

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"
Інститут комп'ютерних наук та інформаційних технологій
Кафедра систем автоматизованого проектування



Звіт
До лабораторної роботи №1
«Системи контролю версій»
За дисципліною «Управління ІТ-проектами»

Виконав:
Ст. гр. КН-408
Москвитин Артем-Денис

Прийняв:
Шпакович І.М.

Львів-2022

Мета роботи:

Ознайомитись з системами контролю версій. Зрозуміти принципи синхронізації робочих груп у гнучких командах. Набути навичок використання технічних засобів та протоколів для роботи з системами контролю версій.

Теоретичний аналіз опрацьованого матеріалу:

Система контролю версій (від англ. Version Control System, VCS або Revision Control System) – програмне забезпечення для полегшення роботи з інформацією, що змінюється. Система контролю версій дозволяє зберігати декілька версій одного й того ж документа, при необхідності звертатись до більш ранніх версій, визначати, хто й коли зробив ті чи інші зміни, та багато іншого. Такі системи найбільш широко використовуються при розробці програмного забезпечення для зберігання вихідних кодів розроблюваної програми. Однак вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю електронних документів, що неперервно змінюються. Зокрема, системи контролю версій застосовуються в САПР, зазвичай в складі систем управління даними про виріб (PDM). Контроль версій використовується в інструментах конфігураційного управління (Software Configuration Management Tools).

Першою дією, яку повинен виконати розробник, є витяг робочої копії проекту або тієї його частини, з якою доведеться працювати. Ця дія виконується за допомогою стандартної команди вилучення версії (checkout або clone) або спеціальною командою, яка фактично виконує ті ж саме дії. Розробник задає версію, яка повинна бути скопійована, за замовчуванням зазвичай копіюється остання (або вибрана адміністратором в якості основної) версія. За командою вилучення встановлюється з'єднання з сервером, і проект (або його частина – один з каталогів з підкаталогами) у вигляді дерева каталогів і файлів копіюється на комп'ютер розробника. Звичайною практикою є дублювання робочої копії: крім основного каталогу з проектом на локальний диск (або в окремий, спеціально вибраний каталог, або в системні підкаталоги

основного дерева проекту) додатково записується ще одна його копія. Працюючи з проектом, розробник змінює тільки файли основною

робочою копією. Друга локальна копія зберігається як еталон, дозволяючи в будь-який момент без звернення до сервера визначити, які зміни внесені в конкретний файл або проект в цілому і від якої версії була «відгілкована» (branched) робоча копія; як правило, будь-яка спроба ручної зміни цієї копії призводить до помилок у роботі програмного забезпечення VCS.

Індивідуальне завдання:

Створити локальний репозиторій через консоль Git Bash, та репозиторій у GitHub.

Хід роботи:

Створюємо новий репозиторій у GitHub. Вводимо назву репозиторію «IT-Project». Та натискаємо кнопку «Create repository».

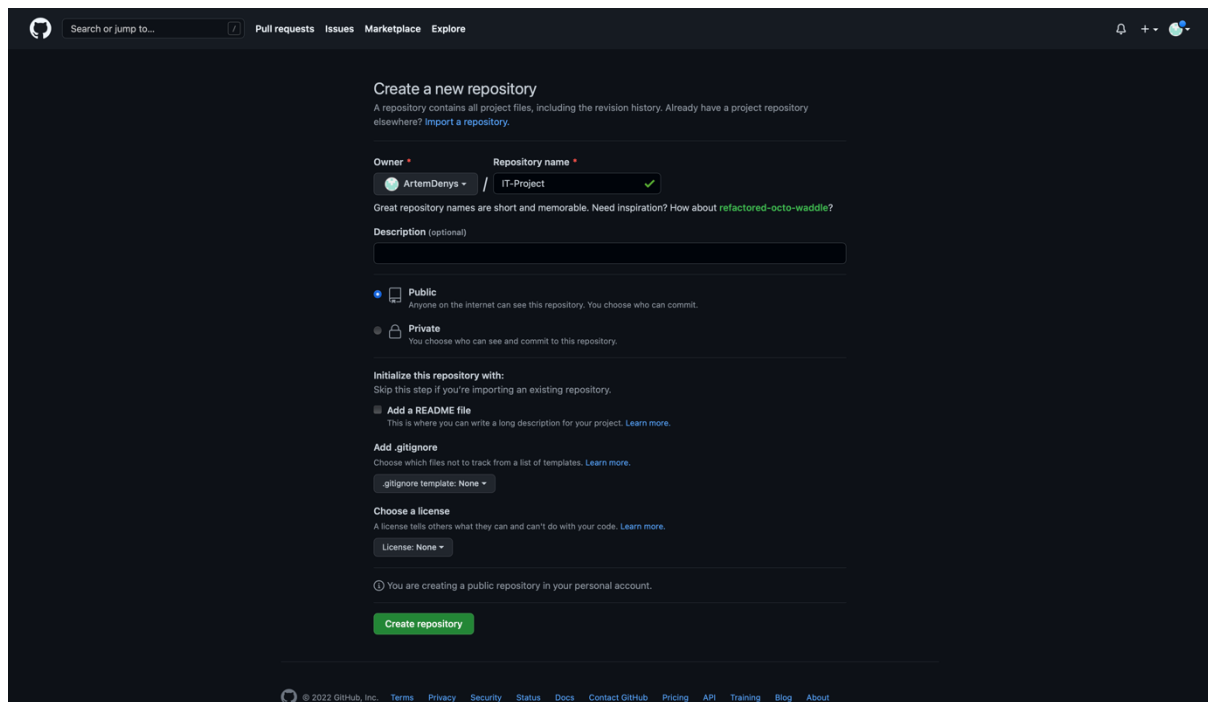


Рис. 1. Створення репозиторію.

Далі можна побачити інструкцію по роботі з репозиторієм.

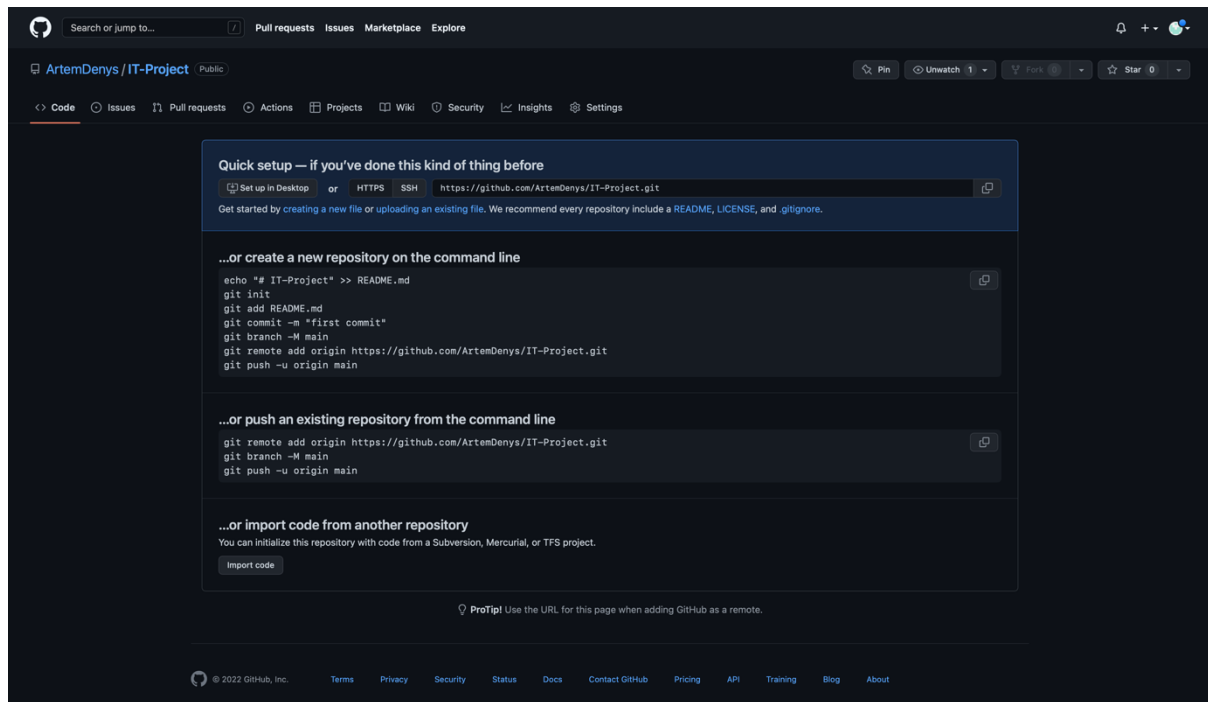


Рис. 2. Вигляд репозиторію.

Обираємо папку проекту для подальшої роботи з нею через термінал. Та ініціалізуємо її за допомогою команди **git init**.

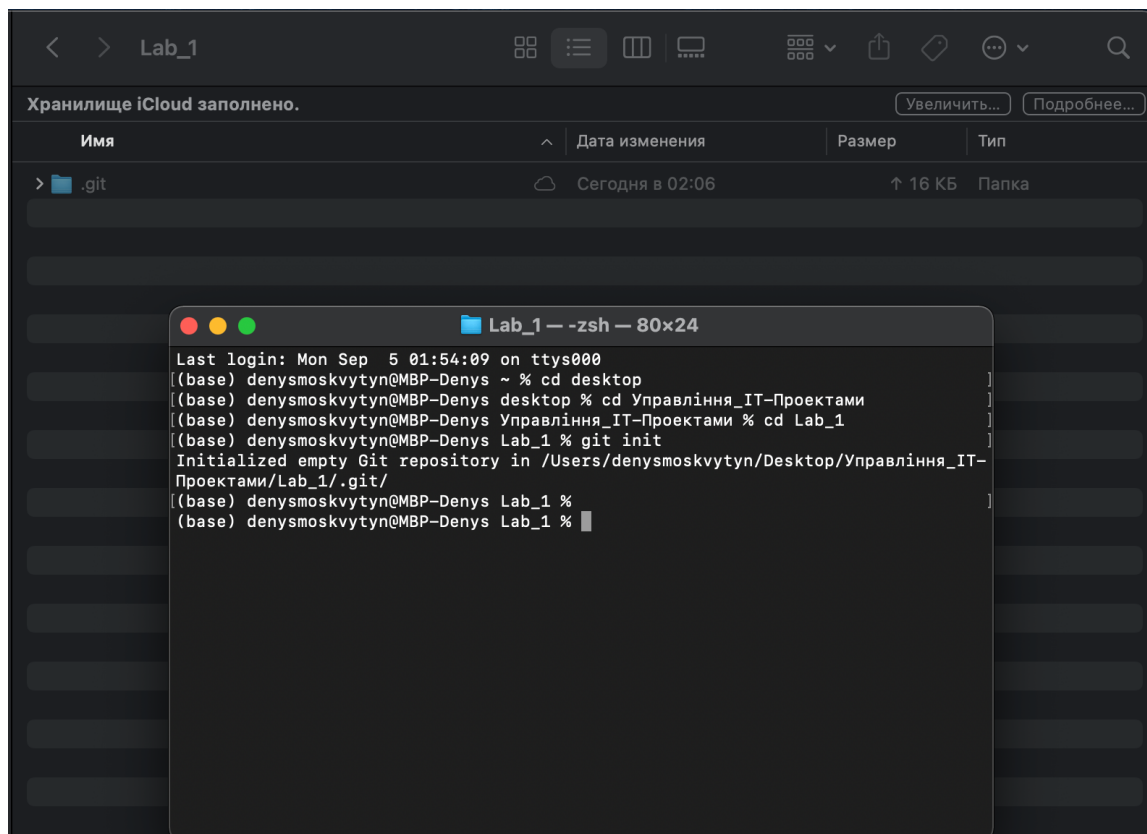


Рис. 3. Робота з папкою через термінал та її ініціалізація.

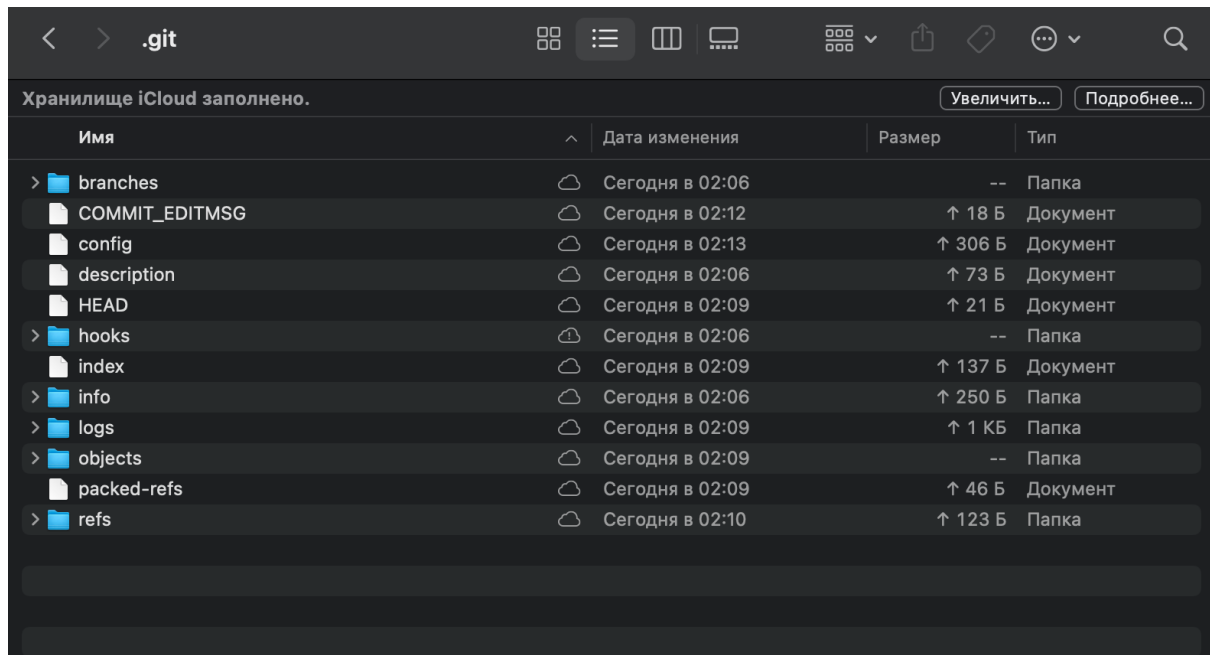


Рис. 4. Вміст папки «.git».

Додаємо новий файл «Lab_1.py» в директорію репозиторію, який будемо загрузити в наш репозиторій, за допомогою команди ***git add Lab_1.py***.

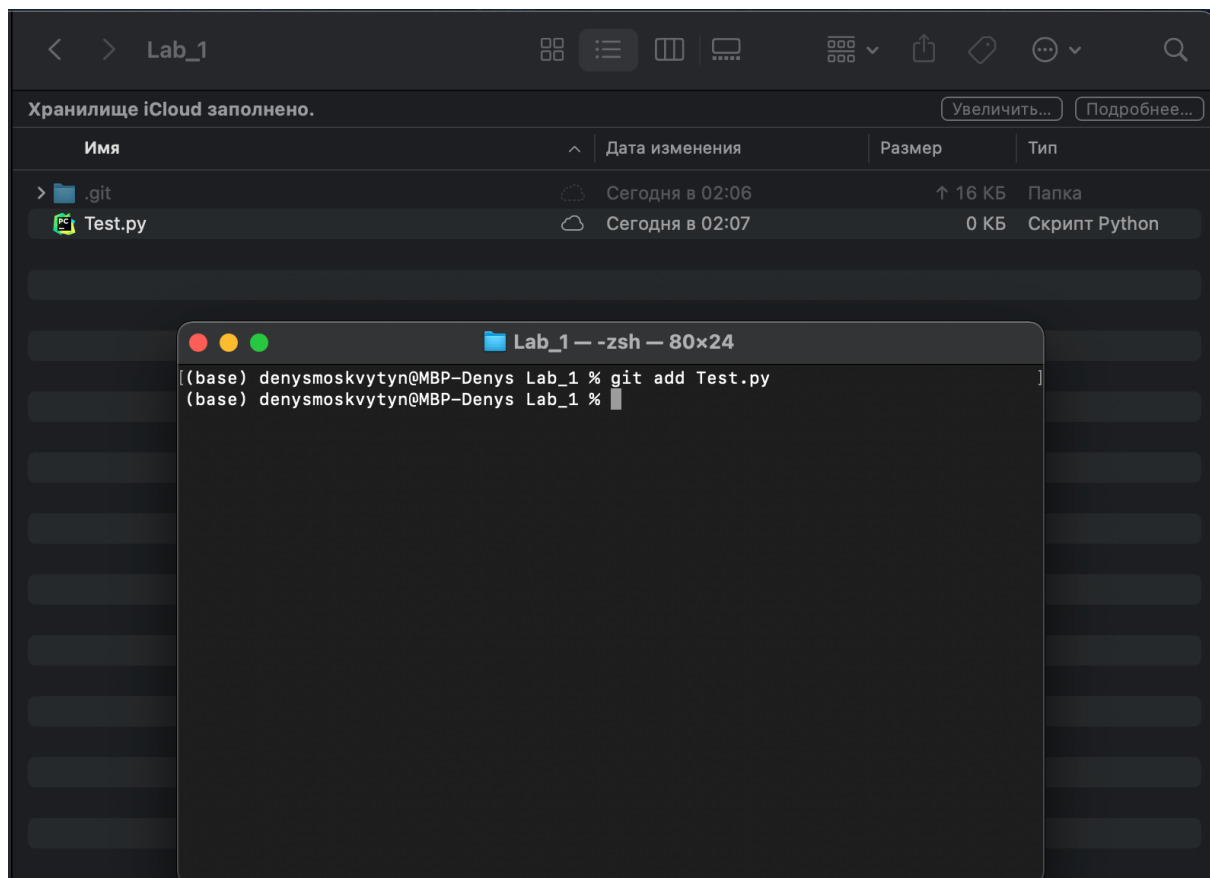


Рис. 5. Виконання команди ***git add newfile.py***.

Тепер добавимо перший коміт до файлу за допомогою команди `git commit -m ""`. Назвемо його « My first commit ».

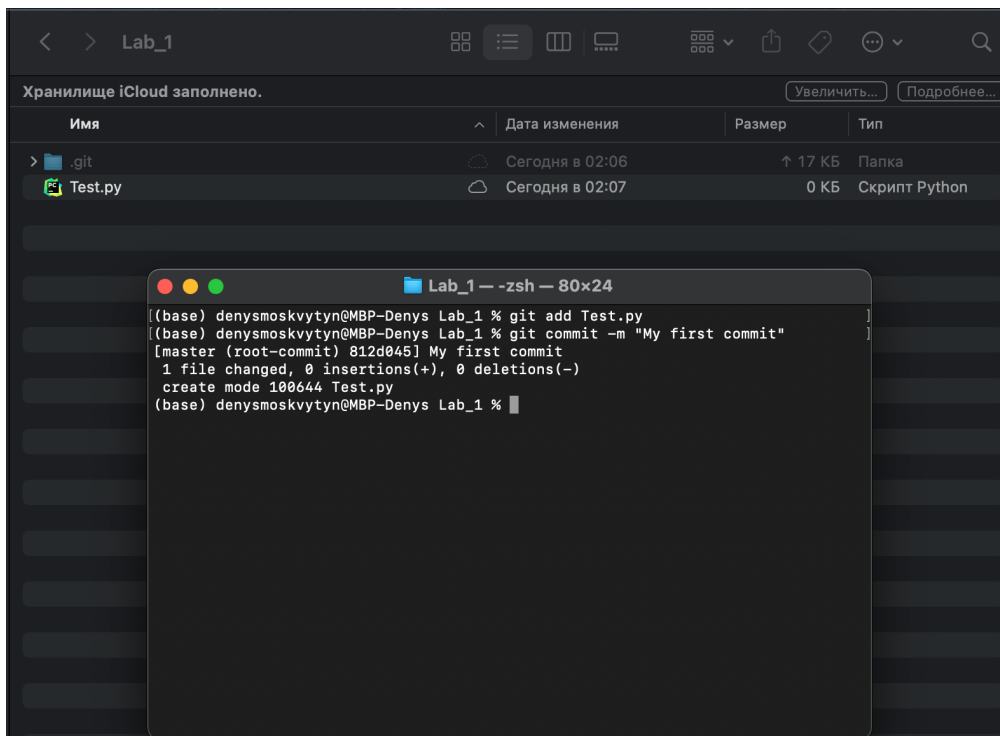


Рис. 6. Виконання коміту.

Створюємо гілку під назвою «main». За допомогою команди ***git branch -M main***

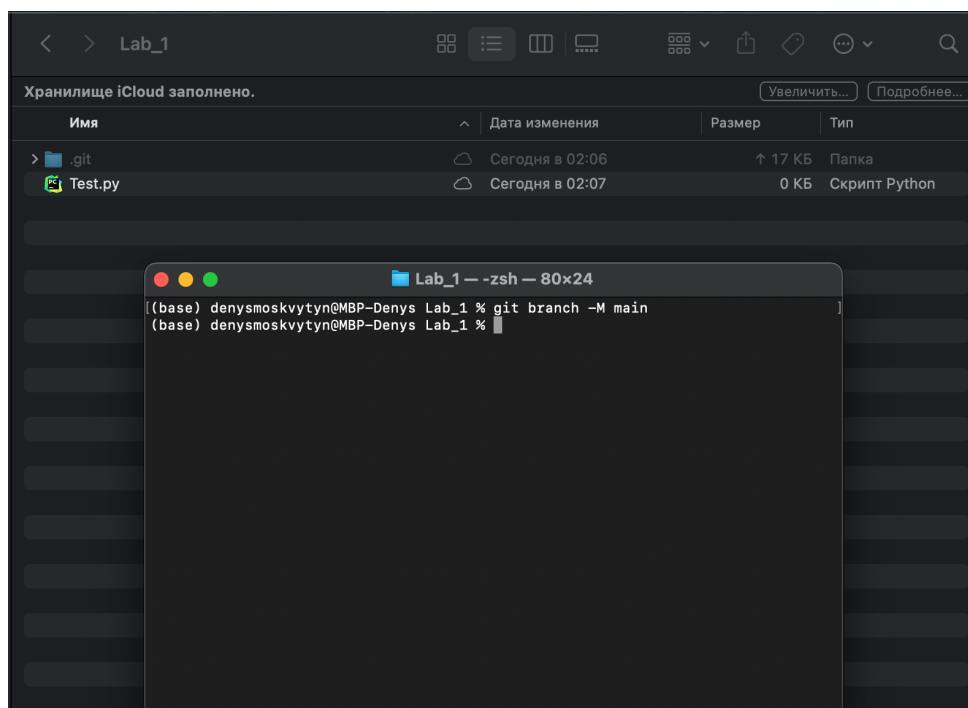


Рис. 7. Створення нової гілки.

Команда ***git remote add*** використовується підключення до віддаленого репозиторію. Для цього нам необхідно ввести ***HTTPS/SSH*** адресу розміщену у віддаленому репозиторії.

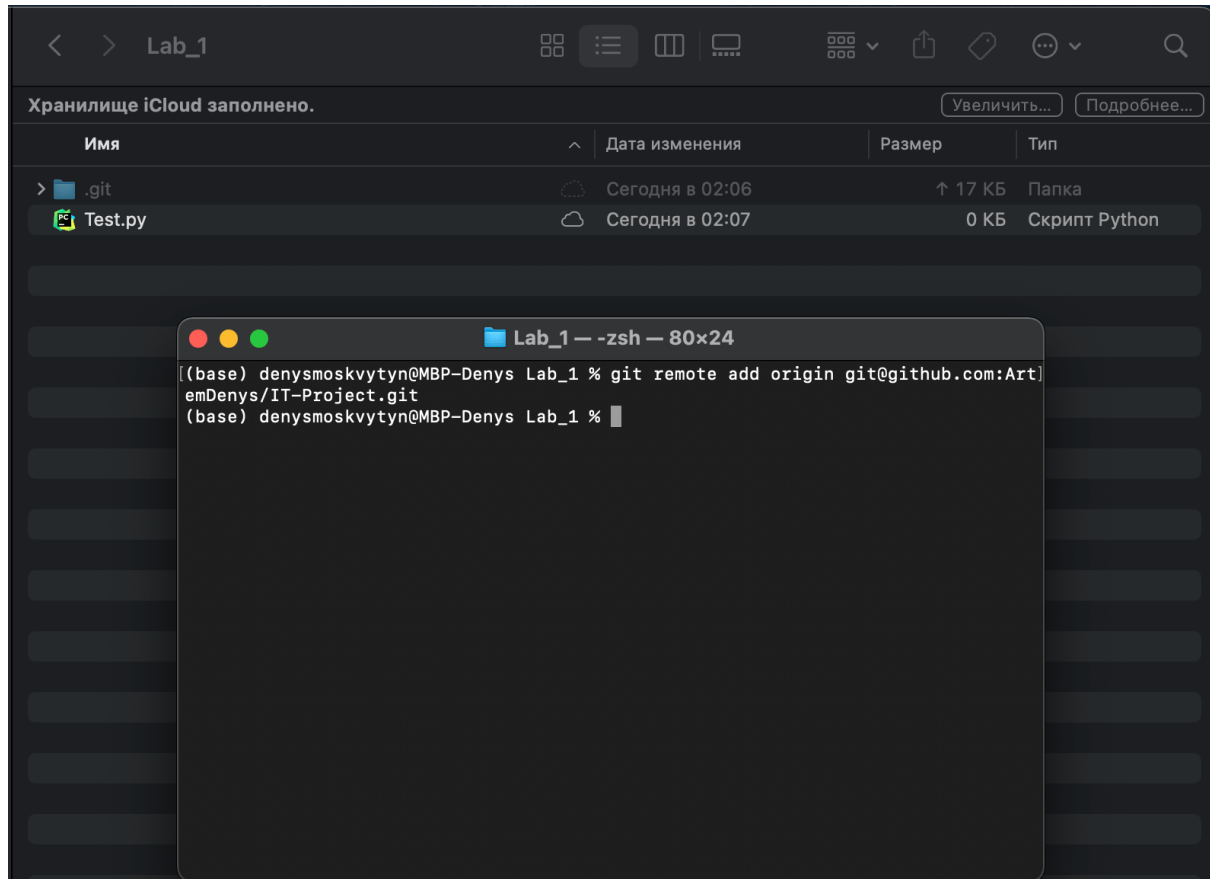


Рис. 8. Підключення до віддаленого репозиторію.

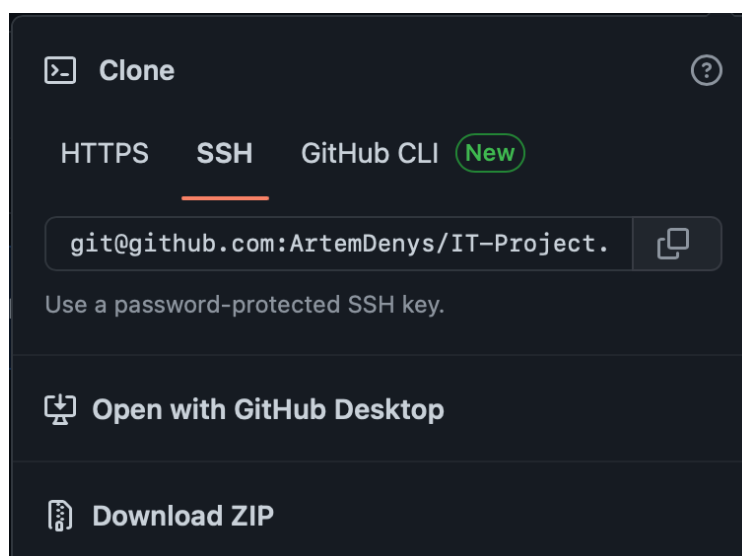
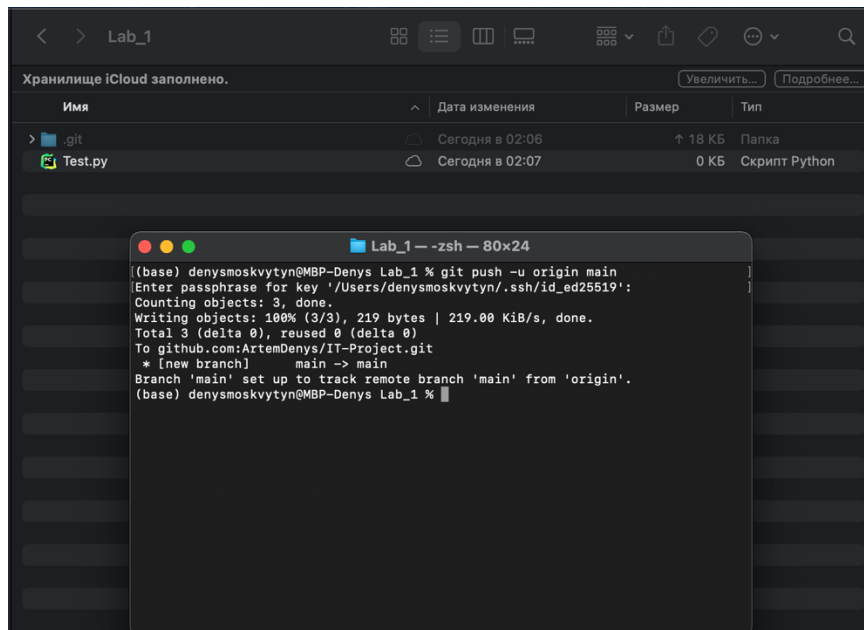


Рис. 9. Адреса віддаленого репозиторію.

Нам залишається зробити **push** даних на GitHub. Для цього скористаємось командою **git push -u origin main**.



```
(base) denysmoskvytn@MBP-Denys Lab_1 % git push -u origin main
Enter passphrase for key '/Users/denysmoskvytn/.ssh/id_ed25519':
Counting objects: 3, done.
Writing objects: 100% (3/3), 219 bytes | 219.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:ArtemDenys/IT-Project.git
 * [new branch]    main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
(base) denysmoskvytn@MBP-Denys Lab_1 %
```

Рис. 10. Відправка файлу на репозиторій GitHub.

Після виконання команди перевіряємо статус виконання у віддаленому репозиторії.

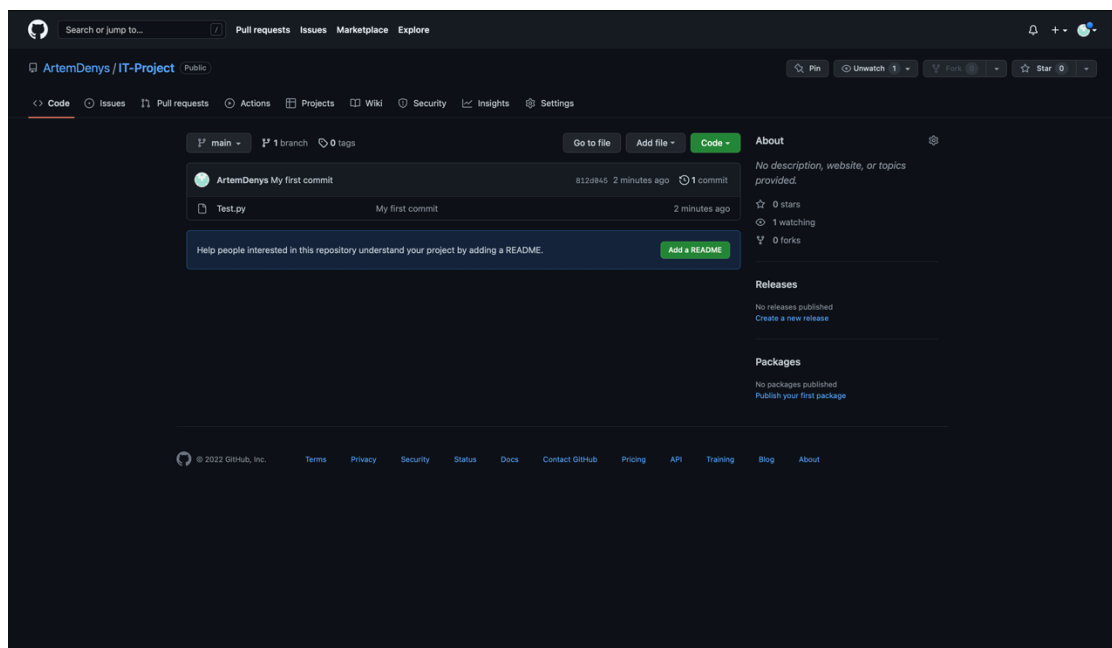


Рис. 11. Перевірка розміщення файлу.

Висновки:

У ході виконання даної лабораторної роботи я ознайомився з базовими системами контролю версій. Набув навички використання технічних засобів та протоколів для роботи з системами контролю версій.

Відповіді на контрольні питання:

1. Що таке системи контролю версій? Для чого вони використовуються?

Система контролю версій – програмне забезпечення для полегшення роботи з інформацією, що змінюється. Система контролю версій дозволяє зберігати декілька версій одного й того ж документа, при необхідності звертатись до більш ранніх версій, визначати, хто й коли зробив ті чи інші зміни, та багато іншого. Такі системи найбільш широко використовуються при розробці програмного забезпечення для зберігання вихідних кодів розроблюваної програми. Однак вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю електронних документів, що неперервно змінюються. Зокрема, системи контролю версій застосовуються в САПР, зазвичай в складі систем управління даними про виріб (PDM). Контроль версій використовується в інструментах конфігураційного управління.

2. Що виконують команди checkout та clone?

Вони витягують робочу копію проекту або тієї його частини, з якою доведеться працювати. За командою вилучення встановлюється з'єднання з сервером, і проект (або його частина – один з каталогів з підкаталогами) у вигляді дерева каталогів і файлів копіюється на комп'ютер розробника. Зазвичай вилучається, або ж клонується остання версія проекту.

3. Що виконує команда commit?

Ця команда передає створені зміни розробником на сервер, або в основну гілку, якщо робота над завданням повністю завершена, чи в окрему гілку даного завдання. Простими словами, команда фіксує зроблені зміни розробником.

4. Що таке branch? Для чого вони використовуються?

Гілка — напрям розробки, який існує незалежно від іншого напрямку, але має з ним спільну історію. Гілка починається як копія чогось і, рухаючись з цієї точки, створює свою власну історію. Тому для таких змін звичайною практикою є створення гілок, тобто «відгілкування» від стовбура в якійсь версії нового варіанту проекту або його частини, розробка в якому ведеться паралельно зі змінами в основній версії. Гілка створюється спеціальною командою. Робоча копія гілки може бути створена заново звичайним чином (командою вилучення робочої копії, із зазначенням адреси або ідентифікатора гілки), або шляхом перемикання наявної робочої копії на задану гілку.

5. Принципи виконання merge?

Саме це може робитися командою злиття:

Базовий робочий цикл при використанні гілок залишається точно таким же, як і в загальному випадку: розробник періодично оновлює робочу копію (якщо з гілкою працює більше однієї людини) і фіксує в ній свою щоденну роботу. Іноді гілку розробки так і залишається самостійною (коли зміни породжують новий варіант проекту, який далі розвивається окремо від основного), але найчастіше, коли робота, для якої створена гілка, виконана, гілка реінтегрується в стовбур (основну гілку).

6. Що таке блокування? Які його недоліки?

Механізм блокування дозволяє одному з розробників захопити в монопольне використання файл або групу файлів для внесення в них змін. На той час, поки файл заблокований, він залишається доступним всім іншим розробникам тільки для читання, і будь-яка спроба внести в нього зміни відкидається сервером.

Недоліки:

- Блокування просто заважають продуктивній роботі, оскільки змушують чекати звільнення заблокованих файлів, хоча в більшості випадків навіть спільні зміни одних і тих же файлів, які робляться в ході різних за змістом робіт, не перетинаються і об'єднуються при злитті автоматично.

- Частота виникнення конфліктів і складність їх вирішення в більшості випадків не настільки великі, щоб створити серйозні труднощі. Виникнення ж серйозного конфлікту змін найчастіше сигналізує або про істотне розходження в думках різних розробників щодо дизайну одного і того ж фрагмента, або про неправильну організацію роботи (коли два або більше розробники роблять одне і те ж).

- Блокування створюють адміністративні проблеми. Типовий приклад: розробник може забути зняти блокування з зайнятих ним файлів, йдучи у відпустку. Для вирішення подібних проблем доводиться застосовувати адміністративні заходи, у тому числі включати в систему технічні засоби для скидання невірних блокувань, а за їх наявності на приведення системи в порядок витрачається час.

7. Що таке tag? Для чого він використовується?

Тег (tag) - це символічна мітка, яка може бути пов'язана з певною версією файлу і/або каталогу в репозиторії. За допомогою відповідної команди всім або частині файлів проекту, що відповідає певним умовам (наприклад, входять до головної версії головної гілки проекту на певний момент часу) може бути присвоєна задана мітка. Таким чином можна ідентифікувати версію проекту (версія «XX.XXX.XXX» - це набір версій

файлів репозиторію, мають тег «XX.XXX.XXX»), зафіксувавши таким чином його стан на деякий бажаний момент. Система тегів досить гнучка і дозволяє помітити одним тегом і не одночасні версії файлів і каталогів. Це дозволяє зібрати «версію проекту» будь-яким довільним чином.

8. Який цикл розробки проекту з використанням VCS?

1. Будь-які робочі, тестові або демонстраційні версії проекту збираються тільки з репозиторію системи. «Персональні» складання, що включають ще незафіксовані зміни, можуть робити тільки розробники для цілей проміжного тестування. Таким чином, гарантується, що репозиторій містить все необхідне для створення робочої версії проекту .
2. Поточна версія головної гілки завжди коректна. Не допускається фіксація в головній гілці неповних або не пройдених хоча б попереднє тестування змін. У будь-який момент збірка проекту, проведена з поточної версії, повинна бути успішною.
3. Будь-яка значима зміна має оформлятися як окрема гілка. Проміжні результати роботи розробника фіксуються в цю гілку. Після завершення роботи над зміною гілка об'єднується зі стовбуром. Винятки допускаються тільки для дрібних змін, робота над якими ведеться одним розробником протягом не більше ніж одного робочого дня.
4. Версії проекту позначаються тегами. Виділена і позначена тегом версія більш ніколи не змінюється.

9. Які протоколи використовує Git?

В більшості прийнято використовувати SSH. Віддалений доступ до репозиторіїв Git забезпечується git-daemon, gitosis, SSH- або HTTP-сервером. TCP-сервіс git-daemon входить в дистрибутив Git і є разом з SSH найбільш поширеним і надійним протоколом доступу. Метод доступу по HTTP, незважаючи на ряд обмежень, дуже популярний в контрольованих мережах, тому що дозволяє використовувати існуючі конфігурації мережевих фільтрів.

10. Що робить команда git –diff

git diff – показує відмінності між деревами проекту, комітів і т.д.

Свого роду підмножиною команди git log можна вважати команду git diff, визначальну зміни між об'єктами в проекті - деревами (файлів і директорій).

11. Чим відрізняється команда git reset —soft від git reset —hard?

Крім роботи з індексом, git reset дозволяє скинути стан проекту до якого-небудь з комітів в історії. У git дана дія може бути двох видів: «м'якого»

(soft reset) і «жорсткого» (hard reset). «М'який» (з ключем **--soft**) резет залишить недоторканими ваші індекс і все дерево файлів і директорій проекту, повернеться до роботи з вказаним комітів. Іншими словами, якщо ви виявляєте помилку у щойно скоєному коміті або коментарі до нього, то легко можна виправити ситуацію.

«Жорсткий» резет (ключ **--hard**) - команда, яку слід використовувати з обережністю. `git reset --hard` поверне дерево проекту та індекс у стан, відповідне вказаному комітів, видаливши зміни наступних комітів :

```
git add
```

```
git commit -m «destined to death»
```

```
git reset --hard HEAD ~ 1 - більше ніхто і ніколи не побачить цей коміт...
```

```
git reset --hard HEAD ~ 3 - ... вірніше , три останні коміти.
```

Якщо команда досягне точки розгалуження, видалення комітів не відбудеться.