Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота №1

із дисципліни «*Технології розроблення програмного забезпечення*»

Тема: «Системи контроля версій.
Розподілена система контролю версій «Git»»

Виконала:
Студент групи ІА-31
Самелюк А.С.

Перевірив:
Мягкий М.Ю.

Київ-2025

Тема: Системи контроля версій. Розподілена система контролю версій «Git».

Мета: Навчитися виконувати основні операції в роботі з децентралізованими системами контролю версій на прикладі роботи з сучасною системою Git.

Хід роботи

1. Створити локальний репозиторій.

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz
$ git init lab_01
Initialized empty Git repository in D:/lab_trpz/lab_01/.git/

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz
$ cd lab_01
```

2. Створити дві гілки.

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (master)
$ git commit --allow-empty -m "empty"
[master (root-commit) 6ed9891] empty

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (master)
$ git branch br1
```

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (master)
$ git switch -c br2
Switched to a new branch 'br2'
```

3. Створити два файли та додати їх на гілку br2.

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br2)
$ echo "1" > file1.txt

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br2)
$ echo "2" > file2.txt

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br2)
$ git add file1.txt
warning: in the working copy of 'file1.txt', LF will be replaced by CRLF the nex
t time Git touches it

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br2)
$ git commit -m "commit file1"
[br2 919a7b1] commit file1
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br2)
$ git add .
warning: in the working copy of 'file2.txt', LF will be replaced by CRLF the nex
t time Git touches it

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br2)
$ git commit -m "commit file2"
[br2 e0b85a1] commit file2
 1 file changed, 1 insertion(+)
 create mode 100644 file2.txt
```

4. Створити файл і додати на гілку br1.

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br2)
$ git switch br1
Switched to branch 'br1'

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br1)
$ echo "A" > file1.txt

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br1)
$ git add .
warning: in the working copy of 'file1.txt', LF will be replaced by CRLF the nex
t time Git touches it

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br1)
$ git commit -m "commit file1"
[br1 608fa5a] commit file1
 1 file changed, 1 insertion(+)
 create mode 100644 file1.txt
```

5. Вивести історію комітів.

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br1)
$ git log
commit 608fa5a08d58cd35dbb2375c83e579368b344bf0 (HEAD -> br1)
Author: Sameliuk <sameliuk.anastasia@gmail.com>
Date:   Sat Sep 13 08:40:43 2025 +0300

    commit file1

commit 6ed9891ac3e5a8dfea12e0afe12bf09883d7683e (master)
Author: Sameliuk <sameliuk.anastasia@gmail.com>
Date:   Sat Sep 13 08:36:25 2025 +0300

    empty

Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br1)
$ git log --graph --all
* commit 608fa5a08d58cd35dbb2375c83e579368b344bf0 (HEAD -> br1)
| Author: Sameliuk <sameliuk.anastasia@gmail.com>
| Date:   Sat Sep 13 08:40:43 2025 +0300
|
|     commit file1
|
| * commit e0b85a12c6a13c1e159290e3d2d7ca0dcfd8525d (br2)
| | Author: Sameliuk <sameliuk.anastasia@gmail.com>
| | Date:   Sat Sep 13 08:39:46 2025 +0300
| |
| |     commit file2
| |
| * commit 919a7b18e48e4d38fc51fd08d6102965e9190856
|/  Author: Sameliuk <sameliuk.anastasia@gmail.com>
|   Date:   Sat Sep 13 08:38:48 2025 +0300
|
|       commit file1
|
* commit 6ed9891ac3e5a8dfea12e0afe12bf09883d7683e (master)
  Author: Sameliuk <sameliuk.anastasia@gmail.com>
  Date:   Sat Sep 13 08:36:25 2025 +0300

      empty
```

6. Зливаємо гілки за допомогою rebase.

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br1)
$ git rebase br2
Auto-merging file1.txt
CONFLICT (add/add): Merge conflict in file1.txt
error: could not apply 608fa5a... commit file1
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --
abort".
```

7. Вирішуємо конфлік.

```
Nastya@Laptop-Nastya MINGW64 /d/lab_trpz/lab_01 (br1)
$ nano file1.txt
```

```
* commit ea351283901f37dd3ac1df07e8c7617046309f5d (HEAD -> br1)
| Author: Sameliuk <sameliuk.anastasia@gmail.com>
| Date:   Sat Sep 13 08:40:43 2025 +0300
|
|     rebase file1
|
* commit e0b85a12c6a13c1e159290e3d2d7ca0dcfd8525d (br2)
| Author: Sameliuk <sameliuk.anastasia@gmail.com>
| Date:   Sat Sep 13 08:39:46 2025 +0300
|
|     commit file2
|
* commit 919a7b18e48e4d38fc51fd08d6102965e9190856
| Author: Sameliuk <sameliuk.anastasia@gmail.com>
| Date:   Sat Sep 13 08:38:48 2025 +0300
|
* commit ea351283901f37dd3ac1df07e8c7617046309f5d (HEAD -> br1)
| Author: Sameliuk <sameliuk.anastasia@gmail.com>
| Date:   Sat Sep 13 08:40:43 2025 +0300
|
|     rebase file1
|
* commit e0b85a12c6a13c1e159290e3d2d7ca0dcfd8525d (br2)
| Author: Sameliuk <sameliuk.anastasia@gmail.com>
| Date:   Sat Sep 13 08:39:46 2025 +0300
|
|     commit file2
|
* commit 919a7b18e48e4d38fc51fd08d6102965e9190856
| Author: Sameliuk <sameliuk.anastasia@gmail.com>
| Date:   Sat Sep 13 08:38:48 2025 +0300
|
|     commit file1
|
* commit 6ed9891ac3e5a8dfea12e0afe12bf09883d7683e (master)
  Author: Sameliuk <sameliuk.anastasia@gmail.com>
  Date:   Sat Sep 13 08:36:25 2025 +0300
```

Висновок: У роботі було опрацьовано основні принципи та команди розподіленої системи контролю версій Git. Отримано практичні навички створення репозиторію, роботи з гілками, фіксації та об’єднання змін. Використання Git забезпечує зручне

відстеження історії, надійність збереження даних та ефективну колективну роботу над проєктами.

Відповіді на контрольні питання:

1. Що таке система контролю версій (СКВ)?

Система управління версіями (від англ. Version Control System або Source Control System) – програмне забезпечення яке призначено допомогти команді розробників керувати змінами в вихідному коді під час роботи. Система керування версіями дозволяє додавати зміни в файлах в репозиторій і таким чином після кожної фіксації змін мани нову ревізію файлів. Це дозволяє повертатися до попередніх версій коду для аналізу внесених змін або пошуку, які зміни привели до появи помилки. Таким чином можна знайти хто, коли і які зміни зробив в коді, а також чому ці зміни були зроблені. Такі системи найбільш широко використовуються при розробці програмного забезпечення для зберігання вихідних кодів програми, що розробляється.

2. Поясніть відмінності між розподіленою та централізованою СКВ.

Централізована СКВ (Subversion (SVN), CVS)

- Має єдиний центральний сервер, де зберігається повний репозиторій з усією історією змін.
- Користувачі отримують лише робочу копію файлів, а не повну історію.
- Для перегляду історії, створення гілок чи коміту потрібне постійне з’єднання з сервером.
- Якщо центральний сервер виходить з ладу, доступ до історії та можливість комітів зникають.
- Адміністрування та контроль простіші, але є єдина точка відмови.

Розподілена СКВ (Git, Mercurial)

- Кожен розробник має повну копію репозиторію разом з історією змін на локальному комп’ютері.

- Можна працювати офлайн: створювати коміти, переглядати історію, створювати гілки.
- Синхронізація з іншими виконується через push/pull між репозиторіями.
- Відсутня єдина точка відмови: якщо сервер зламається, будь-яка локальна копія може відновити весь проект.
- Гнучкіші для командної роботи, але потребують трохи складнішої організації.

3. Поясніть різницю між stage та commit в Git.

Stage (індекс, staging area) — це проміжна зона, куди додаються зміни перед створенням коміту. Використовується команда:

git add <файл>

Commit — це вже фіксація змін із staging area у локальному репозиторії з коментарем:

git commit -m "опис змін"

Отже, stage — це підготовка, а commit — це збереження.

4. Як створити гілку в Git?

Створити гілку: git branch <ім'я_гілки>

Або одразу перейти в нову гілку: git checkout -b <ім'я_гілки>

5. Як створити або скопіювати репозиторій Git з віддаленого серверу?

Клонувати репозиторій (git clone) – отримати копію репозиторію на локальну машину для подальшої роботи з ним: git clone https://github.com/user/repo.git

6. Що таке конфлікт злиття, як створити конфлікт, як вирішити конфлікт?

Конфлікт злиття (merge conflict) виникає, коли дві гілки змінюють один і той самий рядок у файлі, і Git не може автоматично вибрати, яке змінення залишити.

Створити конфлікт можна, якщо в одній гілці змінити рядок, а в іншій — теж цей самий рядок, а потім спробувати зробити git merge.

Вирішити конфлікт:

1. Відкрити файл із конфліктом.
2. Залишити правильний варіант (або об’єднати зміни).
3. Позначити як вирішений: git add <файл>, git commit

7. В яких ситуаціях використовуються merge, rebase, cherry-pick?

- merge — коли треба об’єднати дві гілки, зберігаючи історію.
- rebase — коли треба «переписати» історію, перемістивши коміти на іншу базову гілку, щоб історія була лінійною.
- cherry-pick — коли потрібно взяти один конкретний коміт з іншої гілки і застосувати його у свою.

8. Як переглянути історію змін Git репозиторію в консолі?

git log

Короткий варіант: git log --oneline --graph --all

9. Як створити гілку в Git не використовуючи команду git branch?

Через checkout -b або новий синтаксис switch:

git checkout -b <ім'я_гілки> або git switch -c <ім'я_гілки>

10. Як підготувати всі зміни в поточній папці до коміту?

git add .

11. Як підготувати всі зміни в дочірній папці до коміту?

git add <шлях_до_папки>/

12. Як переглянти перелік наявних гілок в репозиторії?

git branch

Або разом з віддаленими: git branch -a

13. Як видалити гілку?

Локально: git branch -d <ім'я_гілки> (або -D для примусового видалення)

На віддаленому сервері: git push origin --delete <ім'я_гілки>

14. Які є способи створення гілки та в чому між ними різниця?

1. git branch <ім'я> — створює гілку, але не переключає на неї.
2. git checkout -b <ім'я> — створює і одразу перемикає на гілку.
3. git switch -c <ім'я> — новий синтаксис (аналог checkout), теж створює і перемикає.

Різниця лише в зручності: branch — створює без переходу, checkout -b і switch -c — створюють і одразу активують.