання 2. Приклади конфліктів при командній роботі* | 1 |
| 3. | *Завдання 3. Побудова GitHub Flow* | 1,5 |
|  | **Всього за практичну** | **3** |
| 4. | ІНДЗ-1. Групова розробка реферативного повідомлення | 4 |
|  | Разом | **7** |

**План**

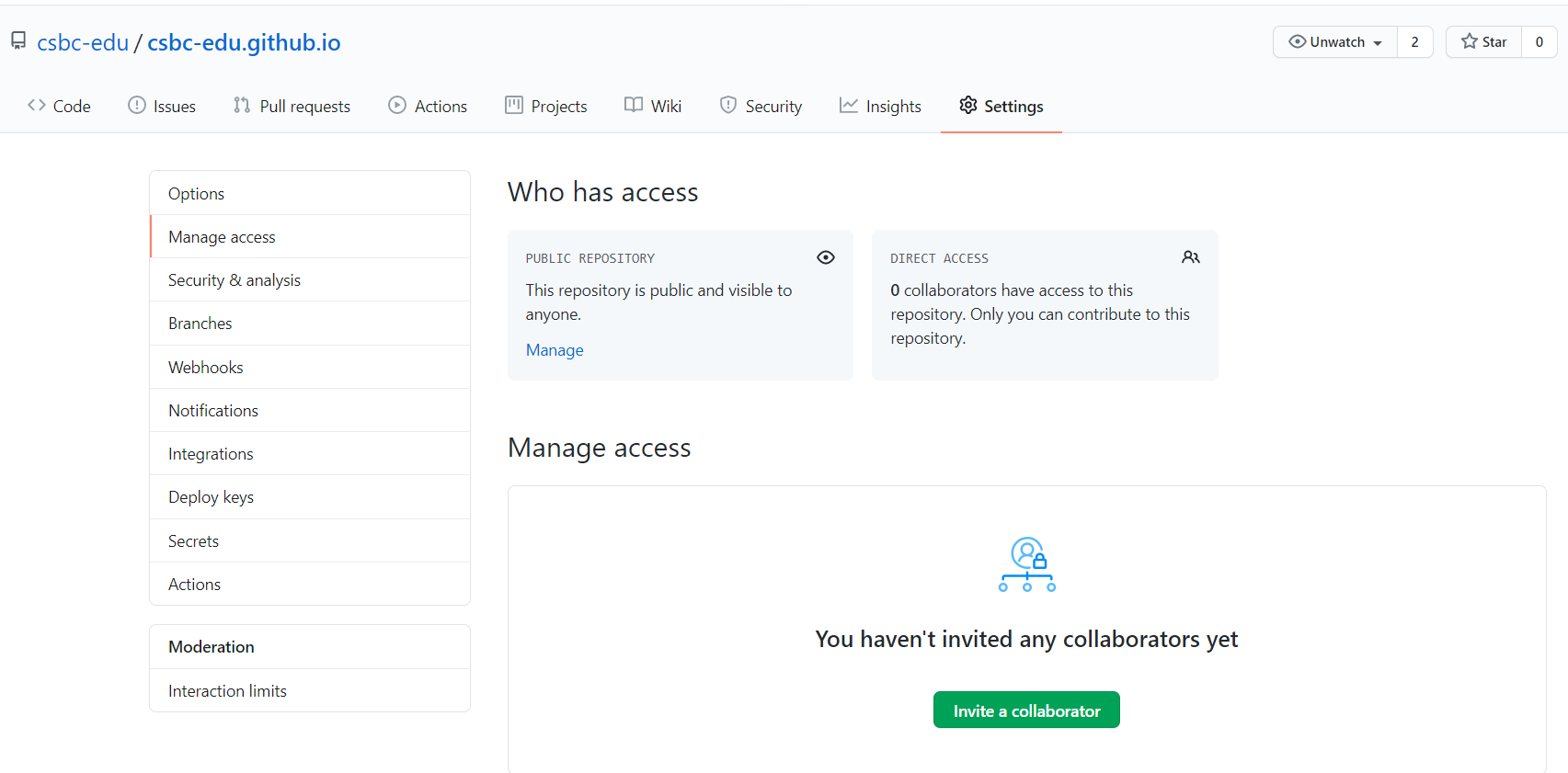
1. Базова взаємодія з сервісом GitHub.
2. Редактори коду та інтеграція з GitHub.
3. Робочий процес GitHub.

**Література:**

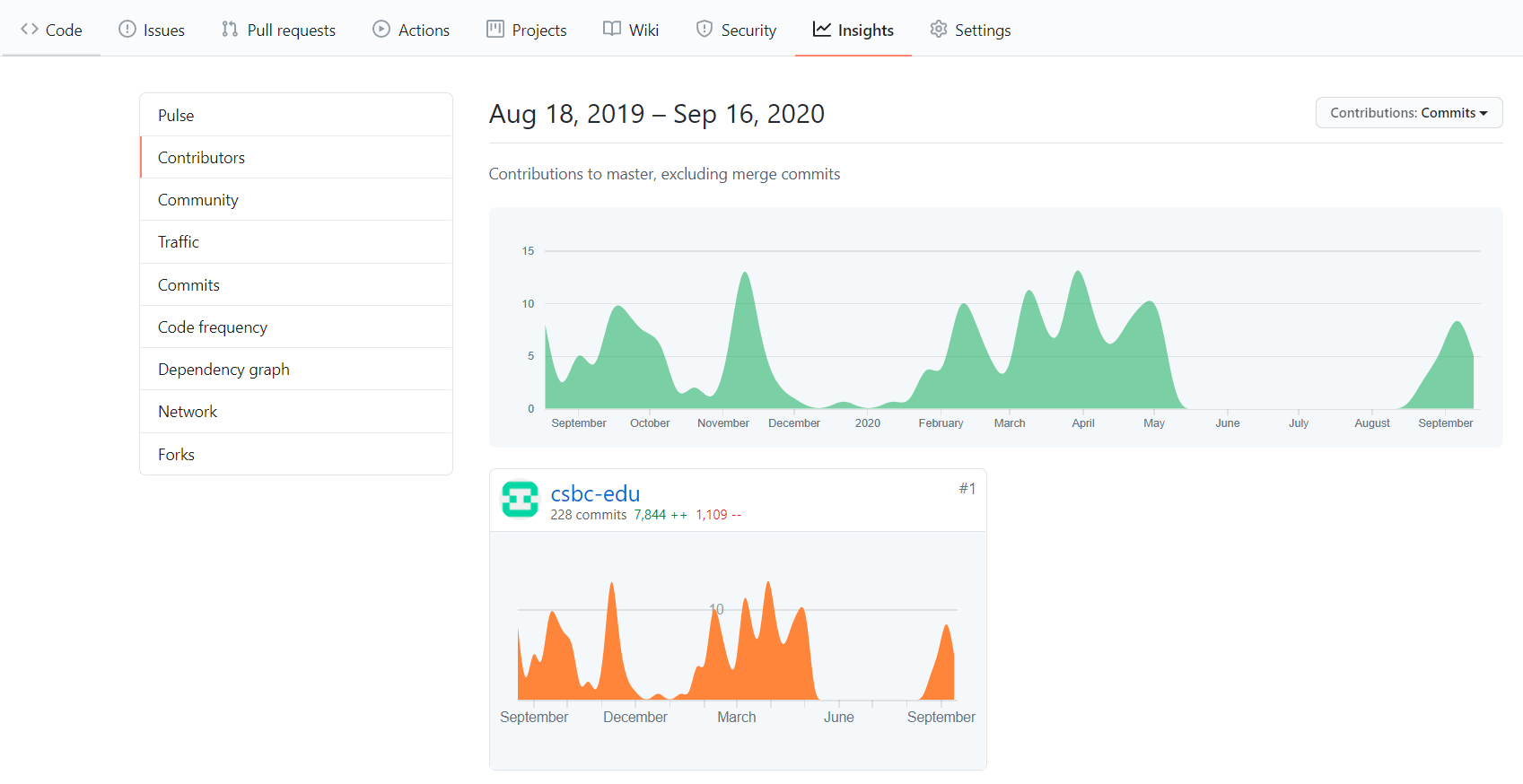
* 1. <https://git-scm.com/book/uk/v2>

***Завдання на практичну роботу:***

1. *Основи роботи з GitHub. Виконайте в консолі наступні операції:*
2. Створіть новий репозиторій локально на своїй машині.
3. Створіть новий GitHub-репозиторій. Назвати його можете довільним чином.
4. Приєднайте Ваш локальний репозиторій до Github-репозиторію.
   * + Необов’язково: перейменуйте типову гілку master на main.
5. Створіть новий файл favorites.txt та залиште його порожнім. Зафіксуйте перший коміт на main-гілці.
6. Запуште Вашу main-гілку на GitHub! Переконайтесь, що отримали порожній файл favorites.txt на GitHub.
7. Далі створіть 2 гілки: foods та movies.
8. Перейдіть на гілку foods. Додайте 3 (або більше) улюблених страви до файлу favorites.txt. Зафіксуйте зміни в гілці foods.
9. Перейдіть на гілку movies та додайте 3 або більше улюблених фільми до файлу favorites.txt file. Зафіксуйте зміни в гілці movies.
10. Запуште гілку foods на GitHub. Переконайтесь, що її стане видно!
11. Запуште гілку movies на GitHub. Переконайтесь, що її стане видно!
12. Виконайте злиття гілки foods з main. Потім злийте гілку movies у main. За потреби вирішіть конфлікт так, щоб улюблені страви та фільми знаходились в одному файлі – favorites.txt.
13. Запуште останні операції в гілку main на Github.
14. Спробуйте налагодити роботу зі спільним репозиторієм. Об’єднайтесь навколо одного репозиторію командами по 2-3 особи та організуйте доступ команди до нього. На прикладі GitHub-репозиторія розгляньте процес додавання співавторів для управління репозиторієм:



Запросіть команду до спільного внесення змін у наповнення репозиторія. Динаміку внесених змін до проекту можна побачити на вкладці Insights, пункт Collaborators

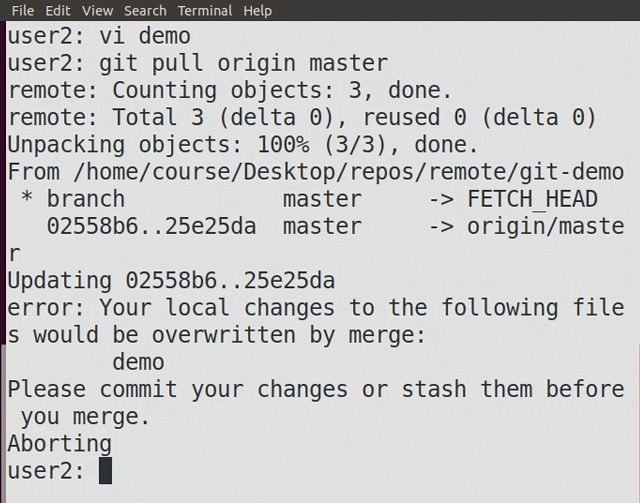


Змоделюйте наступні конфлікти при роботі зі спільним репозиторієм.

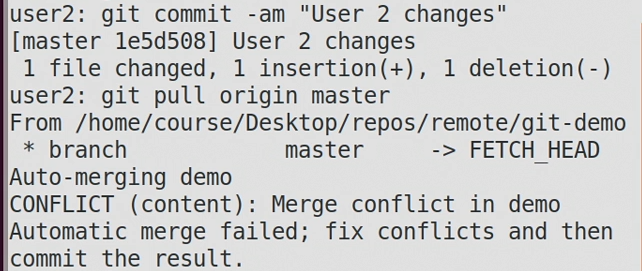
***2.1. Конфлікт: Pulling with Untracked Changes***. Відбувається управління одним файлом для двох користувачів.

Користувач 1: здійснив зміни в файлі та виконав коміт у вітку master.

Користувач 2: спочатку вніс та локально зберіг свої зміни у файл (версія без змін користувача 1), а потім вирішив здійснити пулінг з центрального репозиторію.

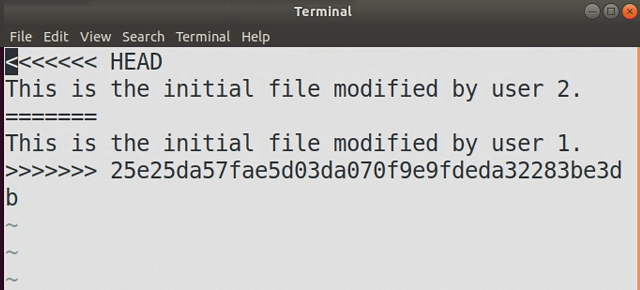


Нові користувачі вважають, що можуть здійснювати пулінг у будь-який момент. Для цього потрібно мати *або* чисту робочу папку (команда git reset --hard), *або* зафіксувати (коміт) свої зміни та виконати пулінг

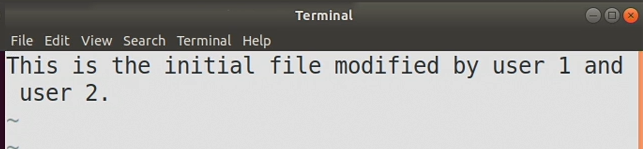


З’явиться конфлікт злиття у результаті проблем додавання локальних змін двома користувачами в один файл.

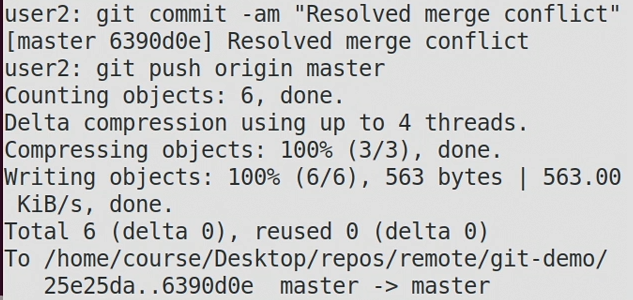
**Вирішення конфлікту**. Відкрити проблемний файл. Git додав метадані з описом конфлікту в цей файл



Виправте файл таким чином, щоб урахувати зміни обох користувачів



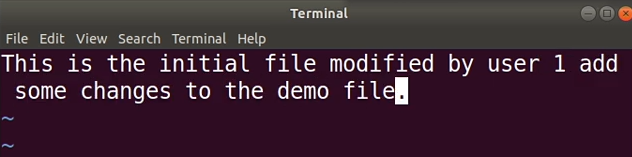
Далі ці зміни фіксуються та пушаться.



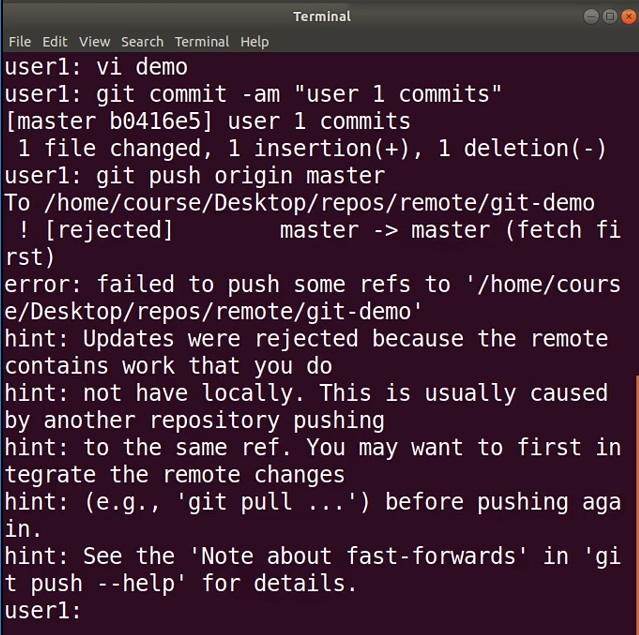
***2.2. Конфлікт: Forced Pushes.*** Користувач 2 створює новий, фіксує та пушить файл у центральний репозиторій:



Користувач 1 у цей час бажає внести зміни в інший, раніше створений файл (тут – demo)



та зафіксувати їх у центральному репозиторії. Git попередить, що в цей час зміни були внесені в центральний репозиторій, а в локальній версії Користувача 1 їх немає.



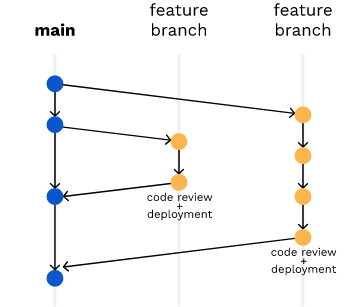
Впертий користувач може захотіти все-одно внести свої зміни:

**git push origin master --force**

Це призведе до втрати змін Користувача 2. *Зробіть скриншот вмісту репозиторія, щоб показати відсутність відповідного файлу*.

1. *Схематично реалізуйте командою з 2-3 осіб стратегію галуження GitHub Flow. Візуалізуйте роботу з репозиторієм за допомогою GUI-клієнта, на зразок SourceTree.*

Стратегія концентрується навколо основної вітки та функціональних віток. Main завжди перебуває в готовому до випуску стані. Розробники створюють нову функціональну гілку для внесення змін. Після огляду цих змін на дефекти та якість коду, вони будуть спрямовані (deploy) в продакшн. Якщо дані зміни створюють проблеми, відбувається відкочування до поточної версії коду (гілки main). Інакше, функціональна вітка буде злита в вітку main.



У порівнянні з Git Flow, GitHub Flow набагато менш важковаговий. Супровід відносно простий, оскільки єдиною довготривалою гілкою є master (main). Проте GitHub Flow успадковує інші недоліки Git Flow. Недисципліновані команди, які тримають функціональні гілки відкритими тижнями, ризикують мати серйозні проблеми при злитті гілок. Загалом, неуспішне злиття може залишити основну гілку в непрацездатному (undeployable) стані, що є неприпустимо при неперервному постачанні.

Отже, який робочий процес GitHub?

* Вміст гілки master завжди працездатний (deployable).
* Починаючи роботу над чимось новим, відгалужуйте від гілки master нову вітку, назва якої буде відповідати її призначенню (наприклад, «new-oauth2-scopes»).
* Зафіксувавши цю гілку локально, відправляйте регулярно свою роботу й у одноіменну гілку на сервері.
* Коли вам знадобиться відгук чи допомога, або коли ви вважатимете гілку готовою до злиття, відправте запит на злиття.
* Після того, як хтось інший переглянув та погодив фічу, ви можете злити вашу гілку в master.
* Як тільки гілка master поповнилась новим кодом, ви можете негайно впровадити його у продакшн, і вам слід це зробити.

Детальніше – [тут](https://habr.com/ru/post/189046/).

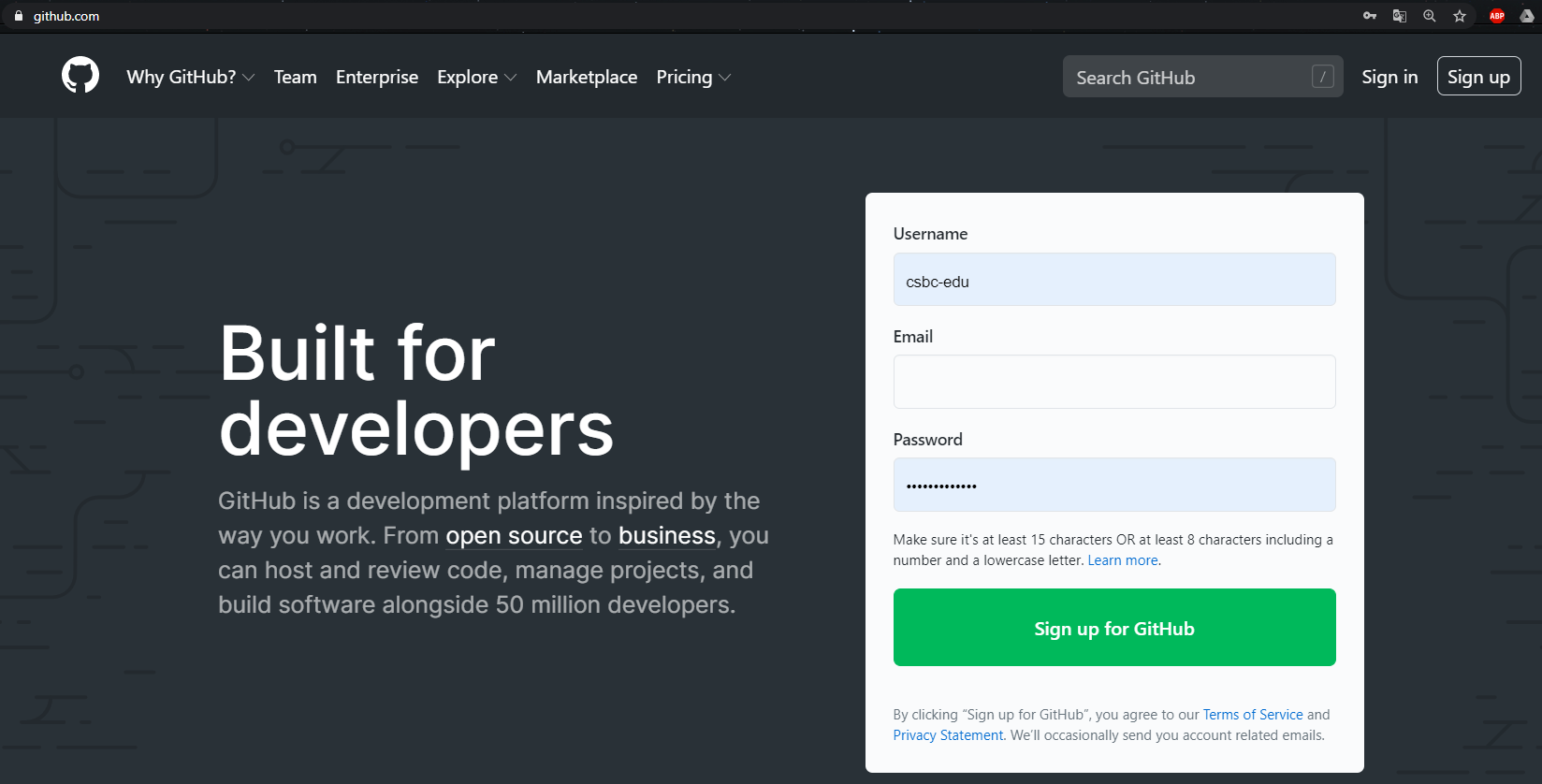
**ІНДЗ**

***Групова розробка реферативного повідомлення***

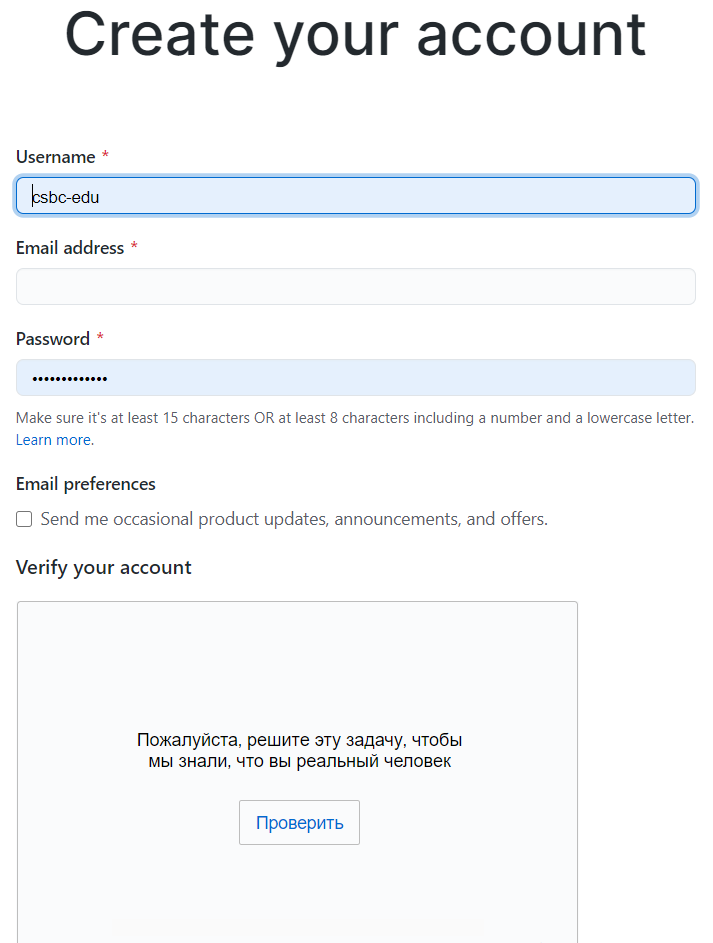
| RJ Agreement - BIGlilFISH Divi Website Design with SEO  **Завдання 1.7.** | *Об'єднайтесь у групи по 2-3 особи та застосуйте отримані навички роботи з Git для створення спільного реферату згідно з обраною тематикою.*   | **№** | **Тема** | **ПІБ студентів** | | --- | --- | --- | | 1. | Стилі кодування та написання якісного коду |  | | 2. | Системи контролю версій |  | | 3. | Програмна платформа Docker |  | | 4. | Scrum-підхід до організації процесу розробки програмного забезпечення |  | | 5. | Методи тестування програмного забезпечення |  | | 6. | Методи оцінювання витрат на розробку програмного забезпечення |  | | 7. | Професії в геймдеві |  | | 8. | Обчислювальна складність алгоритмів |  |   *Реферат повинен включати матеріали принаймні з 3 книг та 8 джерел загалом, мінімум одне джерело – англомовне, російськомовні джерела ДУЖЕ НЕБАЖАНІ. Обсяг реферату повинен бути не меншим за 15 сторінок основного тексту, оформленого за стандартними вимогами: шрифт – Times New Roman, кегль – 14, інтервали – 1.5, відступ для абзацу – 1.25см та ін.*  *Продемонструйте в звіті статистику роботи над рефератом по користувачах, історію коммітів, процес збору джерел (підбір літератури, посилання на сторінки з основним матеріалом та веб-джерела) тощо.* |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

* + - 1. **Базова взаємодія з сервісом GitHub**

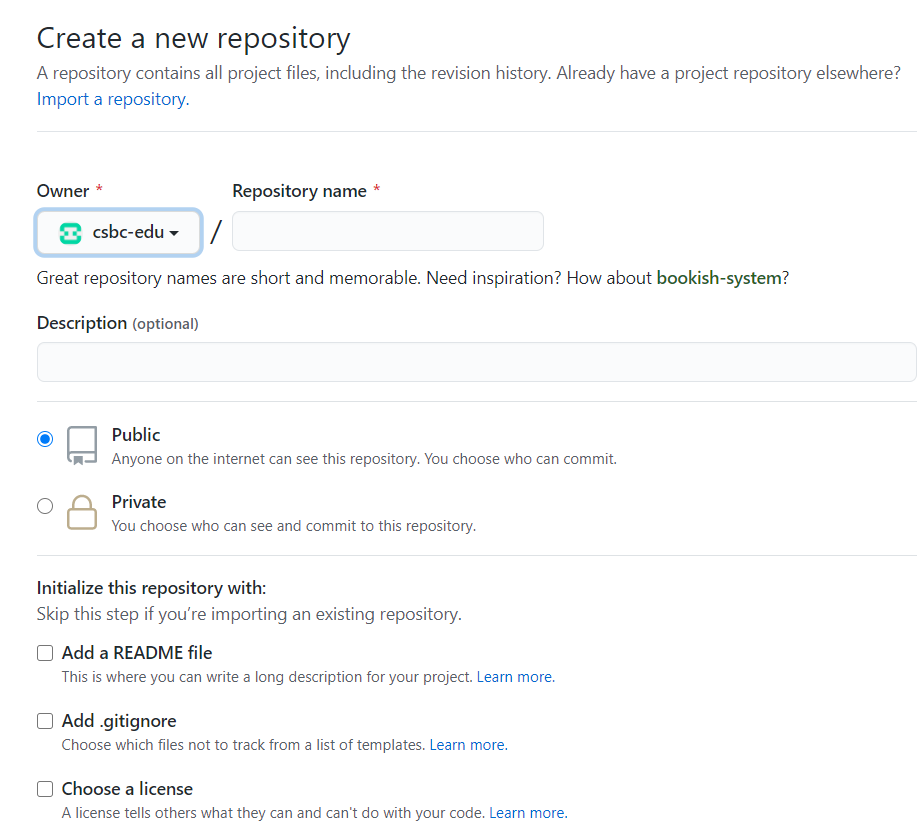
Сервіси хостингу репозиторіїв, на зразок GitHub або BitBucket, дозволяють формувати віддалений репозиторій, доступ до якого може отримувати ціла група розробників. З цією метою слід зареєструвати обліковий запис відповідного хостингу. Цей обліковий запис буде використовуватись протягом усього курсу. У якості прикладу візьмемо GitHub:

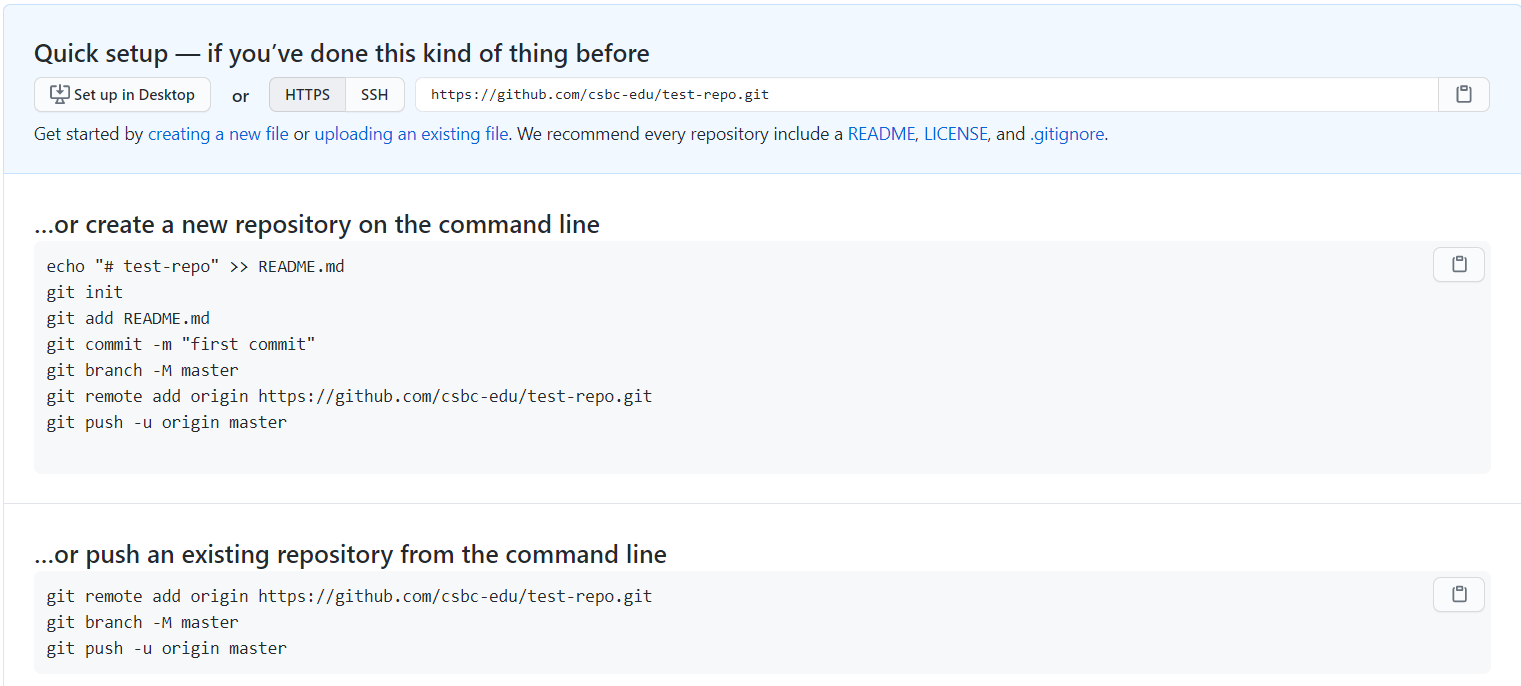


Натисніть кнопку Sign Up для реєстрації облікового запису, якщо досі не маєте такого ж або подібного. Після цього введіть потрібні дані та виконайте реєстрацію на сервісі.



При першому вході відразу доступне створення нового віддаленого репозиторію, куди буде потрібно відправити зафіксовані до цього зміни:





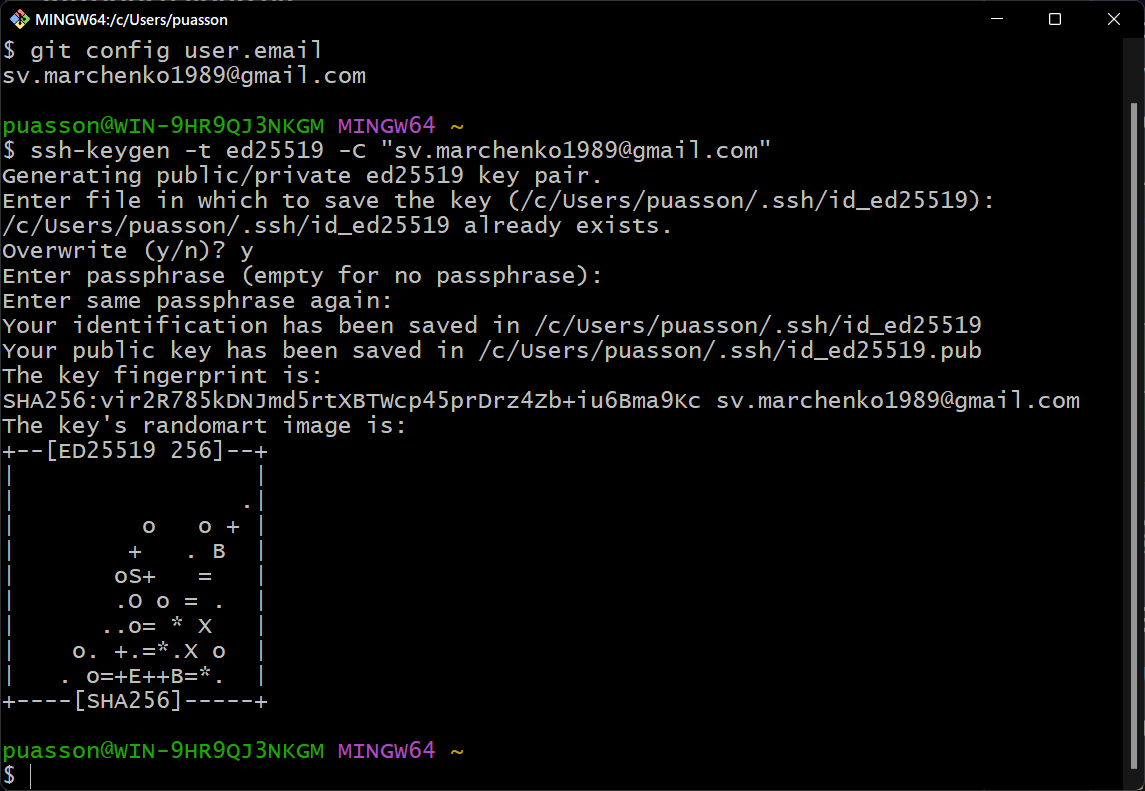
GitHub пропонує набір альтернативних команд для створення нового репозиторію чи відправки своєї локальної версії проекту в новостворений репозиторій.

Наступним, часто необов’язковим, кроком є отримання SSH-ключа для [автентифікації](https://uk.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F_(%D0%B2%D0%B5%D0%B1)) на GitHub. За допомогою протоколу [Secure Shell Protocol](https://en.wikipedia.org/wiki/Secure_Shell) (SSH) ви можете отримувати доступ та записувати дані в репозиторії на GitHub.com. При підключенні по SSH ви підтверджуєте свої права доступу (авторизуєтесь, без постійних введень логіну/паролю) за допомогою файлу приватного ключа, розташованого на вашій машині. Для цього потрібно перевірити наявність такого ключа та за потреби згенерувати його.

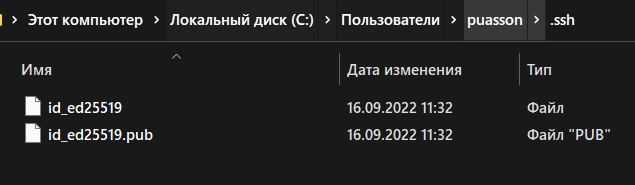
Для перевірки наявності SSH-ключів у терміналі записують наступну команду:

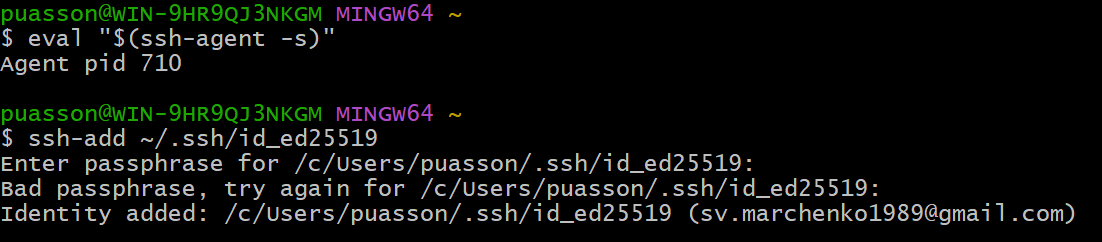


Бачимо, що папки з такою назвою немає, тому потрібно згенерувати новий SSH-ключ. Для цього слід користуватись [покроковою інструкцією з GitHub](https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent) для вашої операційної системи. Зокрема, для Windows матимемо наступні кроки:

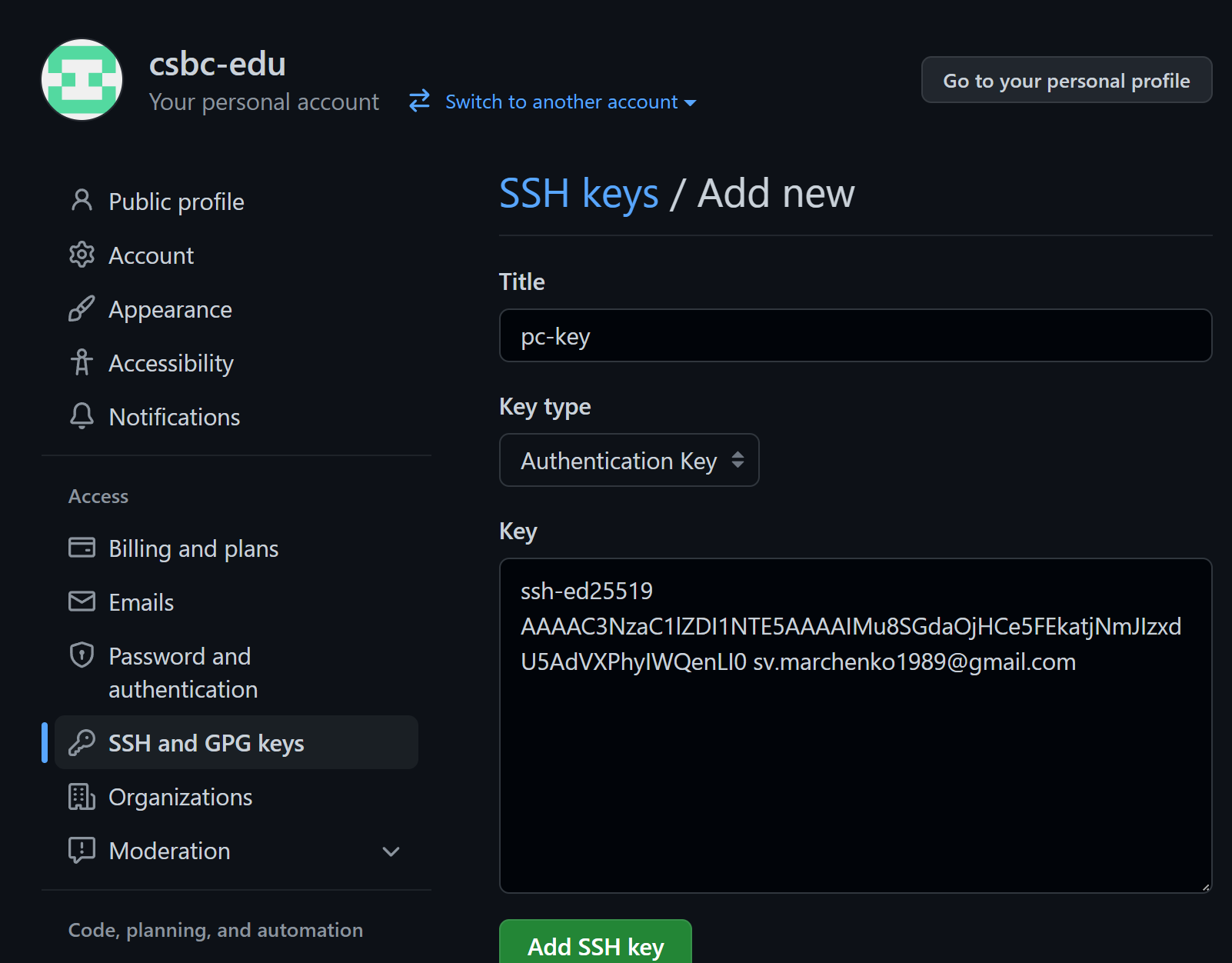


Потрібно буде двічі ввести пароль (passphrase), після чого отримуємо 2 файли: id\_ed25519 id\_ed25519.pub



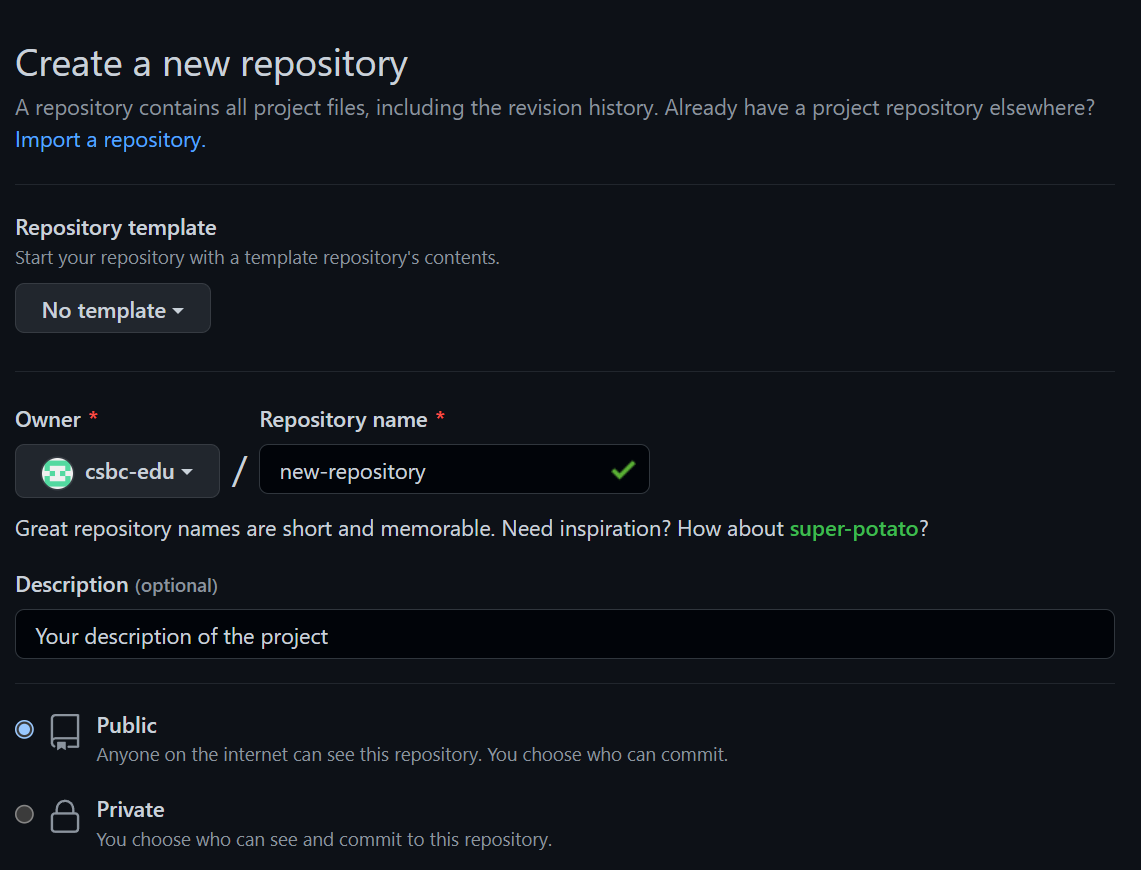


$ clip < ~/.ssh/id\_ed25519.pub

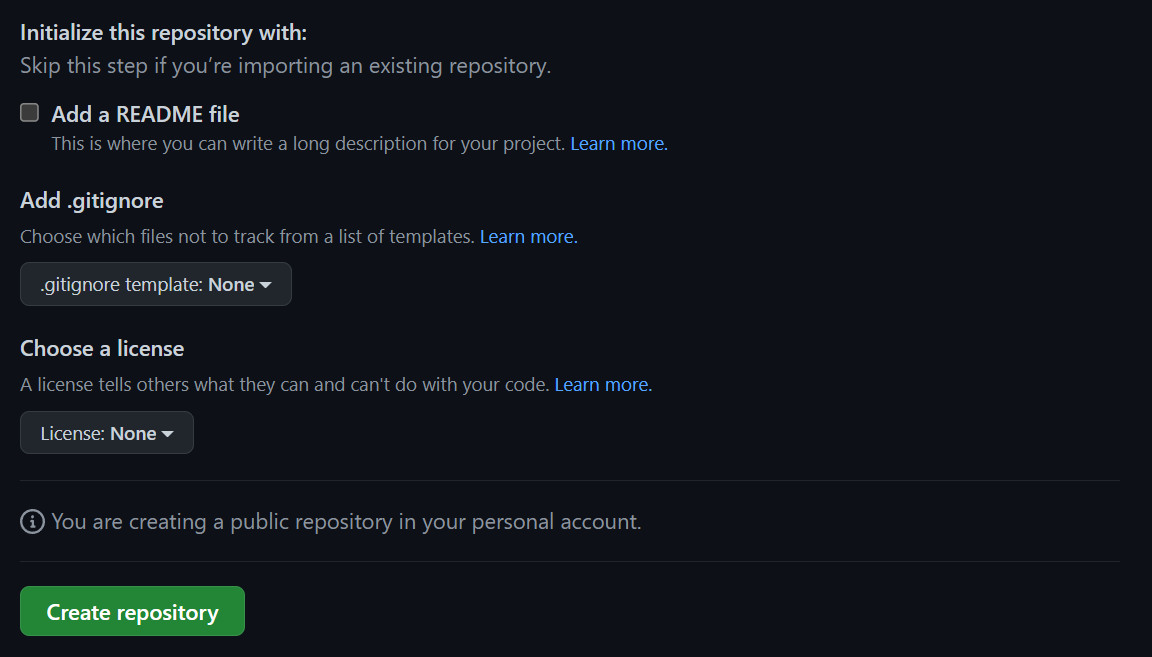


При використанні GitHub Desktop потреби в SSH-ключах немає.

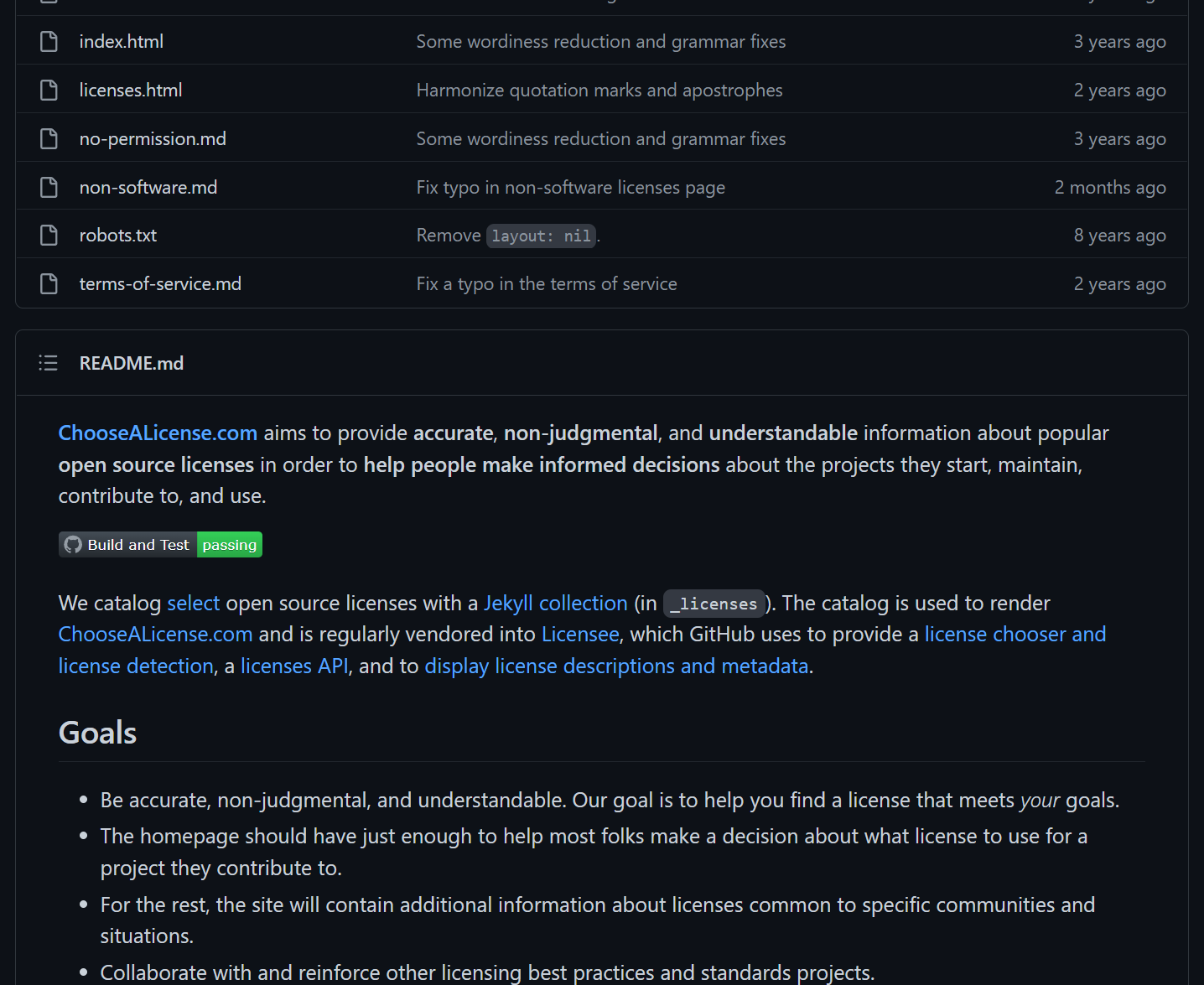
**Сценарій 1. Додавання локального репозиторію на GitHub.** Створити порожній репозиторій на GitHub та підтягнути код з локального репозиторію. Натискаємо зелену кнопку New та переходимо до меню створення репозиторію



Репозиторій повинен мати унікальну для облікового запису назву та бути відкритим (public) або закритим (private). Додатково можна ввести текст з описом проєкту.



Також при створенні можна додати файл README, який міститиме всю описову інформацію, яка відображатиметься разом з репозиторієм. Також зверніть увагу, що якщо оберете пункт Add a README file, гілка master автоматично буде перейменована в main. Відображення цього файлу буде подібним до такого:

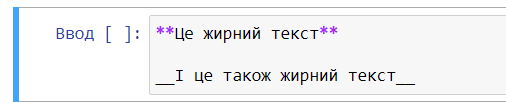
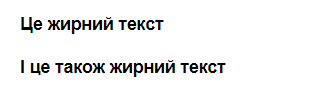


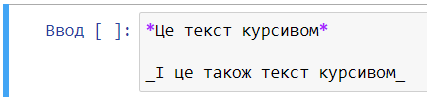
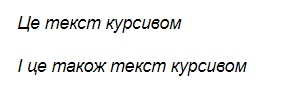
Для оформлення README та Python-блокнотів використовується мова розмітки [Markdown](https://medium.com/analytics-vidhya/the-ultimate-markdown-guide-for-jupyter-notebook-d5e5abf728fd), структурно подібна до HTML / XHTML, проте далеко не така потужна та вичерпна. Мова Markdown призначена для форматування тексту з метою його читання. Приклади наведено в Jupyter Notebook.

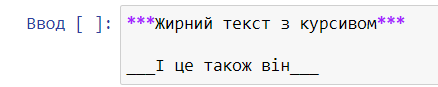
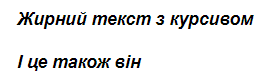
Аналогами тегів <h1> - <h6> з HTML є дописування символу # на початку рядка:

Жирний, курсивний та комбінований текст представляються в Markdown так:

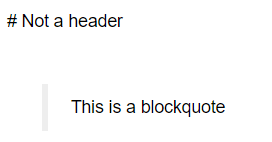
Перекреслений текст:

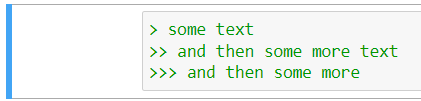
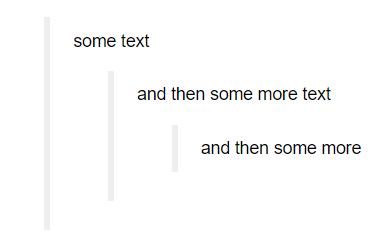
Текст з символом # записується з екрануванням символом \

Для переходу на новий рядок достатньо натиснення клавіші Enter, спеціальні теги не застосовуються. Цитата, взята з деяких джерел, позначається тегом >

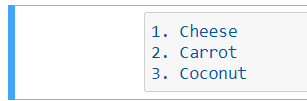
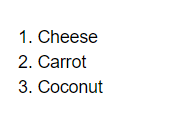
Вкладені цитати оформлюються додатковими тегами >:

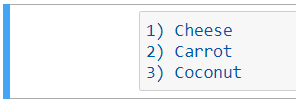
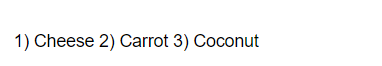
 

Для представлення лінії на весь рядок досить записати --- для товстої риски та \_\_\_ чи \*\*\* для тонкої лінії:

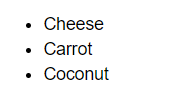
 

Досить зручно створювати нумерований список, проте він обмежується номерами з крапкою.

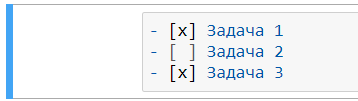
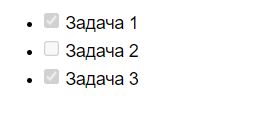
 

Невпорядковані списки представлені за допомогою маркера –

Для додавання чеклісту

Для додавання зображень до блокноту підтримується перетягування файлу зображення чи його вставка з буферу обміну в режимі Markdown:



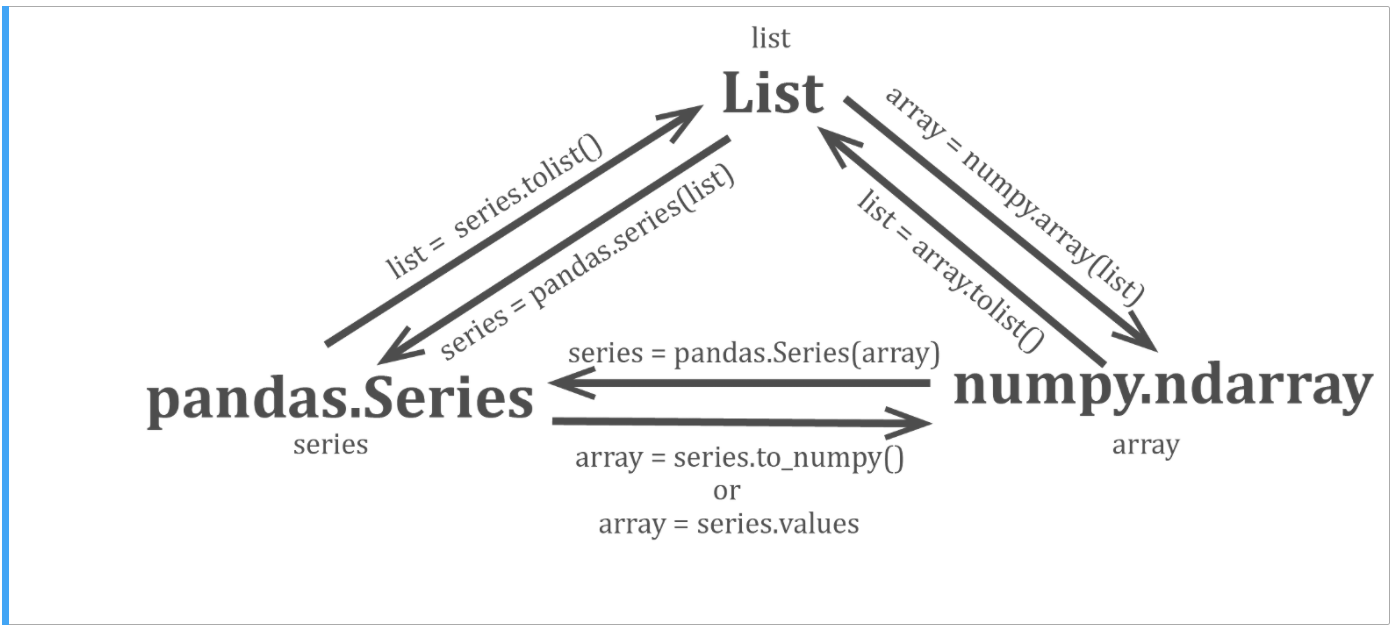


Схема запису команди наступна:

![][певний ІД](офлайн-шлях або URL “Заголовок”)



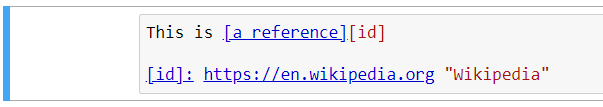


Автоматичні гіперпосилання формуються відразу в тексті при їх вставці:

Стандартні посилання оформлюються з відповідними текстом так:

Якірні (anchor) посилання дозволяють переміщуватись в межах різних секцій документу. Для створення якоря можна просто додати код на зразок

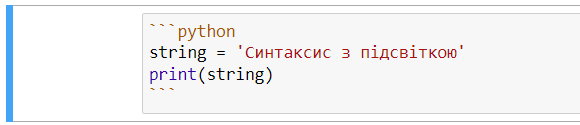
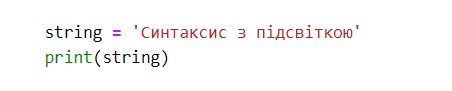
[Текст посилання](#My-Great-Heading)

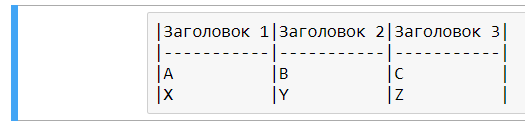
При натисненні на лінк курсор буде переміщено на вказану секцію. Ідентифікатор формується як # + назва секції з дефісами замість пробілів.

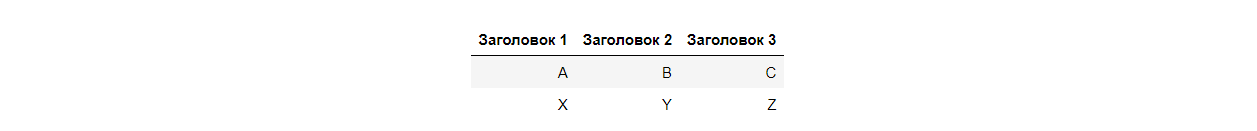


Додавання в секцію коду з підсвіткою вимагає огортання цього коду трьома символами ` (Ё в англійській розкладці) та зазначення мови програмування. Аналогічно можна включити підсвітку JavaScript чи інших мов:

Вбудовування таблиць за допомогою Markdown-синтаксису визначає застосування символів | та -.





Для ручного вирівнювання вмісту комірок раніше застосовувались такі позначення:

* :-: для центрування;
* -: для вирівнювання справа;
* :- для вирівнювання зліва,

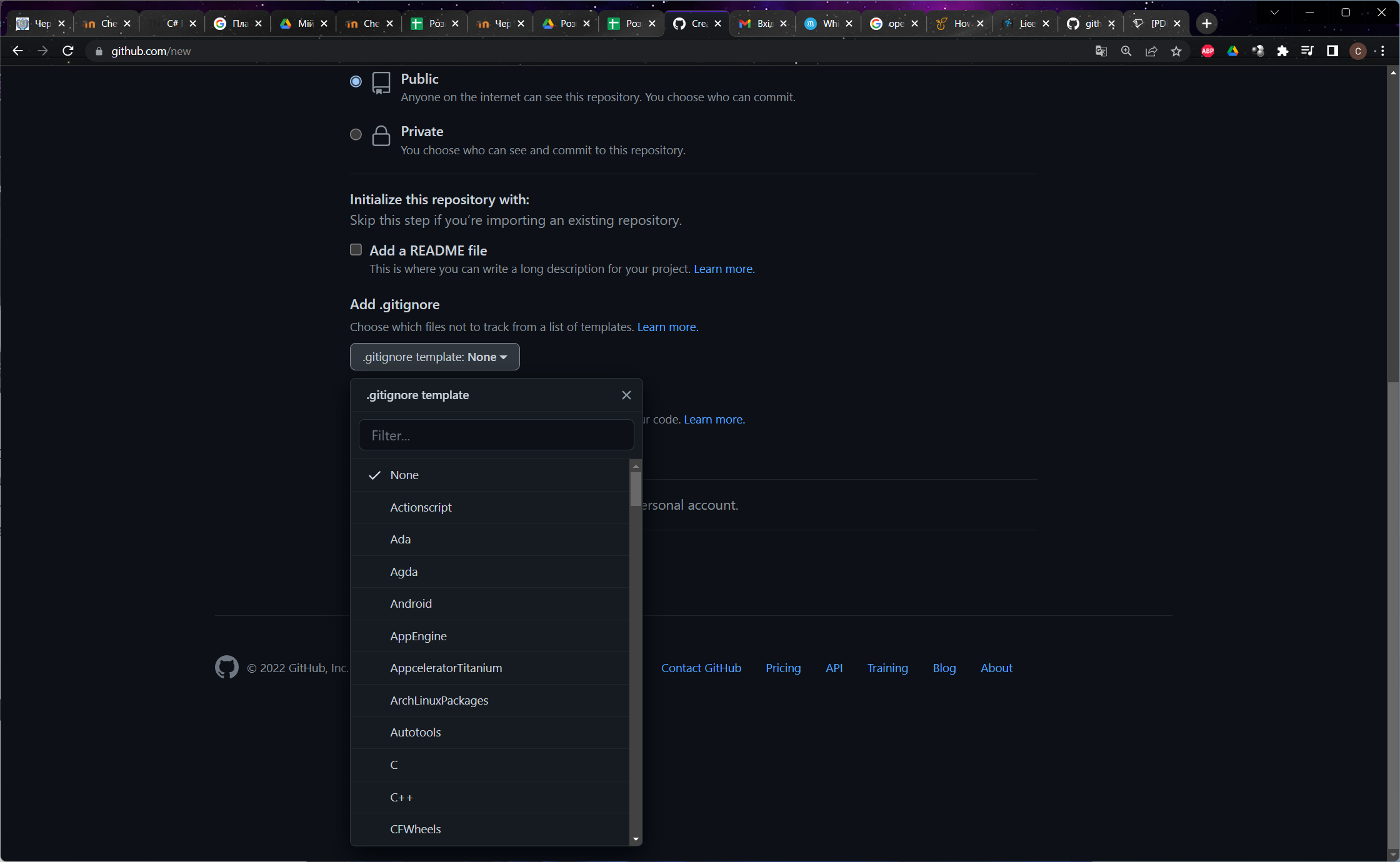
проте зараз у блокнотах *вони не працюють*.

Мова Markdown допускає вбудовування деяких HTML-тегів. Зокрема, розфарбування тексту кольором відповідає [колірним назвам HTML](https://www.w3schools.com/colors/colors_names.asp). Також доступна можливість задавати колір шістнадцятковим значенням. Існує декілька способів задати тексту колір, проте вони можуть не завжди працювати:

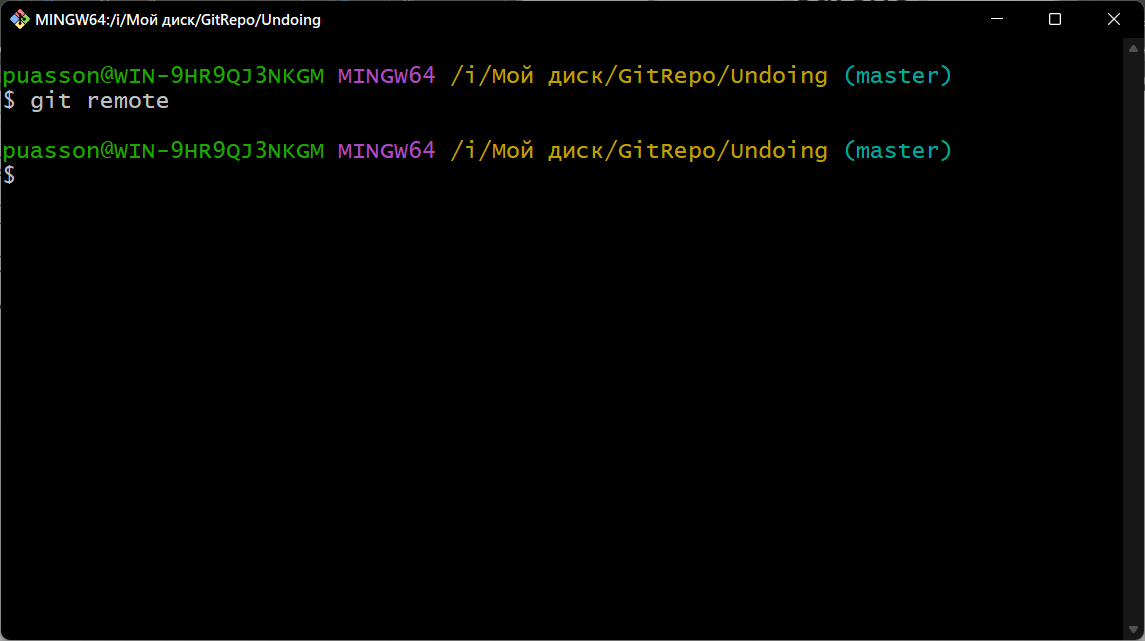
Додаткові інструменти для зміни шрифтів та впровадження інших HTML-тегів також можете розглянути [тут](https://towardsdatascience.com/7-essential-tips-for-writing-with-jupyter-notebook-60972a1a8901#4f95).

Ще одна опція при створенні репозиторію – налаштування файлу .gitignore. У ньому міститься перелік файлів чи розширень, які не будуть відкриватись та додаватись до GitHub-репозиторію. У першу чергу, це файли з закритими даними та різні допоміжні файли від середовища розробки. GitHub пропонує набір готових шаблонів для ігнорування службових файлів різних мов програмування та IDE:

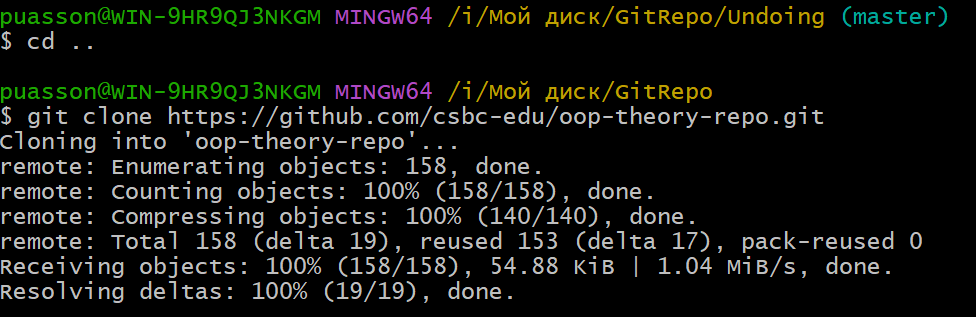


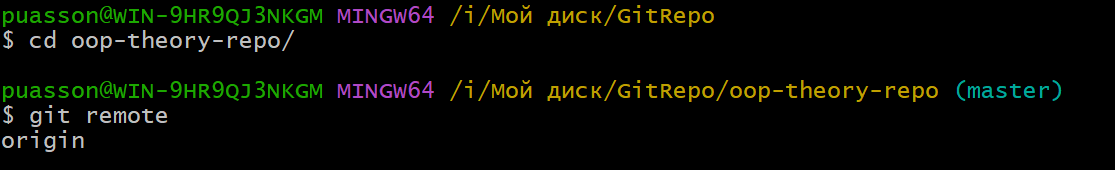
Останній пункт для налаштування при створенні репозиторію – вибір ліцензії розповсюдження вашого коду. Короткий [огляд типів ліцензій](https://choosealicense.com/licenses/) та [порівняльна таблиця](https://choosealicense.com/appendix/) для ПЗ з відкритим кодом.

До того, як відправити (push) код на GitHub, потрібно сповістити Git про наш віддалений (remote, у вигляді URL-адреси) репозиторій на GitHub. Візьмемо останній репозиторій з попередньої теми та застосуємо для нього команду git remote, який виведе перелік віддалених репозиторіїв, до яких ми маємо доступ. Очевидно, що зараз такий перелік порожній:

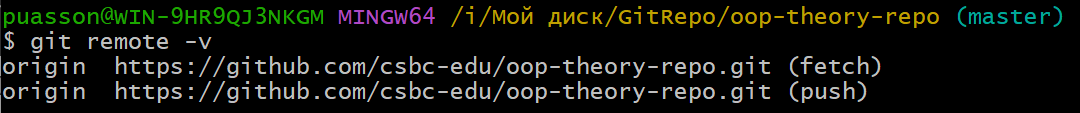


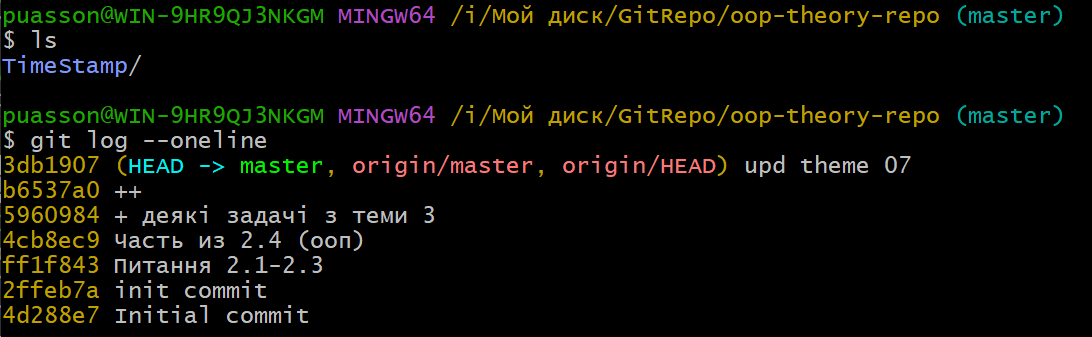
Проте якщо клоную певний репозиторій з GitHub, то для нього матимемо іншу ситуацію:





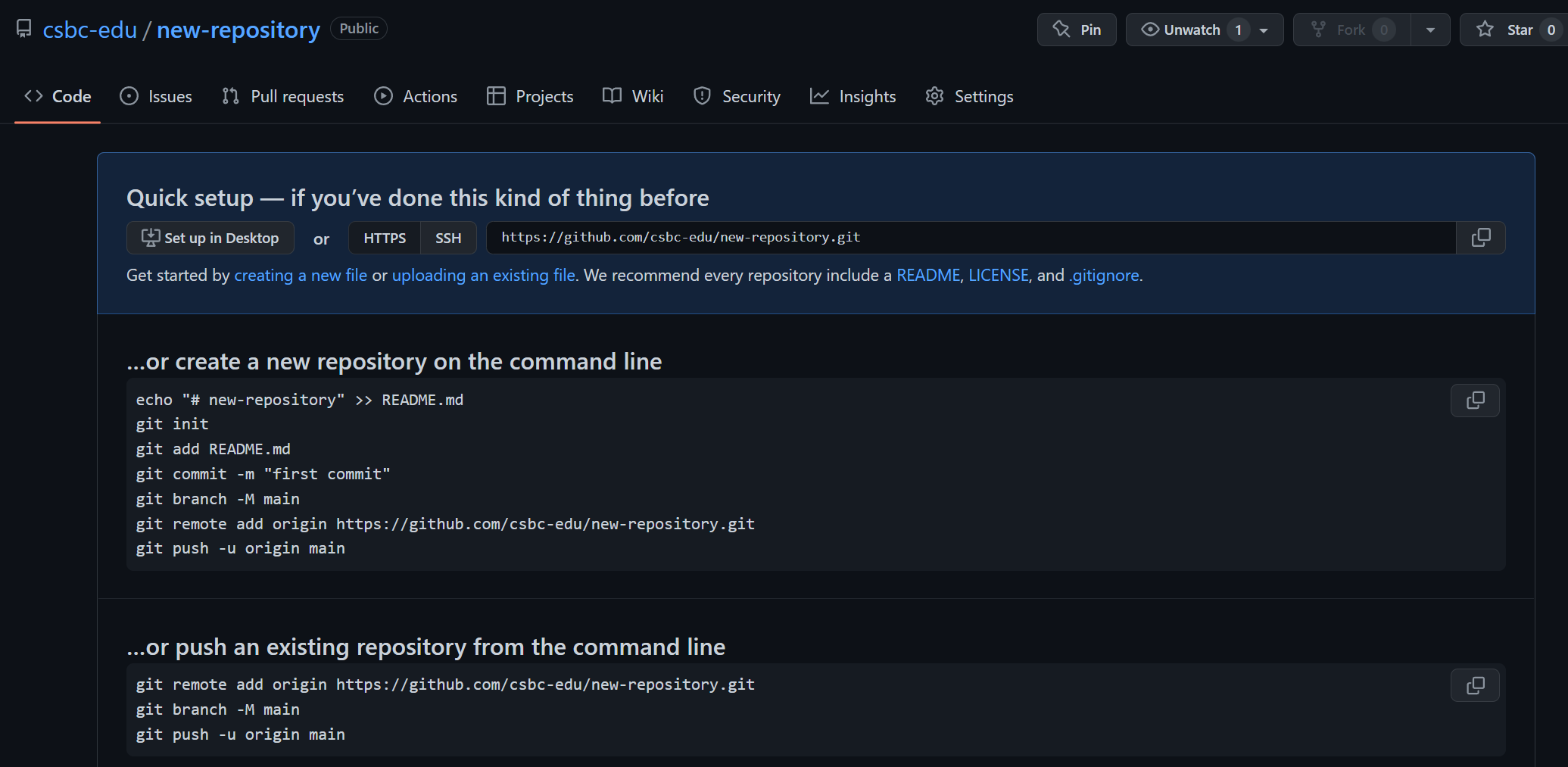
Для більш детального опису можна застосувати ключ -v (від verbose):

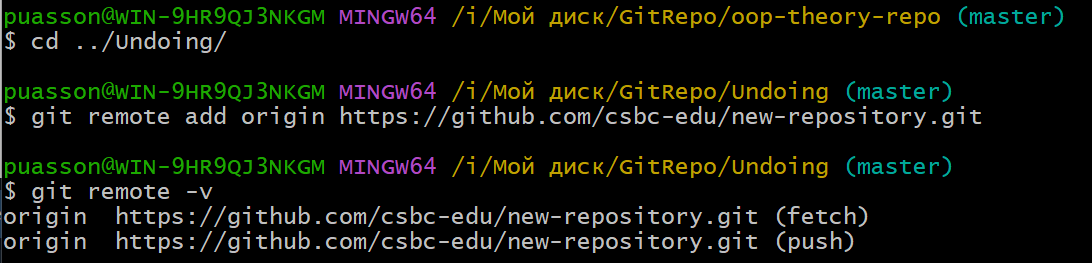




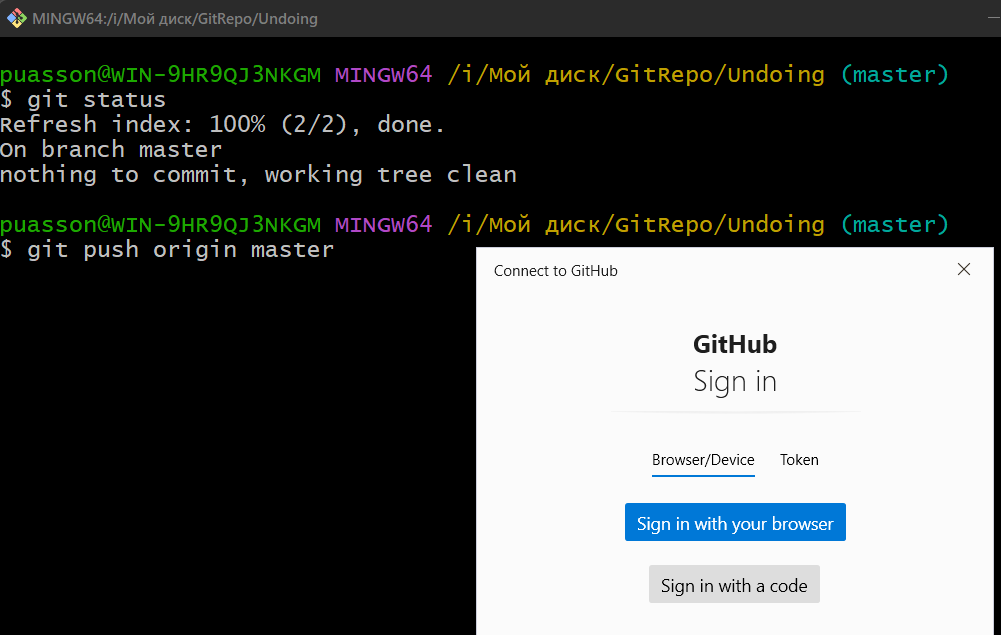
Для локального репозиторію ми можемо створити посилання на віддалений репозиторій за допомогою команди

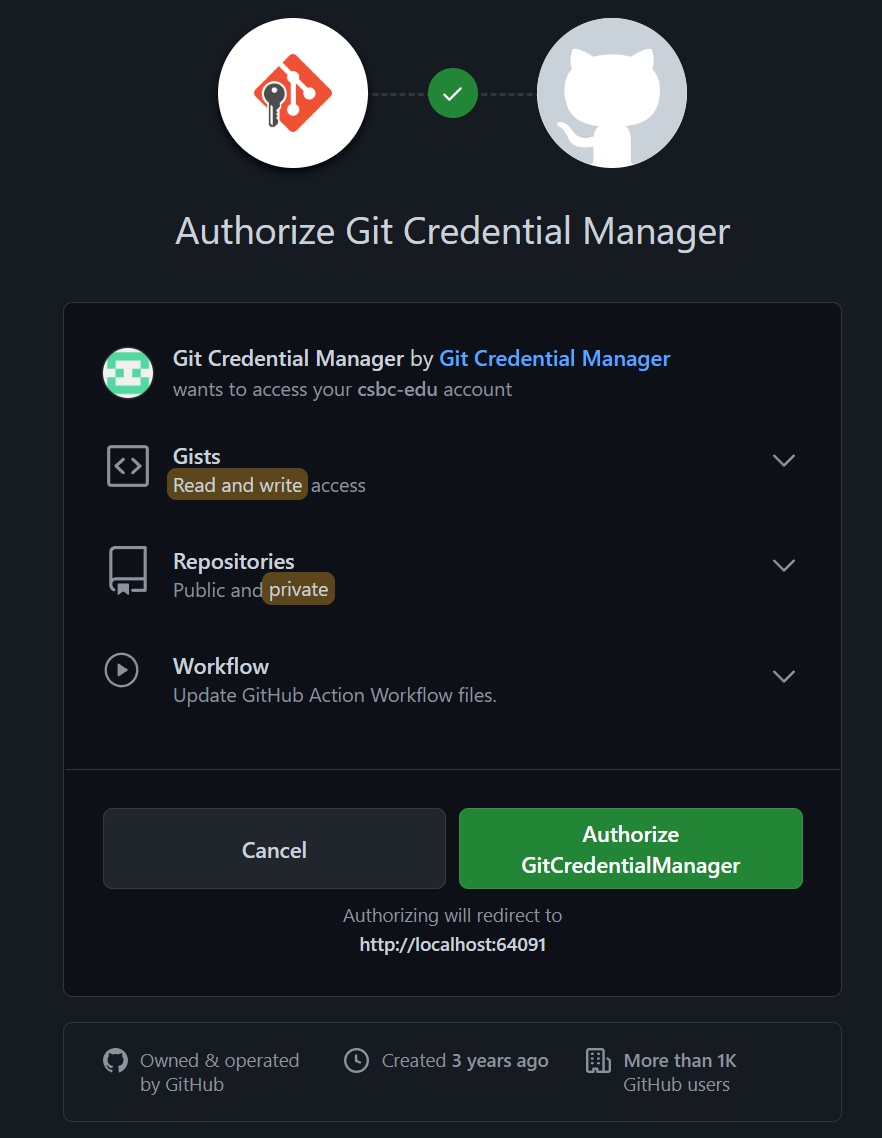
git remote add деяка\_назва url

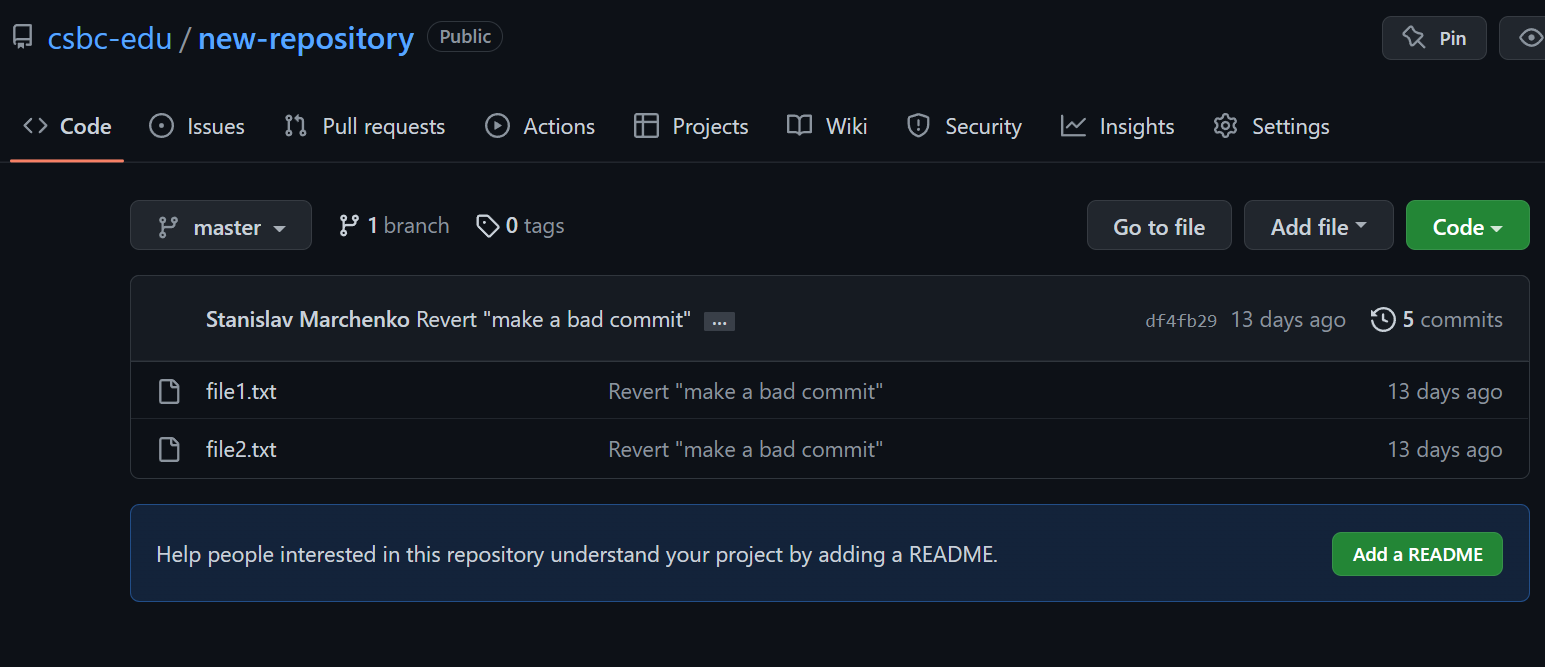




Визначивши «ціль», куди буде завантажуватись ваш локальний код, використовуємо команду git push, щоб розпочати відправку. Як і clone, дана команда не є специфічною для GitHub, а використовується і з іншими хостингами, на зразок BitBucket або Gitlab.



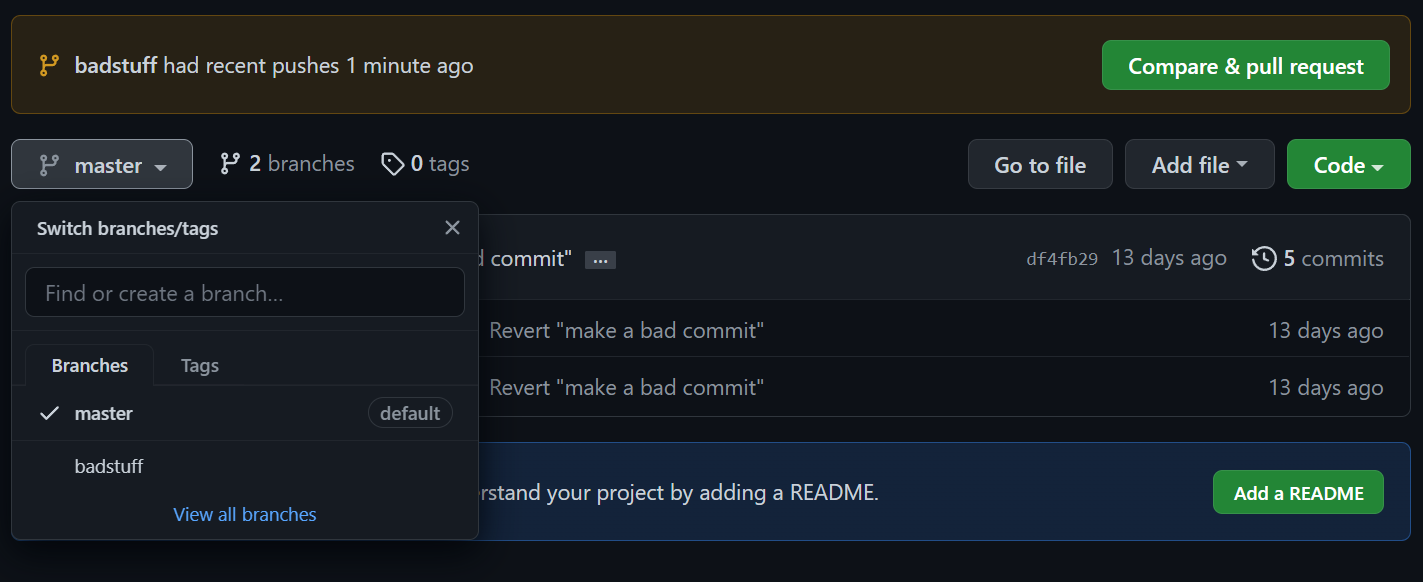




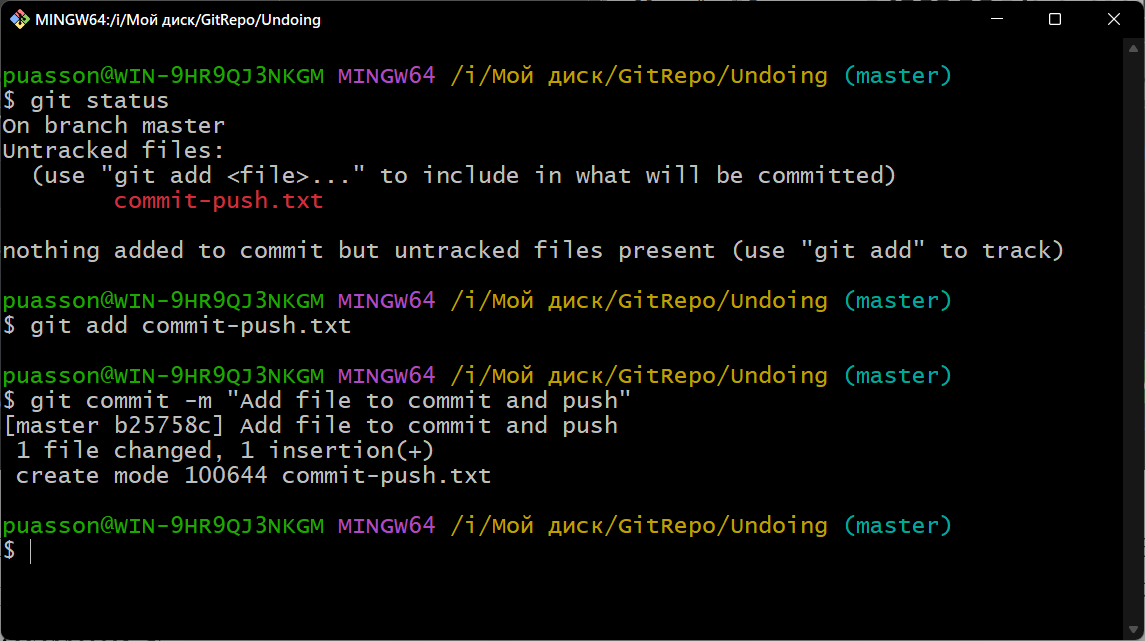
Дана команда відправила на GitHub тільки гілку master. Якщо вам потрібно відправити іншу гілку, повторіть команду з її назвою замість master. Для відправки всіх гілок відразу застосовують

git push --all origin

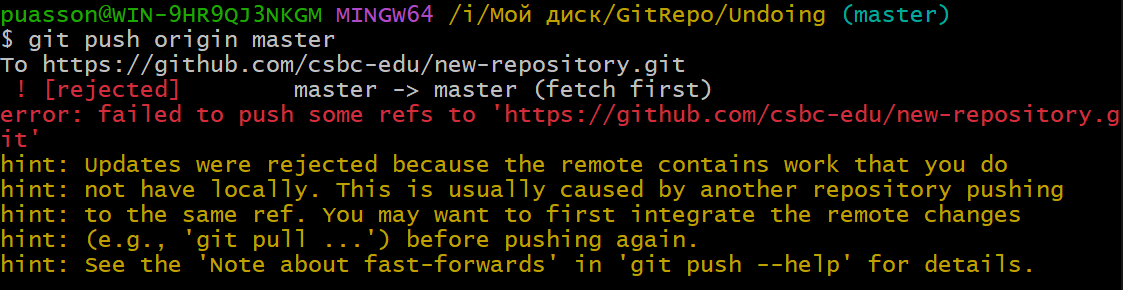


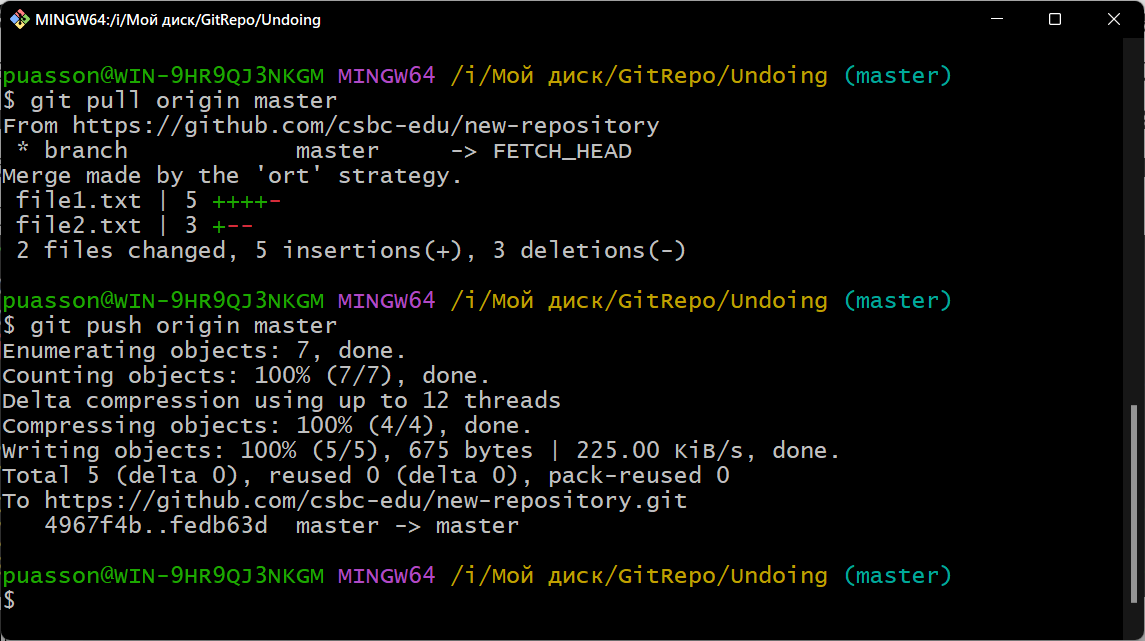


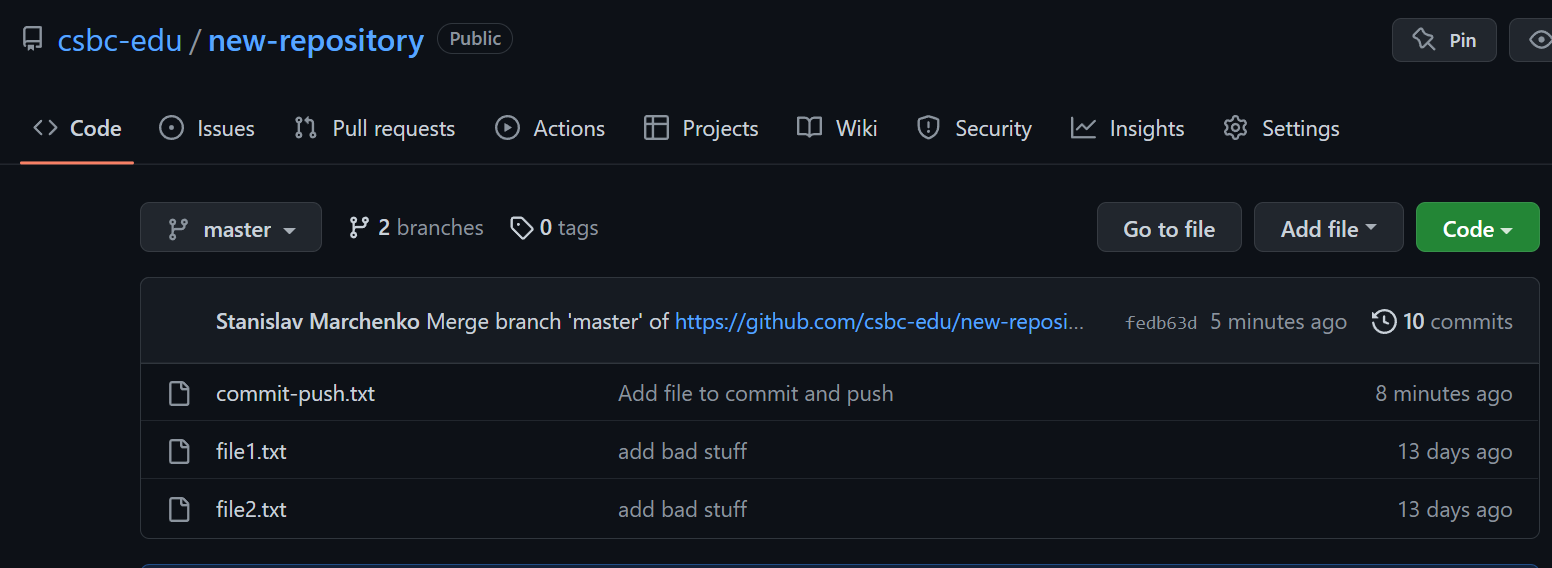
При появі нового коміта в локальному репозиторії автоматично в віддалений репозиторій він не надішлеться. Додамо новий текстовий файл та зафіксуємо зміни:



Якщо в репозиторії вже з’явились нові зміни, зокрема, в даному випадку, виконано pull request для об’єднання гілок, слід отримати останню версію коду за допомогою команди git pull origin master. Продемонструємо наявність проблеми: Git не приймає нові зміни, доки у вас локальний репозиторій не буде мати найсвіжіший код.







**Сценарій 2. Робота з нуля.**

* Створити новий репозиторій на GitHub;
* Клонувати його собі на машину;
* Здійснити деяку роботу локально;
* Запушити зміни на GitHub.

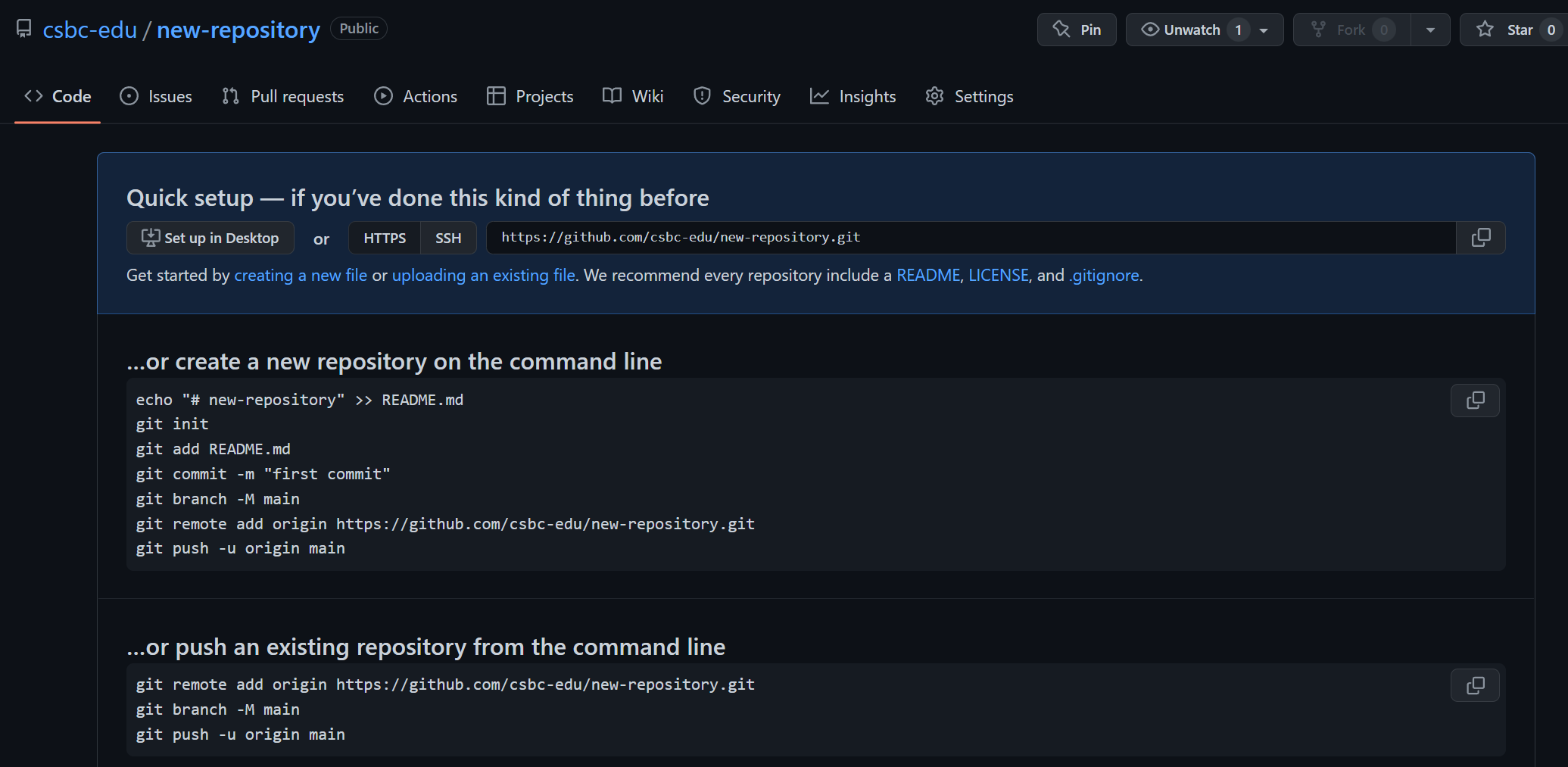
Зауважте, що команда push працює й тоді, коли вам потрібно відправити зміни у віддалену гілку з іншою назвою. Якщо до цього ми переважно мали справу з випадком master -> master, то загалом, віддалена гілка може називатись не так, як локальна. Для цього слід в команді писати так:

git push деяка\_назва локальна\_гілка:віддалена\_гілка

Наприклад,

git push origin pancake:waffle

Також часто зустрічається ключ -u при застосуванні команди push. Зокрема серед команд при створенні нового репозиторію на GitHub.



Опція -u відповідає слову upstream та ключу --set-upstream, що в цілому означає вимогу *запам’ятати* зв’язок між локальною та віддаленою гілкою. Тобто команда

git push -u origin dogs

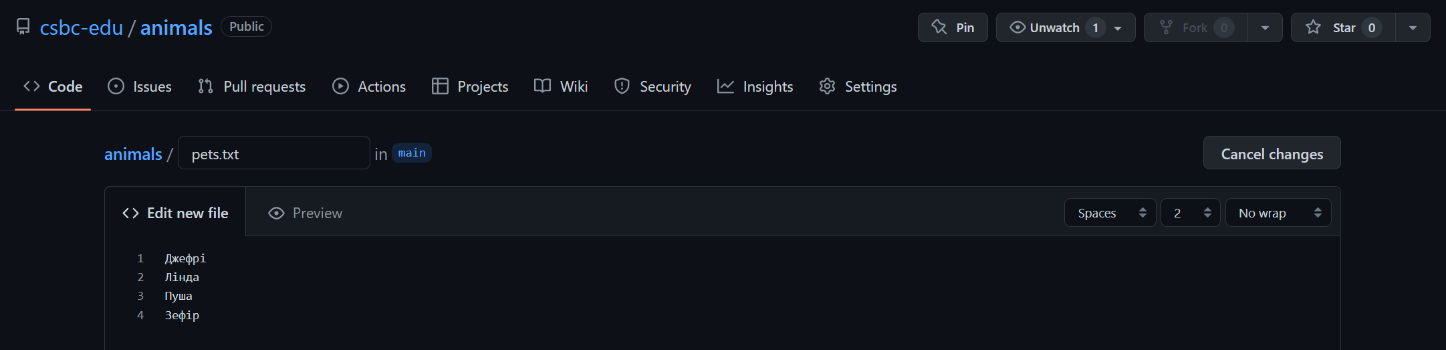
визначає, що буде встановлено та записано зв’язок між локальною гілкою dogs та віддаленою гілкою origin/dogs. Тепер у подальшому ми можемо просто писати

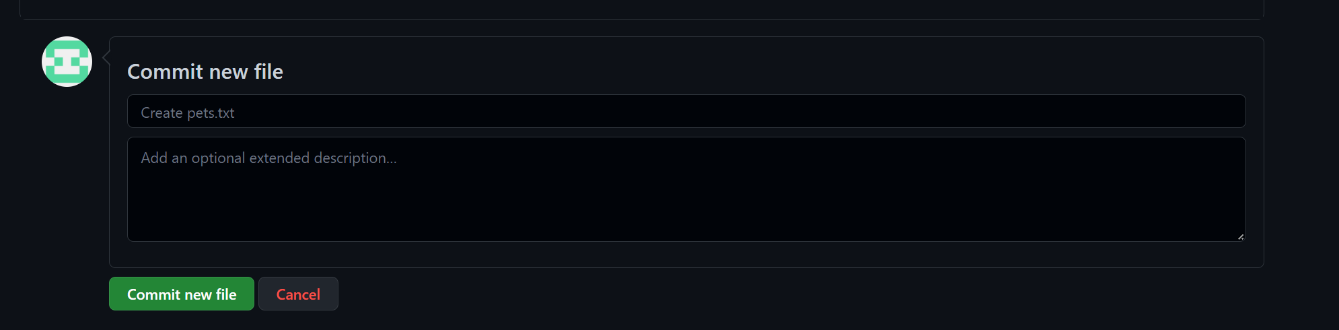
git push

і Git автоматично відправлятиме зміни в origin/dogs. Цей ключ працює і для попереднього випадку з різними назвами локальної та віддаленої гілок.

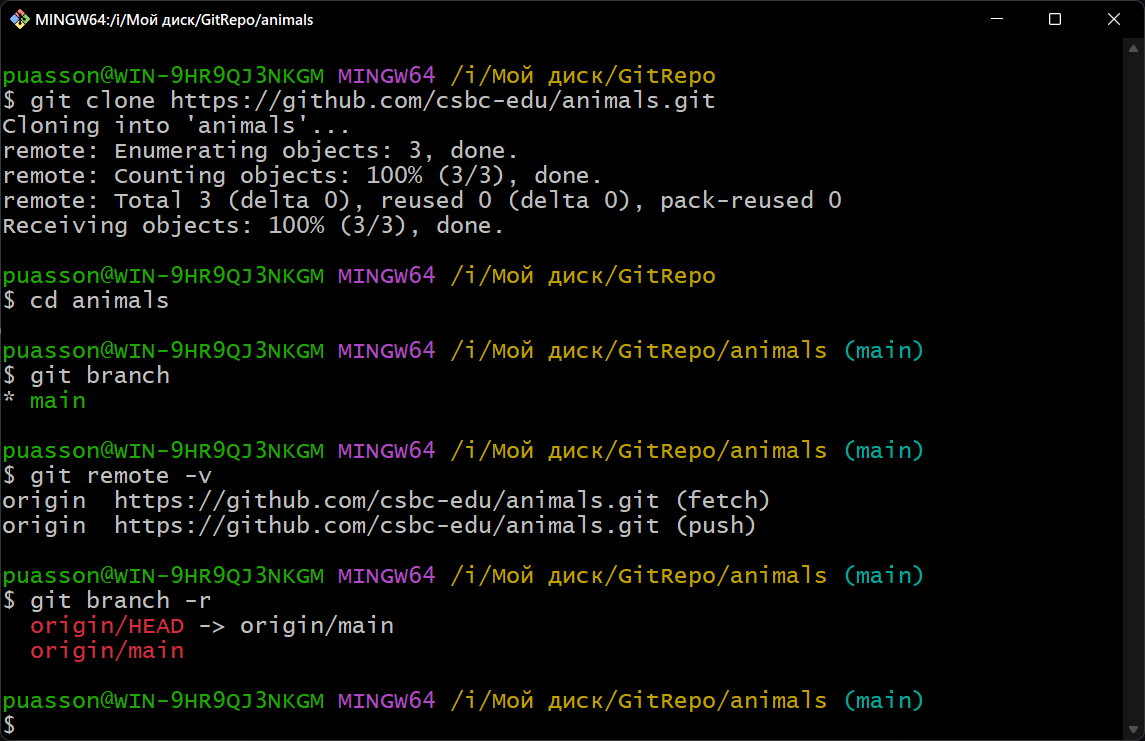
**Отримання коду з GitHub: команди git fetch та git pull**. Створимо репозиторій та файл у ньому засобами GitHub. Натиснемо на «creating a new file»:





******

Тепер створимо на комп’ютері порожню папку та клонуємо в неї наш репозиторій:



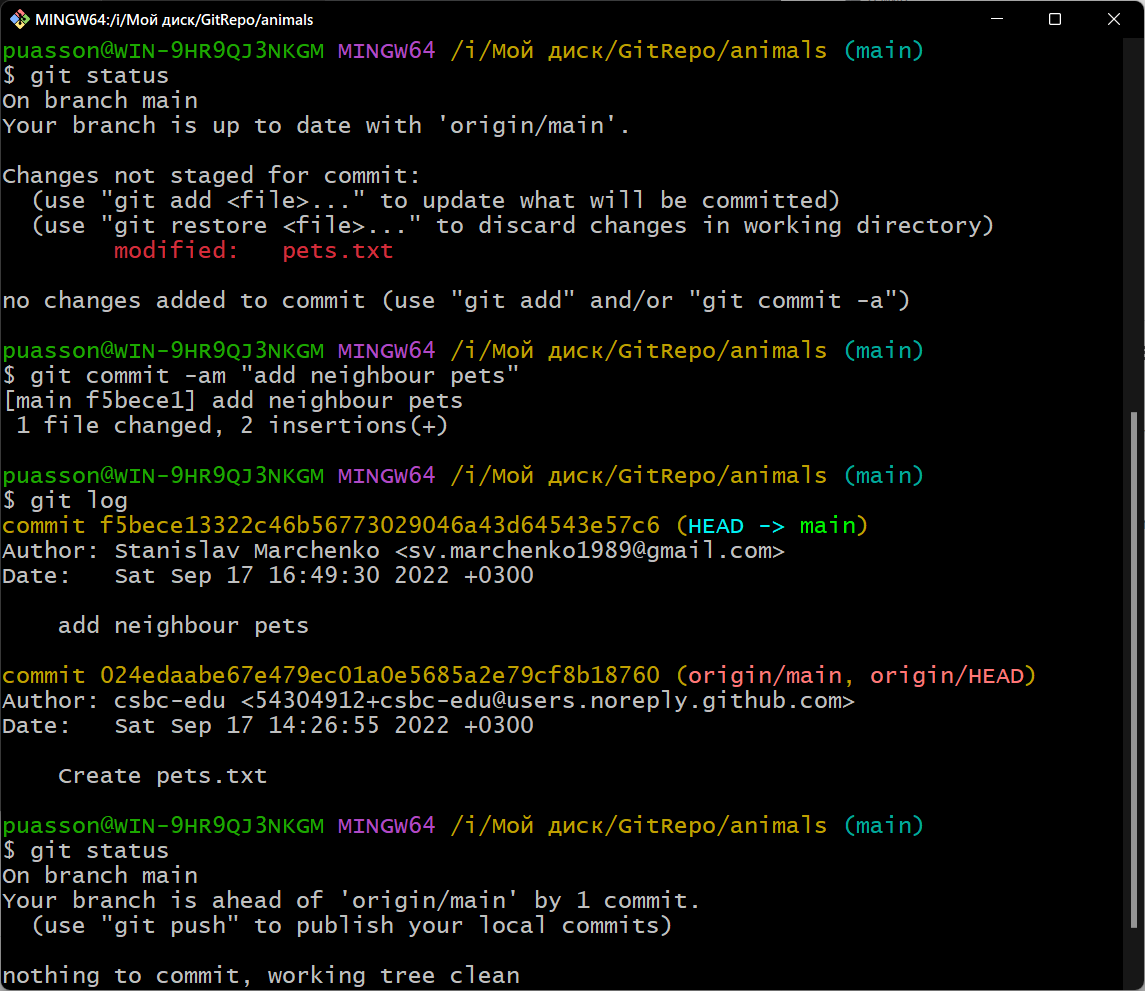
Для перегляду посилань на віддалені гілки (remote tracking branch references) використовують команду

git branch -r

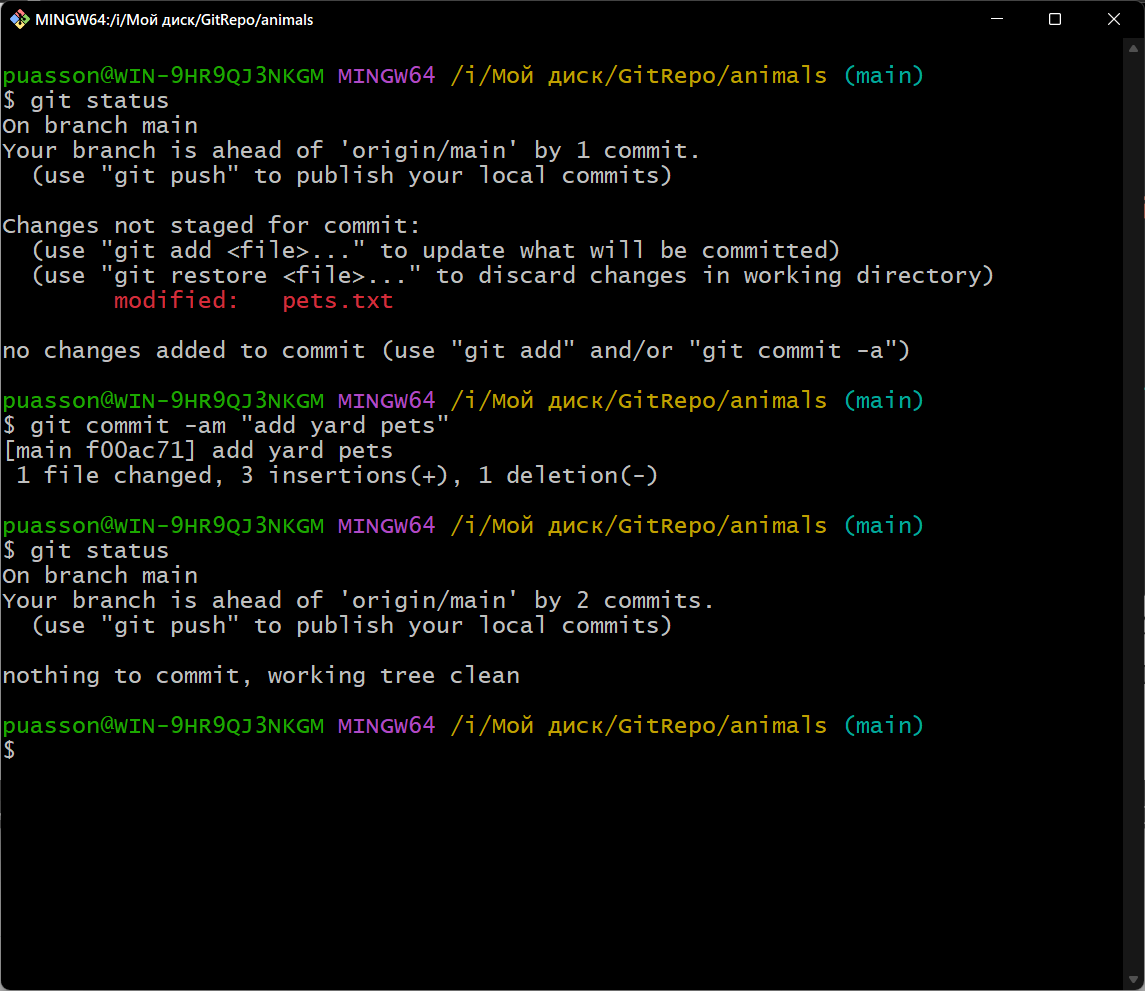
На поточний момент маємо таку ситуацію:

Після виконання роботи локально матимемо стан

Спочатку зробимо одну зміну – додамо імена домашніх тварин сусідів:



Бачимо, що остання команда status вказує на те, що гілки main та origin/main тепер вказують на різні коміти! Зробимо ще один коміт з додаванням імен дворових тварин:

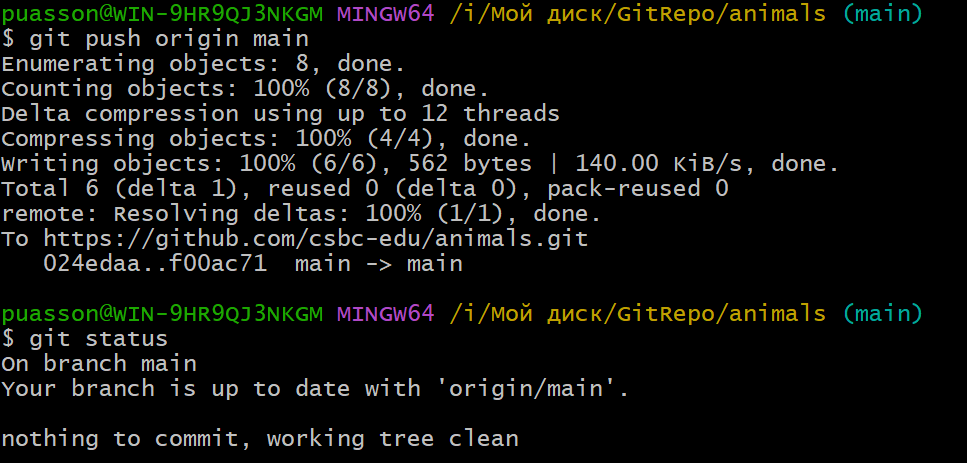


Переглянути стан віддаленого репозиторію без змін у локальному завжди можна за допомогою

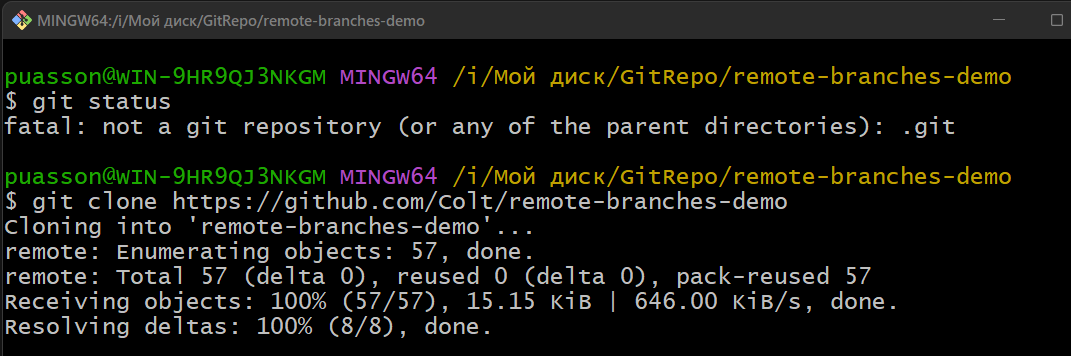
git checkout origin/main

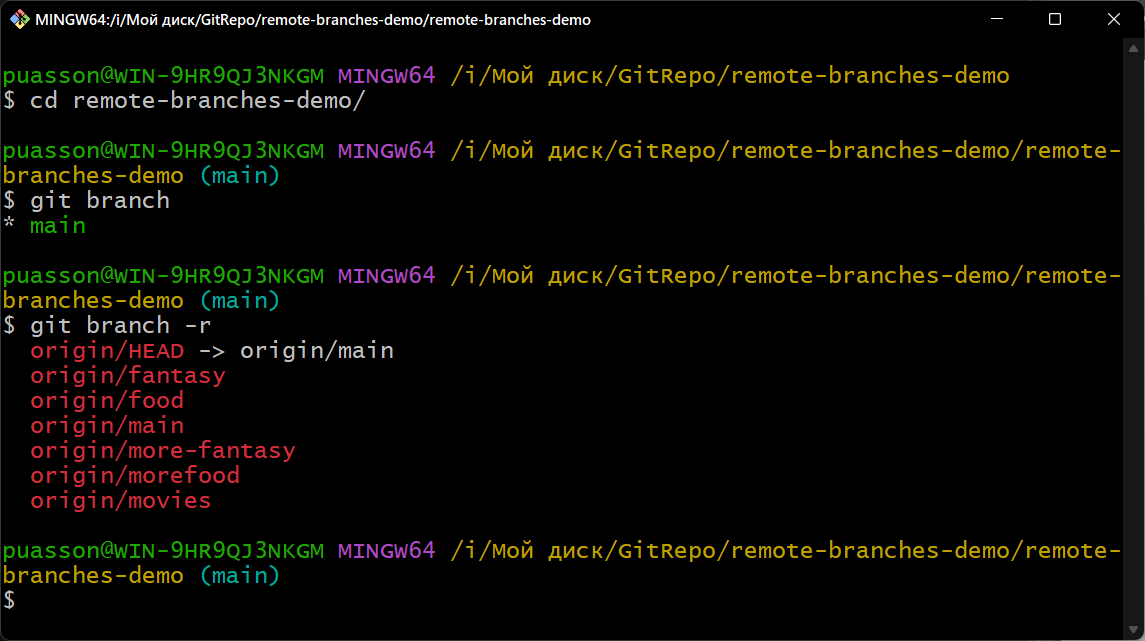
а відправити фактичні зміни з локального у віддалений репозиторій так:

git push origin main



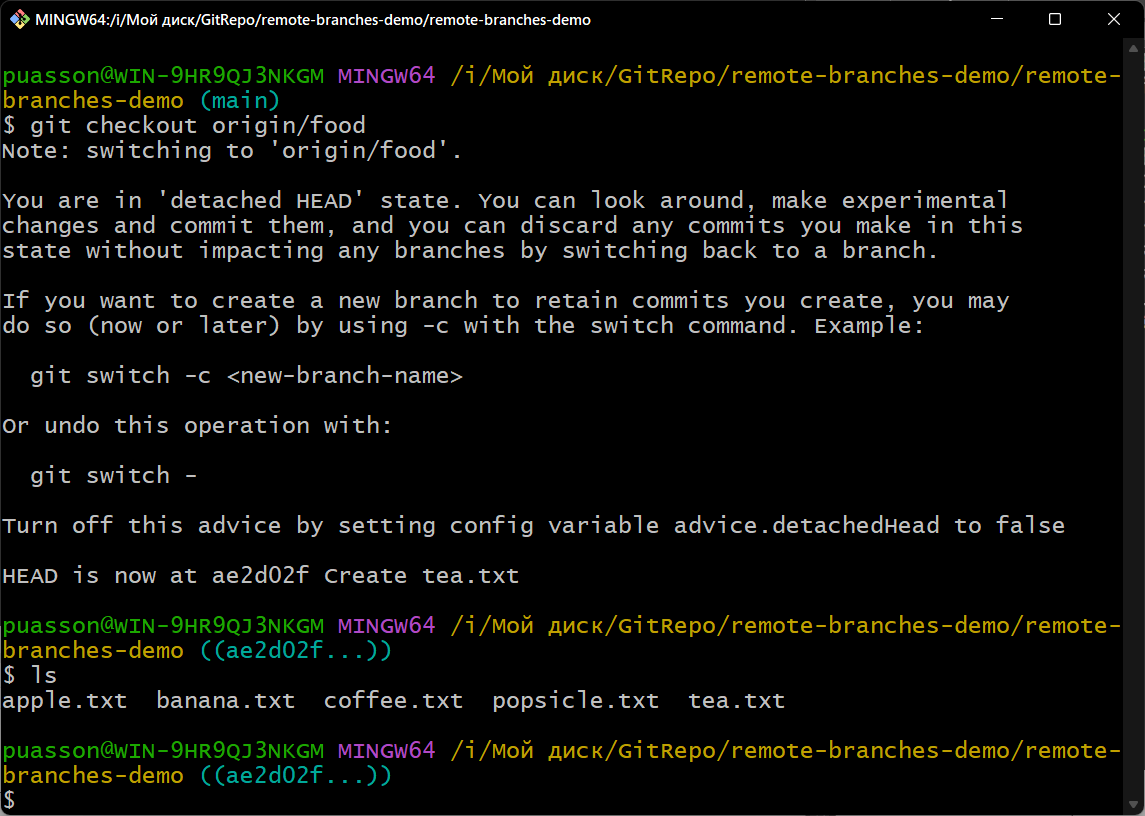
Далі детальніше розглянемо ситуацію з клонуванням проєкту з багатьма гілками. Для прикладу візьмемо https://github.com/Colt/remote-branches-demo:



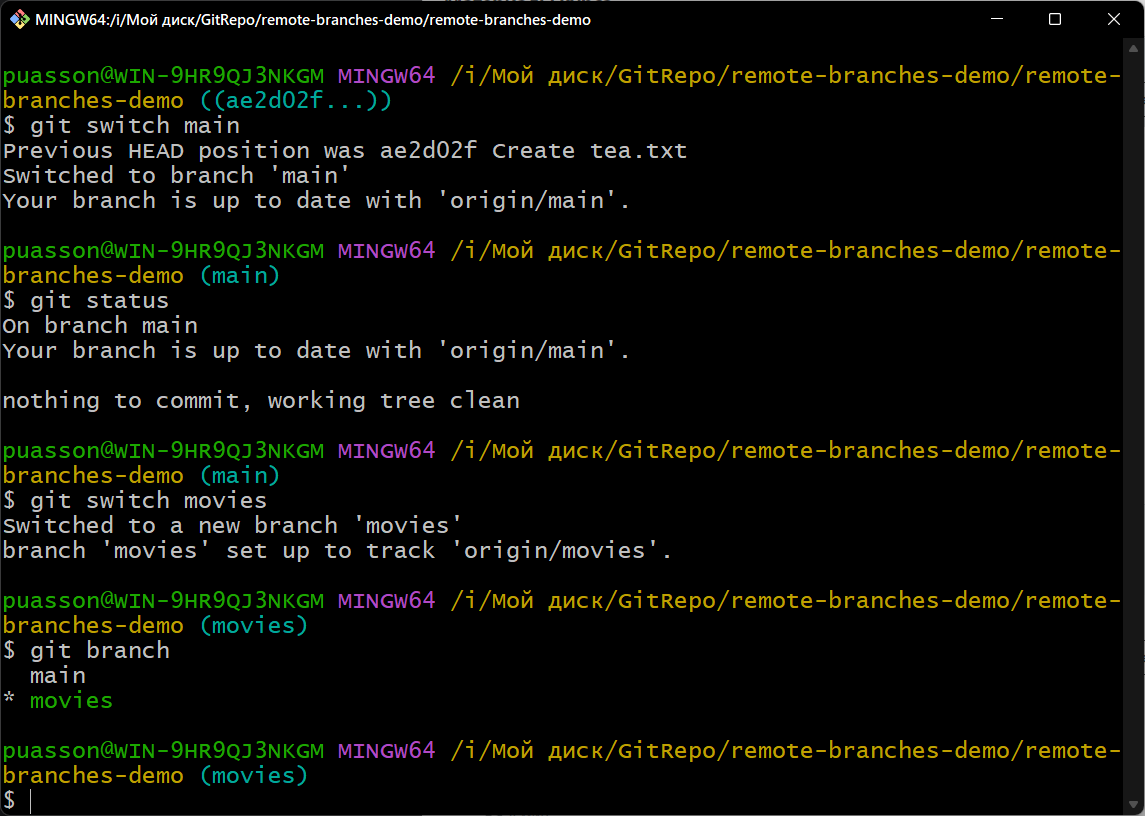


Цікавим є те, що локальний репозиторій бачить лише гілку main, проте йому відомо й про інші гілки віддаленого репозиторію. Зв’язок між основними гілками локального та віддаленого репозиторію встановлюється автоматично.

Якщо просто переключитись на гілку віддаленого репозиторію, то перейдемо в стан detached HEAD. Тим не менш, файли репозиторію відповідатимуть віддаленій гілці.



Проте основне бажання тут – мати локальні гілки, що відповідають віддаленим. Завдяки команді git switch ми можемо звернутись до віддаленої гілки, згенерувати локальну версію з неї та здійснити переключення:



До появи git switch з цією ж метою використовувалась команда

git checkout --track origin/movies

Якщо в гілці будуть відбуватись та фіксуватись зміни, відправити їх у віддалений репозиторій зможемо командою

git push origin movies

Тепер уявімо ситуацію, що поки ви робили локальні зміни в своїй копії репозиторію, інші розробники встигли додати нових комітів у віддалений репозиторій. Вище подібна ситуація вже розглядалась: оновлення від інших розробників треба отримати, при цьому зберігши й свої напрацювання. Для цього можуть застосовуватись команди git fetch та git pull. Принципову відмінність між ними можете помітити на наступному рисунку:

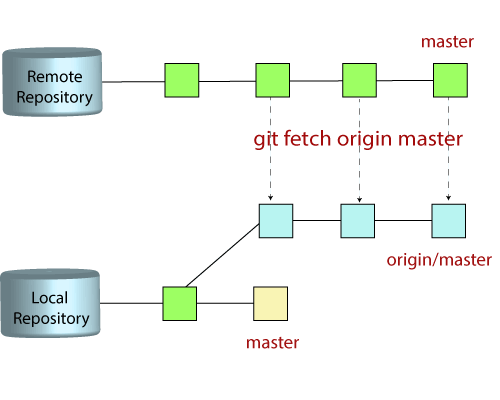


Таким чином, операція отримання (fetching) завантажує зміни з віддаленого репозиторію, проте вони не інтегруються автоматично в ваші робочі файли. Вона просто дозволяє переглянути, над чим працювали інші, не вносячи зміни та не здійснюючи злиття в ваш локальний репозиторій. Тобто команда працює в стилі «покажи свіжі зміни, проте не псуй мій репозиторій»:

git fetch назва\_віддаленого\_репозиторію

Типово пишуть git fetch (якщо віддалений репозиторій ідентифіковано як origin). Також можемо отримати лише конкретну гілку віддаленого репозиторію, вказавши її через пробіл

git fetch origin main



До локального репозиторію впроваджується гілка origin/main (для віддаленого репозиторію origin та його гілки main), що дозволяє не включати завантажені зміни в вашу основну гілку. Зокрема, така команда може бути корисна, коли ви залишились без Інтернету, проте встигли завантажити останні оновлення.

Проте майте на увазі, що ваш локальний репозиторій не опитує весь час віддалений щодо змін. Тобто для оновлення доведеться знову виконувати команду

git fetch origin

а для перегляду, наприклад, оновлень гілки origin/movies -

git checkout origin/movies

При переході на гілку після git fetch вже фактично отримаємо всі оновлення та зможемо щось зробити на їх основі. Для тренування спробуйте зробити коміт через веб інтерфейс GitHub або за допомогою іншої людини через механізм колаборації. Тоді git status допоможе відобразити стан репозиторію до та після git fetch для віддаленої гілки.

У той же час команду витягування (git pull) можна розглядати як суму:

git pull = git fetch + git merge

Тобто ми оновлюємо відстежуючу віддалену гілку (remote tracking branch) з віддаленого репозиторію та отримуємо свіжу версію поточної гілки, включивши в неї зміни, що надійшли. Загальний синтаксис команди:

git pull назва\_віддаленого\_репозиторію назва\_гілки

Зокрема для origin/master матимемо

git pull origin master

Також доступна коротка версія команди:

git pull

що передбачає назву віддаленого репозиторію – origin, а назву гілки – поточну для локального репозиторію.

**Редактори коду**

IDLE, PyCharm, Colab, VS Code