Лабораторна робота №1. Знайомство з GIT

Мета роботи: ознайомитись з основними концепціями та командами системи контролю версій Git. Навчитись створювати репозиторії, відстежувати зміни в файлах, здійснювати коміти, працювати з гілками.

Теоретичні відомості

Git являє собою розподілену систему контролю версій, що забезпечує ефективне управління змінами у файлах проекту в процесі його розробки. Цей інструмент відіграє ключову роль у сферах, де важливо зберігати історію модифікацій та мати можливість повернутись до попередніх станів файлів, таких як розробка програмного забезпечення, створення документації та інші види проектної діяльності.

У контексті систем контролю версій, Git належить до класу розподілених систем, які відрізняються від централізованих тим, що кожен розробник має певну копію репозиторію на своєму локальному комп'ютері, включаючи всю історію змін. Такий підхід забезпечує значну гнучкість та підвищену надійність, оскільки розробники можуть працювати автономно, навіть за відсутності постійного підключення до мережі. На противагу цьому, у централізованих системах контролю версій існує єдиний центральний репозиторій, з яким взаємодіють усі розробники. Це може створювати певні обмеження, особливо у випадку проектів або розподілених систем.

У Git репозиторій функціонує як централізоване сховище, де зберігаються всі файли проекту та їх хронологія змін. Репозиторій може бути локальним або віддаленим. Коміт, в свою чергу, представляє собою знімок стану всіх файлів проекту на певний момент часу. Кожен коміт має унікальний ідентифікатор, повідомлення, що описує внесені зміни, та посилання на попередній коміт, формуючи таким чином ланцюжок модифікацій проекту.

Гілки в Git дозволяють створювати паралельні лінії розробки, що дає змогу працювати над різними функціональними можливостями або виправленнями одночасно, не впливаючи на основну гілку розробки. Такий підхід забезпечує гнучкість та можливість експериментування, дозволяючи розробникам ізолювати свої зміни та інтегрувати їх з основною гілкою лише після досягнення необхідного рівня готовності. Процес інтеграції змін з однієї гілки в іншу називається злиттям. Git автоматично намагається виконати злиття, але у випадку виникнення конфліктів, наприклад, коли два розробники змінили один і той самий фрагмент коду, їх необхідно вирішити вручну.

Застосування Git надає ряд переваг у процесі розробки програмного забезпечення. Він дозволяє ефективно відстежувати всі зміни, внесені до файлів проекту, та за потреби повертатися до попередніх версій. Git також сприяє покращенню співпраці між розробниками, надаючи їм можливість працювати над проектом одночасно, не заважаючи один одному. Крім того, гілки

дозволяють проводити експерименти з новими ідеями або функціями, не ризикуючи пошкодити основний код. Віддалені репозиторії забезпечують додатковий рівень захисту даних, зберігаючи резервну копію проекту на сервері, а також спрощують обмін кодом з іншими розробниками.

Для інсталяції Git відвідайте офіційний веб сайт за адресою https://git-scm.com/downloads.

- **Git** – це розподілена система контролю версій, що дозволяє ефективно керувати змінами в коді та інших файлах проекту.
- **Репозиторій** – це сховище, де зберігається історія змін файлів проекту.
- **Коміт** – це збережений стан файлів проекту на певний момент часу.
- **Гілка** – це окрема лінія розробки, що дозволяє працювати над різними функціями або виправленнями паралельно.

Команди Git

- **git init** – ініціалізація порожнього репозиторію
- **git add <ім'я файлу> або git add .** – додавання файлів до коміту
- **git commit -m "message"** – створення коміту
- **git status** – показати зміни в репозиторії
- **git diff** – показати конкретні зміни в коді
- **git push** – завантажити зміни в віддалений репозиторій
- **git pull** – отримання змін з віддаленого репозиторія
- **git branch <ім'я гілки>** - створення гілки на основі поточної
- **git checkout <ім'я гілки>** - перемикання поточної гілки на вказану
- **git checkout -b <ім'я гілки>** - створення нової гілки на основі поточної та одразу перемикання на неї

Зміст роботи

Завдання 1: Встановити Git.

Завдання 2: Увійти на сайт https://git.ztu.edu.ua/ та створити репозиторій. Для входу потрібно використовувати дані з облікового запису університету. Необхідно створити репозиторій з назвою oop-lab1. Також необхідно надати доступ до репозиторію викладачу.

Завдання 3: Створити консольний проект Visual C#.

Параметри проекту:

- Назва рішення: OOPLab1
- Назва проекту: GitConsoleApp

Завдання 4: Ініціалізувати репозиторій.

4.1 Необхідно відкрити каталог, де знаходиться рішення, в командному рядку і запустити команду. Як аналог можна використовувати Git Bash, або Developer PowerShell у Visual Studio

```
git init
```

4.2 Після ініціалізації репозиторію необхідно створити .gitignore файл, так

як не всі файли потрібно завантажувати у репозиторій. Потрібно додати ось такий контент в .gitignore

```
# build output
bin/
obj/

# user preferences
.vs/
```

Завдання 5: Додати файли до репозиторію та створити перший коміт

5.1 Перш ніж додати файли, потрібно налаштувати користувача. Для цього необхідно виконати команди git config і задати email та ім'я (блок <email> потрібно замінити власним email, блок <name> потрібно замінити власним іменем та фамілією)

```
git config --global user.email "<email>"
git config --global user.name "<name>"
```

5.2 Щоб додати файли проекту до репозиторію необхідно запустити команду

```
git add .
```

5.3 Після цього потрібно створити перший коміт, для цього використовуємо команду

```
git commit -m "Створено консольний додаток C#"
```

5.4 Для перевірки чи коміт був створений, необхідно запустити команду

```
git log
```

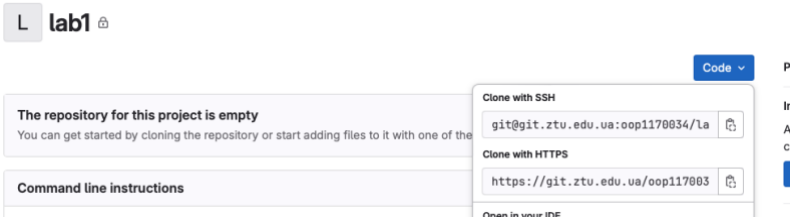
Якщо все вдало, то результат команди виглядатиме приблизно так

```
PS Y:\OOP\Lab1\Lab1> git log
commit 01410d37148c92b9eef813dae0637f10f198fcfc (HEAD -> master)
Author: Denys Datsiuk <kkik_ddv@ztu.edu.ua>
Date:   Mon Aug 19 17:07:15 2024 +0300

    feat: add c# app
```

Завдання 6: Підключення репозиторію та відправка коміту в віддалений репозиторій

6.1 Знаходимо адресу репозиторію в GitLab. Потрібно скопіювати HTTPS адресу



6.2 Додаємо адресу віддаленого репозиторію до локального репозиторію

```
git remote add origin https://git.ztu.edu.ua/oop1170034/lab1.git
```

6.3 Відправляємо локальний коміт в віддалений репозиторій

```
git push
```

При відправці коміту в пустий репозиторій може виникнути ось така помилка

```
PS Y:\OOP\Lab1\Lab1> git push
fatal: The current branch master has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin master

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

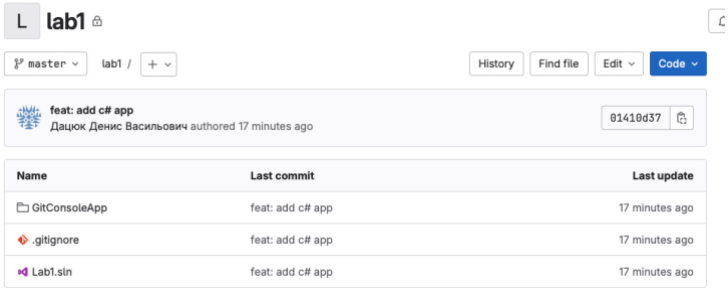
Для її вирішення просто запускаємо команду

```
git push –set-upstream origin master
```

і після цього ще раз запускаємо

```
git push
```

Якщо все пройшло успішно, то в віддаленому репозиторії з’являться файли проекту



Завдання 7: Робота з гілками

7.1 Необхідно створити нову гілку з назвою feature/math

7.2 Згідно варіанту реалізувати математичну формулу в коді. Обчислити значення функції на відрізку [-10, 10] з кроком 1.

1	$y = \frac{x^2 + 5}{\sqrt{x}}$	9	$y = \frac{1}{\tan(x)}$
2	$y = \sqrt{x} + x^3$	10	$y = \frac{x}{\sin(x^5)}$
3	$y = \frac{1}{x} + 10$	11	$y = 1 - x^2 + \frac{1}{x}$
4	$y = \cos(x) - \frac{1}{x^3}$	12	$y = \sqrt{\sin(x)}$
5	$y = \sqrt{5 - x^2}$	13	$y = \sqrt{x} - \cos(x)$
6	$y = \frac{1}{\sqrt[3]{x} + 5}$	14	$y = e^x - x^2$
7	$y = 2\sin(x) - \cos(x)$	15	$y = x^2 + 3x + 5$
8	$y = \ln(\lvert x \rvert + 1)$		

7.3 Зробити коміт та завантажити його в репозиторій

Контрольні запитання

1. Що таке система контролю версій і чому вона важлива в розробці програмного забезпечення?
2. Що таке репозиторій?
3. Поясніть поняття коміту. Яка інформація зберігається в коміті?
4. Яка різниця між локальним і віддаленим репозиторієм у системі контролю версій?
5. Що таке pull і push у Git? Яка між ними різниця?
6. Як працює команда merge, і які конфлікти можуть виникати при її використанні?
7. Що таке .gitignore, і для чого він використовується?
8. Як переглянути історію змін у Git? Які команди для цього використовуються?
9. Як працює команда stash, і для чого вона використовується в Git?
10. Що таке гілка в Git і для чого вона використовується?