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

Лабораторна робота №1

із дисципліни «**Технології розроблення програмного забезпечення**»

Тема **«Команди Git»**

Виконав:

студент групи ІА–24

Тильна М.С.

Перевірив:

Мягкий М.Ю.

Київ 2024

Мета: ознайомитися з основними командами системи контролю версій Git

Теоретичні відомості

Git — це система контролю версій, яка дозволяє розробникам відстежувати зміни в коді, співпрацювати над проектами та зберігати історію змін. Основні функції **Git** включають створення знімків (комітів) поточного стану проекту, можливість роботи з гілками для розвитку різних функціональностей незалежно одна від одної, а також злиття і вирішення конфліктів між різними версіями коду. У цій лабораторній роботі ми розглянемо основні команди **Git**, які використовуються для роботи з локальними репозиторіями, а також способи організації роботи з гілками.

Основні команди Git

git init
Створює новий локальний репозиторій у поточній директорії. Ця команда ініціалізує репозиторій і дозволяє Git почати відстеження змін.

- `git init` — створює порожній репозиторій у поточній папці.

git add
Додає зміни у файлах до індексу (стейджингу) для подальшого коміту.

- `git add <файл>` — додає конкретний файл до індексу.
- `git add .` — додає всі файли з поточної директорії.

git remove
Видаляє файл з репозиторію та, за необхідності, з робочого каталогу.

- `git rm <файл>` — видаляє файл з індексу та робочого каталогу.
- `git rm --cached <файл>` — видаляє файл з індексу, але залишає його в робочому каталозі.

git commit
Зберігає знімок стану проєкту з файлами, що були додані до індексу.

- `git commit -m "Опис змін"` — створює коміт з описом змін.
- `git commit -a -m "Опис змін"` — додає і фіксує всі змінені файли, оминувши команду `git add`.

git status
Показує інформацію про поточний стан репозиторію: які файли змінені, які готові до коміту, а які потребують додавання до індексу.

- `git status` — перегляд поточного статусу файлів у репозиторії.

git log
Виводить історію комітів, включаючи інформацію про авторів, час та повідомлення комітів.

- `git log` — перегляд історії всіх комітів.
- `git log --oneline` — скорочений перегляд історії комітів.

git branch
Показує існуючі гілки або дозволяє створити нову гілку.

- `git branch` — показує список усіх локальних гілок.
- `git branch <назва-гілки>` — створює нову гілку з поточного стану.

git checkout
Використовується для перемикання між гілками або створення нової гілки і перемикання на неї одночасно.

- `git checkout <назва-гілки>` — перемикається на існуючу гілку.
- `git checkout -b <назва-гілки>` — створює нову гілку і перемикається на неї.

git switch
Новіша команда для перемикання між гілками, яка частково замінює `git checkout`. Команда `git switch` має спрощену синтаксис для роботи з гілками.

- `git switch <назва-гілки>` — перемикається на існуючу гілку.
- `git switch -c <назва-гілки>` — створює нову гілку і перемикається на неї.

git merge
Зливає зміни з однієї гілки в іншу. Під час злиття Git намагається автоматично об'єднати зміни, але може виникнути конфлікт, якщо зміни в одному й тому ж файлі були внесені в обох гілках.

- `git merge <назва-гілки>` — зливає зміни з вказаної гілки в поточну.

git rebase
Інструмент для переписування історії комітів. Використовується для інтеграції змін з однієї гілки в іншу без створення окремого коміту злиття. Це дозволяє зберегти лінійну історію проєкту.

- `git rebase <назва-гілки>` — змінює базу поточної гілки на вказану, інтегруючи зміни.

git cherry-pick
Використовується для вибіркового перенесення окремих комітів з однієї гілки в іншу. Це корисно, коли потрібно інтегрувати певні зміни без злиття всієї гілки.

- `git cherry-pick <хеш-коміту>` — застосовує вказаний коміт до поточної гілки.

git reset
Використовується для скасування комітів або скасування змін в індексі.

- `git reset <файл>` — знімає файл зі стейджингу.
- `git reset --hard <хеш-коміту>` — повертає репозиторій до конкретного коміту, видаляючи всі зміни після нього.

Хід роботи

1. Створити порожній репозиторій та 3 коміти.

```
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz
$ git init
Initialized empty Git repository in C:/trpz/.git/

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ touch file1.txt file2.txt file3.txt
```

Ініціалізуємо репозиторій. Створюємо три файли .

Створюємо файл в нашій директорії

```
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git add file1.txt
git c
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git commit -m "Commit 1"
[master (root-commit) 243f315] Commit 1
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git add file2.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git commit -m "Commit 2"
[master 69904d5] Commit 2
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file2.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git add file3.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git commit -m "Commit 3"
[master ffcf48f] Commit 3
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file3.txt
```

Додаємо ці файли до системи контролю версій та комітимо

```
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git log --oneline
ffcf48f (HEAD -> master) Commit 3
69904d5 Commit 2
243f315 Commit 1
```

Переглядаємо історію комітів за допомогою git log --oneline, щоб переконатися, що всі три коміти були успішно створені.

2. Перевірити різницю між індексом та останнім комітом.

```
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ touch file4.txt
g
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git add file4.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git diff --cached
diff --git a/file4.txt b/file4.txt
new file mode 100644
index 0000000..e69de29
```

Створюємо файл file4.txt, додаємо його до індексу, перевіряємо зміни за допомогою git diff --cached

3. Створити гілку та перейменувати її.

```
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git switch -c branch1
Switched to a new branch 'branch1'
```

Створюємо нову гілку branch1, перемкнувшись на неї.

```
 create mode 100644 file4.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (branch1)
$ git branch -m branch2

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (branch2)
$ git status
On branch branch2
nothing to commit, working tree clean

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (branch2)
$ git branch
* branch2
  master
```

Перейменовуємо поточну гілку з branch1 на branch2. Перевіряємо статус гілки, щоб переконатися, що немає незбережених змін, і переглядаємо список наявних гілок (branch2 і master).

4. Створити дві гілки, які базуються на master. Зробити злиття за допомогою merge.

```
KTOP-IAR1PH9 MINGW64 /c/trpz (branch2)
itch master
 to branch 'master'

KTOP-IAR1PH9 MINGW64 /c/trpz (master)
eckout -b b3
 to a new branch 'b3'

KTOP-IAR1PH9 MINGW64 /c/trpz (b3)
 > file4.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (b3)
$ git add file4.txt
warning: in the working copy of 'file4.txt', LF will be replaced by CRLF the next time Git touches it

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (b3)
$ git commit -m "Commit4"
[b3 06b1665] Commit4
 1 file changed, 1 insertion(+)
 create mode 100644 file4.txt
```

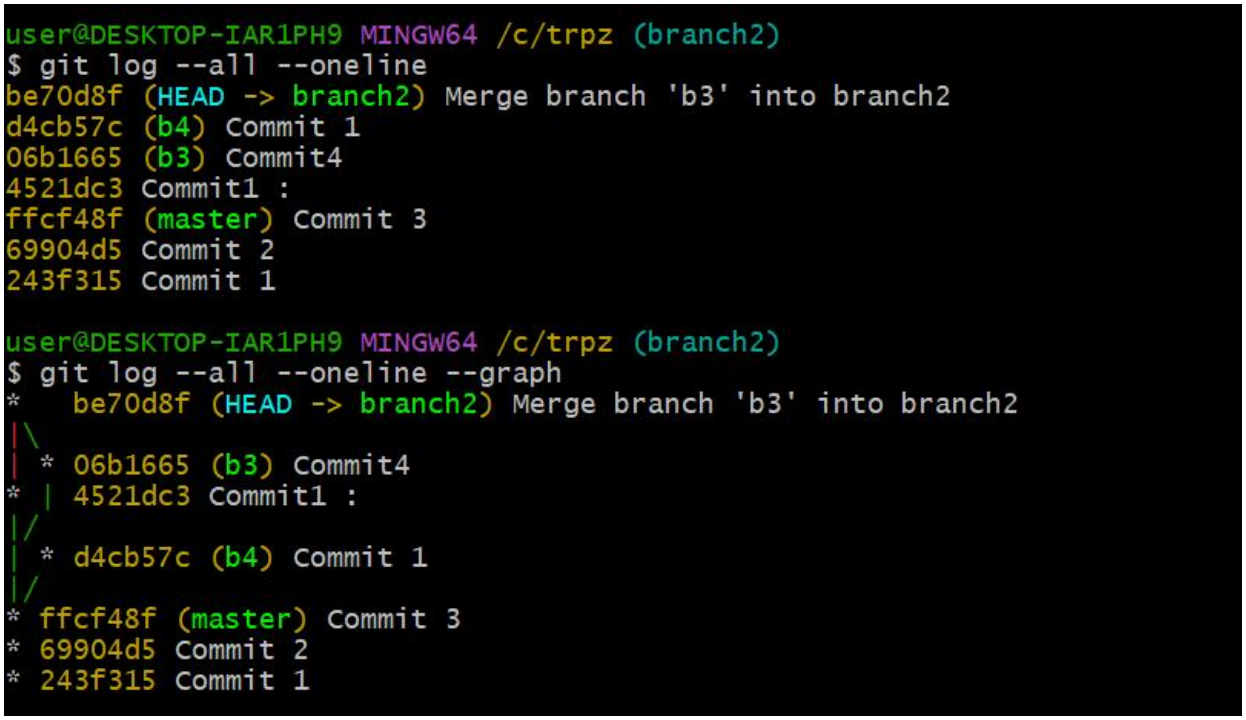
Переключаємось на гілку master.
Створюємо і переходимо на нову гілку b3, яка базується на master.
Створюємо файл file4.txt. Додаємо файл до індексу за допомогою git add, і комітимо.

```
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (b3)
$ git switch master
Switched to branch 'master'

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (master)
$ git switch -c b4
Switched to a new branch 'b4'

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (b4)
$ echo 1 > 1.txt

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (b4)
$ git add 1.txt
warning: in the working copy of '1.txt', LF will be replaced by CRLF the next time Git touches it
gi
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (b4)
$ git commit -m "Commit 1"
[b4 d4cb57c] Commit 1
 1 file changed, 1 insertion(+)
 create mode 100644 1.txt
```

Повертаємось на master і створюмо гілку b4, переключившись на неї. Так само створюємо файл, індексуємо та комітимо.

```
user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (b4)
$ git switch branch2
Switched to branch 'branch2'

user@DESKTOP-IAR1PH9 MINGW64 /c/trpz (branch2)
$ git merge b3
Auto-merging file4.txt
Merge made by the 'ort' strategy.
 file4.txt | 1 +
 1 file changed, 1 insertion(+)
```

Переключаємось на гілку branch2 і об’єдуємо її з гілкою b3. Злиття пройшло успішно, і зміни з b3 були додані до branch2. Гілка branch2 тепер містить зміни з гілки b3, включаючи файл file4.txt. Гілка b4 містить свій власний коміт (додавання 1.txt), і цей коміт не включений в branch2.



Переглядаємо історію всіх комітів за допомогою команд git log --all --oneline і git log --all --oneline --graph, щоб побачити структуру гілок і злиття.

Висновок: в ході виконання даної лабораторної роботи ми познайомились з такою системою контролю версій як Git. Вивчили основні команди для роботи з гітом: ініціалізація репозиторію, додавання файлів у систему контролю, створення коміту, перенесення комітів між гілками, злиття гілок та багато інших. Окрім того ми застосували ці команди на практиці, вирішили конфлікти при переносі комітів та засвоїли вивчений матеріал.