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

Технології розроблення програмного забезпечення

Лабораторна робота №1

Виконала:
Студентка групи ІА-22
Патрик Вікторія

Київ 2024

Системи контролю версій. Git.

Мета: ознайомитись та навчитися працювати з основними командами Git, вивчити способи створення репозиторіїв, комітів, роботи з гілками, злиття та вирішення конфліктів.

Хід роботи:

Крок 1. Створення нової директорії:

```
C:\Users\Vika>mkdir test

C:\Users\Vika>cd test
```

1. mkdir test
Команда створює нову директорію з назвою test у поточному розташуванні (C:\Users\Vika). Це підготовка простору для подальшої роботи з Git.
2. cd test
Команда змінює поточну робочу директорію на щойно створену test, де будуть виконуватись усі подальші дії.

Крок 2. Ініціалізація Git-репозиторію:

```
C:\Users\Vika\test>git init
Initialized empty Git repository in C:/Users/Vika/test/.git/
```

1. git init
Команда ініціалізує новий порожній Git-репозиторій у поточній директорії test. Це створює приховану папку .git, яка містить усі необхідні файли для відстеження змін у проекті. Тепер ця директорія є репозиторієм Git, і можна виконувати команди для контролю версій.

Крок 3. Перевірка статусу репозиторію:

```
C:\Users\Vika\test>git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

1. git status
Ця команда показує поточний стан робочої директорії та індексу (staging area). Вона допомагає дізнатися, чи є зміни, які ще не закомічені, або чи є файли, які не додані на stage.

- On branch master: я знаходжусь на гілці master. Це поточна гілка, в якій я працюю.
- No commits yet: у репозиторії поки що немає комітів, оскільки репозиторій тільки що ініціалізовано, і я ще не фіксувала жодних змін.

- nothing to commit (create/copy files and use "git add" to track): це повідомлення вказує на те, що немає змін, які потрібно комітити. Git пропонує створити або скопіювати файли у директорію та додати їх на stage за допомогою команди git add, щоб почати відслідковувати зміни.

Крок 4. Створення порожнього коміту та перегляд журналу комітів:

```
C:\Users\Vika\test>git commit --allow-empty -m "empty_commit"
[master (root-commit) dda0438] empty_commit

C:\Users\Vika\test>git log
commit dda043893a39382c99517ba57b514da24ceda167 (HEAD -> master)
Author: Vika <patrik.vika2016@gmail.com>
Date:   Sat Oct 12 10:34:43 2024 +0300

    empty_commit
```

1. git commit --allow-empty -m "empty_commit"
Ця команда створює порожній коміт (без змін у файлах) з повідомленням "empty_commit".
   - --allow-empty дозволяє створити коміт навіть тоді, коли немає ніяких змін, які потрібно зафіксувати. Це може бути корисно для тестування або для додавання коментаря в історію репозиторію.

2. git log
Ця команда відображає журнал комітів, де показана інформація про всі коміти в репозиторії.
   - commit dda0438 - хеш цього коміту.
   - (HEAD -> master) - вказує, що цей коміт є останнім в гілці master, і що моя поточна позиція (HEAD) знаходиться саме на ньому.
   - Author - інформація про автора коміту (ім'я та електронна адреса).
   - Date - дата і час створення коміту.
   - empty_commit - повідомлення коміту, яке я вказала.

Крок 5. Створення та перемикання між гілками:

```
C:\Users\Vika\test>git branch branch1

C:\Users\Vika\test>git checkout -b branch2
Switched to a new branch 'branch2'

C:\Users\Vika\test>git switch master
Switched to branch 'master'

C:\Users\Vika\test>git switch -c branch3
Switched to a new branch 'branch3'
```

1. git branch branch1
Ця команда створює нову гілку з назвою branch1, але не перемикає мене на неї. Гілки дозволяють розвивати проект паралельно, не впливаючи на основну гілку.
2. git checkout -b branch2
Ця команда одночасно створює нову гілку branch2 і перемикає мене на неї. Тепер я можу вносити зміни в цій гілці без впливу на інші.
3. git switch master
Ця команда перемикає мене назад на гілку master. Використовуючи switch, я можете швидко змінювати гілки, не вдаючись до команди checkout, що робить це більш зручним.
4. git switch -c branch3
Ця команда створює нову гілку з назвою branch3 і автоматично перемикає мене на неї. Тепер я можете працювати в новій гілці, яка є незалежною від інших гілок у репозиторії.

Крок 6. Перегляд списку гілок:

```
C:\Users\Vika\test>git branch
  branch1
  branch2
* branch3
  master
```

1. git branch
Ця команда виводить список усіх гілок у репозиторії. У результаті я бачу:
branch1 — існуюча гілка, створена раніше.
branch2 — ще одна існуюча гілка.
* branch3 — зірочка (*) вказує, що я зараз перебуваю на цій гілці.

master — основна гілка репозиторію.
Ця команда дозволяє швидко побачити всі гілки та визначити, на якій з них я зараз працюю.

Крок 7. Створення нових файлів у робочій директорії та перевірка їхнього статусу:

```
C:\Users\Vika\test>echo 1 > master.txt

C:\Users\Vika\test>git status
On branch branch3
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        master.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\Vika\test>echo 1 > 1.txt

C:\Users\Vika\test>echo 2 > 2.txt

C:\Users\Vika\test>echo 3 > 3.txt
```

1. echo 1 > master.txt
Ця команда створює файл з назвою master.txt і записує в нього число 1. Файл зберігається в робочій директорії репозиторію.
2. git status
Після створення файлу, команда git status показує поточний стан робочої директорії.

- On branch branch3: я знаходжусь на гілці branch3.
- Untracked files: git показує, що новий файл master.txt є незатреканим (Git ще не відстежує його).
- nothing added to commit but untracked files present: це означає, що у мене є новий файл, але його потрібно додати на stage за допомогою git add, якщо я хочу його відстежувати та включити в наступний коміт.

3. echo 1 > 1.txt / echo 2 > 2.txt / echo 3 > 3.txt
Кожна з цих команд створює нові файли (1.txt, 2.txt, 3.txt) та записує в них відповідні числа (1, 2, 3). Ці файли також залишаються незатреканими до того моменту, поки я не додам їх на stage за допомогою git add.

Крок 8. Додавання файлів, створення комітів у різних гілках та перемикання між гілками:

```
C:\Users\Vika\test>git add master.txt 1.txt 2.txt 3.txt

C:\Users\Vika\test>git commit 3.txt -m "1 commit"
[branch3 aef759c] 1 commit
 1 file changed, 1 insertion(+)
 create mode 100644 3.txt

C:\Users\Vika\test>git switch branch2
A       1.txt
A       2.txt
A       master.txt
Switched to branch 'branch2'

C:\Users\Vika\test>git commit 2.txt -m "2 commit"
[branch2 75c4040] 2 commit
 1 file changed, 1 insertion(+)
 create mode 100644 2.txt

C:\Users\Vika\test>git switch branch1
A       1.txt
A       master.txt
Switched to branch 'branch1'

C:\Users\Vika\test>git commit 1.txt -m "3 commit"
[branch1 f91bdf3] 3 commit
 1 file changed, 1 insertion(+)
 create mode 100644 1.txt

C:\Users\Vika\test>git switch master
A       master.txt
Switched to branch 'master'

C:\Users\Vika\test>git commit master.txt -m "4 commit"
[master 44a923a] 4 commit
 1 file changed, 1 insertion(+)
 create mode 100644 master.txt
```

1. git add master.txt 1.txt 2.txt 3.txt
Ця команда додає файли master.txt, 1.txt, 2.txt та 3.txt на stage, готуючи їх до наступного коміту. Тепер ці файли Git починає відстежувати.
2. git commit 3.txt -m "1 commit"
Цей коміт фіксує файл 3.txt з повідомленням "1 commit".

3. git switch branch2
Я перемикаюсь на гілку branch2.
- A 1.txt, A 2.txt, A master.txt: Після перемикання на гілку, ці файли (1.txt, 2.txt, master.txt) додаються на stage автоматично, оскільки вони вже були на stage у попередній гілці.

4. git commit 2.txt -m "2 commit"
Цей коміт фіксує файл 2.txt з повідомленням "2 commit" у гілці branch2.
5. git switch branch1
Я перемикаюсь на гілку branch1.
- A 1.txt, A master.txt: Після перемикання, файли 1.txt та master.txt знову додаються на stage.

6. git commit 1.txt -m "3 commit"
Цей коміт фіксує файл 1.txt з повідомленням "3 commit" у гілці branch1.
7. git switch master
Я перемикаюсь на гілку master.
- A master.txt: Після перемикання на гілку, файл master.txt додається на stage.

8. git commit master.txt -m "4 commit"
Цей коміт фіксує файл master.txt з повідомленням "4 commit" у гілці master.

Крок 9. Перегляд журналу комітів у всіх гілках:

1. git log --all --oneline
Ця команда виводить журнал комітів для всіх гілок у стислому форматі. Кожен коміт показаний у вигляді короткого хешу, назви гілки, якщо це актуально, та повідомлення коміту.
- 44a923a (HEAD -> master) 4 commit — Останній коміт у гілці master, в якому зафіксований файл master.txt.
- f91bdf3 (branch1) 3 commit — Останній коміт у гілці branch1, де зафіксований файл 1.txt.
- 75c4040 (branch2) 2 commit — Останній коміт у гілці branch2, де зафіксований файл 2.txt.
- aef759c (branch3) 1 commit — Останній коміт у гілці branch3, де зафіксований файл 3.txt.
- dda0438 empty_commit — Порожній коміт, створений раніше у гілці master.

Крок 10. Додавання та комітування всіх змін у гілці master:

```
C:\Users\Vika\test>git add .

C:\Users\Vika\test>git commit -m "5 commit"
[master a430cb6] 5 commit
 4 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 3.txt

C:\Users\Vika\test>git log master --oneline
a430cb6 (HEAD -> master) 5 commit
44a923a 4 commit
dda0438 empty_commit
```

1. git add .
Додає всі файли та зміни в робочій директорії до індексації (staging area). У цьому випадку додані файли 1.txt, 2.txt, 3.txt та інші зміни у файлі master.txt.
2. git commit -m "5 commit"
Створює новий коміт з повідомленням "5 commit", фіксуючи всі додані файли та зміни в гілці master.
3. git log master --oneline
Виводить стислий журнал комітів для гілки master.

Крок 11. Оновлення гілки branch1 до останньої версії гілки master за допомогою merge:

```
C:\Users\Vika\test>git switch branch1
Switched to branch 'branch1'

C:\Users\Vika\test>git merge master
Auto-merging 1.txt
CONFLICT (add/add): Merge conflict in 1.txt
Automatic merge failed; fix conflicts and then commit the result.

C:\Users\Vika\test>git add 1.txt

C:\Users\Vika\test>git merge --continue
[branch1 ea018eb] Merge branch 'master' into branch1
```

1. git switch branch1
Я перемикаюсь на гілку branch1.
2. git merge master
Ця команда намагається злити зміни з гілки master у поточну гілку branch1. Під час цього процесу Git виявляє конфлікт у файлі text.txt, оскільки в обох

гілках були внесені зміни в один і той же рядок. Git не може автоматично злитись, тому виводить повідомлення про помилку.
3. git add 1.txt
Після вирішення конфлікту я додаю файл 1.txt на stage, вказуючи Git, що конфлікт вирішено.
4. git merge --continue
Ця команда завершує процес злиття після додавання файлу. Коміт створюється з повідомленням, яке автоматично вказує на злиття гілок.

Крок 12: Оновлення гілки branch2 до останньої версії гілки master за допомогою rebase:

```
C:\Users\Vika\test>git switch branch2
Switched to branch 'branch2'

C:\Users\Vika\test>git rebase master
Auto-merging 2.txt
CONFLICT (add/add): Merge conflict in 2.txt
error: could not apply 75c4040... 2 commit
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".
hint: Disable this message with "git config advice.mergeConflict false"
Could not apply 75c4040... 2 commit

C:\Users\Vika\test>git add 2.txt

C:\Users\Vika\test>git rebase --continue
[detached HEAD 1c4efb1] 2 commit
 1 file changed, 3 insertions(+), 1 deletion(-)
Successfully rebased and updated refs/heads/branch2.
```

1. git switch branch2
Я перемикаюсь на гілку branch2.
2. git rebase master
Ця команда намагається перенести (rebase) всі коміти з поточної гілки (в даному випадку branch2) на верхівку гілки master. Під час цього процесу Git виявляє конфлікт у файлі text.txt, оскільки зміни в цьому файлі не можуть бути автоматично об'єднані.
3. git add 2.txt
Після вирішення конфлікту я додаю файл 2.txt на stage, вказуючи Git, що конфлікт вирішено.
4. git rebase --continue
Ця команда продовжує процес rebase, фіксуючи зміни в гілці branch2. У результаті створюється новий коміт з повідомленням "2 commit", і процес rebase успішно завершено. Гілка branch2 оновлена, і зміни з коміту 75c4040 були перенесені на нове місце.

Крок 13: Оновлення гілки branch3 до останньої версії гілки master за допомогою cherry-pick:

```
C:\Users\Vika\test>git switch branch3
Switched to branch 'branch3'

C:\Users\Vika\test>git log master --oneline
a430cb6 (master) 5 commit
44a923a 4 commit
dda0438 empty_commit

C:\Users\Vika\test>git cherry-pick 44a923a a430cb6
[branch3 bdc2bdf] 4 commit
 Date: Sat Oct 12 10:44:12 2024 +0300
 1 file changed, 1 insertion(+)
 create mode 100644 master.txt
Auto-merging 3.txt
CONFLICT (add/add): Merge conflict in 3.txt
error: could not apply a430cb6... 5 commit
hint: After resolving the conflicts, mark them with
hint: "git add/rm <pathspec>", then run
hint: "git cherry-pick --continue".
hint: You can instead skip this commit with "git cherry-pick --skip".
hint: To abort and get back to the state before "git cherry-pick",
hint: run "git cherry-pick --abort".
hint: Disable this message with "git config advice.mergeConflict false"

C:\Users\Vika\test>git add 3.txt

C:\Users\Vika\test>git cherry-pick --continue
[branch3 3528ad5] 5 commit
 Date: Sat Oct 12 10:48:12 2024 +0300
 4 files changed, 5 insertions(+), 1 deletion(-)
 create mode 100644 1.txt
 create mode 100644 2.txt
```

1. git switch branch3
Я перемикаюсь на гілку branch3.
2. git log master --oneline
Ця команда відображає коміти основної гілки (master) в короткому форматі. Я бачу, що останніми комітами є 5 commit, 4 commit, і empty_commit.
3. git cherry-pick 44a923a a430cb6
Тут я намагаюсь перенести два коміти (4 commit і 5 commit) з основної гілки в branch3. Під час виконання цієї команди виникає конфлікт у файлі 3.txt, оскільки в обох гілках внесені зміни.
4. git add 3.txt
Після вирішення конфлікту я додаю файл 3.txt на stage, вказуючи Git, що конфлікт вирішено.
5. git cherry-pick --continue
Ця команда завершує процес cherry-pick, і я отримую новий коміт 5 commit з усіма змінами з обох вибраних комітів. У результаті, я успішно перенесла

коміти з основної гілки до branch3, і тепер у моїй гілці є зміни з 4 commit і 5 commit.

Крок 14. Перегляд журналів комітів та усіх комітів з графічним представленням:

```
C:\Users\Vika\test>git log --oneline
3528ad5 (HEAD -> branch3) 5 commit
bdc2bdf 4 commit
aef759c 1 commit
dda0438 empty_commit

C:\Users\Vika\test>git log branch2 --oneline
1c4efb1 (branch2) 2 commit
a430cb6 (master) 5 commit
44a923a 4 commit
dda0438 empty_commit

C:\Users\Vika\test>git log --all --oneline --graph
* 3528ad5 (HEAD -> branch3) 5 commit
* bdc2bdf 4 commit
* aef759c 1 commit
| * 1c4efb1 (branch2) 2 commit
| | *   ea018eb (branch1) Merge branch 'master' into branch1
| | |\
| | |/
| |/|
| * | a430cb6 (master) 5 commit
| * | 44a923a 4 commit
|/ /
| * f91bdf3 3 commit
|/
* dda0438 empty_commit
```

1. git log --oneline

Ця команда виводить список комітів у гілці branch3 в короткому форматі. Результат показує, що:

- 3528ad5 — це коміт 5 commit, який є останнім у branch3.
- bdc2bdf — це коміт 4 commit, який я перенесла з master за допомогою cherry-pick.
- aef759c — це коміт 1 commit, створений раніше в branch3.
- dda0438 — це коміт empty_commit.

2. git log branch2 --oneline

Ця команда виводить список комітів у гілці branch2. Результат показує, що:

- 1c4efb1 — це коміт 2 commit, останній в branch2.
- a430cb6 — це коміт 5 commit з master, який вже є в branch2.
- 44a923a — це коміт 4 commit, який також з master.

- dda0438 — це коміт empty_commit.

3. git log --all --oneline --graph

Ця команда виводить усі коміти з усіх гілок в графічному вигляді. Графік показує:

- Зірочка (*) означає коміт, а стрілки вказують на зв'язок між комітами.
- Гілка branch3 має коміт 5 commit (3528ad5) на вершині, потім 4 commit (bdc2bdf) та 1 commit (aef759c).
- Гілка branch2 має коміт 2 commit (1c4efb1), який відокремлюється від гілки master.
- Гілка master має коміти 5 commit (a430cb6) та 4 commit (44a923a), а також базу з комітом empty_commit (dda0438).

Висновок: завдяки цій лабораторній роботі, я ознайомилась та навчилась працювати з основними командами Git, вивчила способи створення репозиторіїв, комітів, роботи з гілками, злиття та вирішення конфліктів.