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

Технології розроблення програмного забезпечення

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

Виконала:
студентка групи ІА-32
Чайковська С. В.

Перевірив:
Мягкий М. Ю.

Київ 2025

Зміст

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.

1 Теоретичні відомості

1.1 Призначення систем управління версіями

Система управління версіями (англ. Version Control System, VCS) — це програмне забезпечення, що допомагає розробникам керувати змінами у вихідному коді під час роботи. Вона дозволяє зберігати історію змін, повертатися до попередніх версій, аналізувати внесені зміни та визначати їх авторів. СКВ широко застосовуються при розробці програмного забезпечення, а також у роботі з документами чи проєктами, що постійно змінюються (наприклад, у САПР).

1.2 Історія розвитку систем контролю версій

Розвиток систем контролю версій можна поділити на чотири основні етапи:

Ранній етап

Спочатку розробники копіювали проєкти вручну (наприклад, у вигляді zip-архівів). Це було незручно і не дозволяло відслідковувати зміни. У 1982 році з’явився RCS — система, що зберігала зміни у вигляді дельт (лише змінені рядки). Проте вона працювала тільки з текстовими файлами та не мала централізованого репозиторію.

Централізовані системи

У 1990-х роках з’явилися централізовані СКВ, які дозволили декільком розробникам одночасно працювати з одним сервером. Найпопулярнішою стала система CVS, яка сформувала уявлення про сучасні інструменти контролю версій. Наступником CVS стала SVN — проста та надійна клієнт-серверна система, яка зберігала резервні копії та мала зручні команди (commit, update). Її недоліком було складне злиття гілок.

Етап децентралізації

Децентралізовані системи усунули залежність від центрального сервера. Кожен розробник отримував повну копію репозиторію. Одним із перших рішень був ClearCase (1992), який пропонував розширені функції, включно з динамічними уявленнями файлів. У 2005 році з'явилися дві революційні системи — Git і Mercurial, які зробили роботу значно швидшою та гнучкішою. Вони підтримували розгалуження й злиття, але реалізовували це по-різному. Починаючи з 2018 року, Git став домінуючою безкоштовною системою контролю версій.

Етап хмарних платформ

Приблизно з 2010 року Git інтегрується у хмарні сервіси — GitHub, GitLab, Bitbucket. Такі платформи надають інструменти для командної роботи, CI/CD, DevOps, автоматичного тестування та аналітики. Таким чином, СКВ перетворилися на повноцінні екосистеми для спільної розробки й автоматизації процесів.

1.3 Робота з Git

Git можна використовувати як через командний рядок, так і через графічні оболонки (Git Extensions, SourceTree, GitKraken, GitHub Desktop). Основна ідея Git — кожен розробник має власний локальний репозиторій із повною історією, а синхронізація з іншими виконується через обмін комітами.

1.4 Основні команди Git:

1. git clone — отримати копію репозиторію;
2. git fetch / git pull — отримати зміни з віддаленого репозиторію;
3. git add — додати файли у staging area;
4. git commit — зафіксувати зміни у локальному репозиторії;
5. git push — передати зміни у віддалений репозиторій;
6. git merge — об'єднати гілки.

Типовий робочий процес виглядає так: клонування репозиторію → створення гілки → внесення змін і коміти → синхронізація з віддаленим репозиторієм → об'єднання гілок.

2 Хід роботи

Крок 1. Створення директорії та ініціалізація репозиторію:

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

C:\Users\chaik>cd test

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

1. mkdir test

   Команда створює нову папку *test* у поточному каталозі (C:\Users\chaik). Це підготовка робочого простору для проєкту.

2. cd test

   Перехід у створену директорію, у якій далі буде працювати Git.

3. git init

   Створює новий порожній Git-репозиторій у папці *test*. З'являється прихована директорія *.git*, де зберігатиметься вся історія змін.

Крок 2. Перевірка наявності гілки main:

```
C:\Users\chaik\test>git status
On branch main

No commits yet

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

1. git status

   Показує стан робочої директорії:

   - гілка main активна;
   - комітів ще немає;
   - файли відсутні для відстеження.

Крок 3. Створення порожнього коміту:

```
C:\Users\chaik\test>git commit --allow-empty -m "initial commit"
[main (root-commit) 6498643] initial commit
```

1. git commit --allow-empty -m "initial commit"

   Ця команда створює порожній коміт (без змін у файлах) з повідомленням "initial 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.

Крок 4. Створення нових гілок:

```
C:\Users\chaik\test>git branch test1

C:\Users\chaik\test>git checkout -b test2
Switched to a new branch 'test2'

C:\Users\chaik\test>git branch
  main
  test1
* test2
```

1. git branch test1

   Ця команда створює нову гілку з назвою test1, але не перемикає мене на неї. Гілки дозволяють розвивати проект паралельно, не впливаючи на основну гілку

2. git checkout -b test2

   Ця команда одночасно створює нову гілку test2 і перемикає мене на неї. Тепер я можу вносити зміни в цій гілці без впливу на інші.

3. git branch

   Ця команда виводить список гілок: main, test1, test2. Зірочка біля test2 означає, що вона зараз активна.

Крок 5. Створення нових файлів:

```
C:\Users\chaik\test>echo 1 > f1.txt

C:\Users\chaik\test>echo 2 > f2.txt

C:\Users\chaik\test>git add .

C:\Users\chaik\test>git commit -m "add f1.txt" f1.txt
[test2 de1bd86] add f1.txt
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt

C:\Users\chaik\test>git commit -m "add f2.txt" f2.txt
[test2 890f583] add f2.txt
 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt
```

1. echo 1 > f1.txt / echo 2 > f2.txt

Кожна з цих команд створює нові файли (f1.txt, f2.txt) та записує в них відповідні числа (1, 2). Файли зберігаються в робочій директорії репозиторію.

2. git add .

   Ця команда додає всі нові файли у staging area (область підготовки). Тепер ці файли Git починає відстежувати.

3. git commit -m "add f1.txt" f1.txt / git commit -m "add f2.txt" f2.txt

   Ці команди фіксують зміни у файлах f1.txt та f2.txt з повідомленням "add f1.txt" та "add f2.txt".

Крок 6. Робота в гілці test1:

```
C:\Users\chaik\test>git switch test1
Switched to branch 'test1'

C:\Users\chaik\test>echo b > f1.txt

C:\Users\chaik\test>git add f1.txt

C:\Users\chaik\test>git commit -m "add f1.txt branch test1"
[test1 38f0a7e] add f1.txt branch test1
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt
```

1. git switch test1

   Команда для перемикання на гілку test1.

2. echo b > f1.txt

   За допомогою цієї команди у файлі f1.txt записано новий вміст (“b”).

3. git add f1.txt

   Файл додається у staging area.

4. git commit -m "add f1.txt branch test1"

   Фіксується зміна файлу f1.txt у гілці test1.

Крок 7. Злиття гілок:

```
C:\Users\chaik\test>git merge test2
Auto-merging f1.txt
CONFLICT (add/add): Merge conflict in f1.txt
Automatic merge failed; fix conflicts and then commit the result.

C:\Users\chaik\test>notepad f1.txt

C:\Users\chaik\test>git add f1.txt

C:\Users\chaik\test>git merge --continue
hint: Waiting for your editor to close the file...
[main 2025-09-13T09:02:32.670Z] update#setState idle
[main 2025-09-13T09:02:36.070Z] Extension host with pid 20932 exited with code: 0, signal: unknown.
[test1 414b4d0] Merge branch 'test2' into test1
```

1. git merge test2

   Спроба об’єднати гілку test2 з test1. Git виявляє конфлікт у файлі f1.txt, оскільки в обох гілках були різні зміни цього файлу.

2. notepad f1.txt

   Відкривається файл для редагування. Вирішуємо конфлікт шляхом вибору правильного варіанту вмісту.

3. git add f1.txt

   Файл додається у staging area.

4. git merge --continue

   Ця команда завершує злиття. Створюється коміт із повідомленням "Merge branch 'test2' into test1".

Крок 8. Перегляд журналів коммітів:

```
C:\Users\chaik\test>git log --all --graph
*   commit 414b4d0f144f1aaa02d1829719270a0949d68b47 (HEAD -> test1)
|\  Merge: 38f0a7e 890f583
| | Author: Chaikovska Sofiia <chaikovska.sofiia@lll.kpi.ua>
| | Date:   Sat Sep 13 12:02:31 2025 +0300
| |
| |     Merge branch 'test2' into test1
| |
| * commit 890f58377ef5c4eff038905847e09d83f012ca9f (test2)
| | Author: Chaikovska Sofiia <chaikovska.sofiia@lll.kpi.ua>
| | Date:   Sat Sep 13 11:57:22 2025 +0300
| |
| |     add f2.txt
| |
| * commit de1bd863b87b9b58e11e68381af691445f1626c6
| | Author: Chaikovska Sofiia <chaikovska.sofiia@lll.kpi.ua>
| | Date:   Sat Sep 13 11:57:09 2025 +0300
| |
| |     add f1.txt
| |
* | commit 38f0a7e270d2d96f4a7a6cf8cb58336bda1d30ce
|/  Author: Chaikovska Sofiia <chaikovska.sofiia@lll.kpi.ua>
|   Date:   Sat Sep 13 11:59:10 2025 +0300
|
|       add f1.txt branch test1
|
* commit 64986434bd358c4186dcf132c1b5471d6ba910f5 (main)
  Author: Chaikovska Sofiia <chaikovska.sofiia@lll.kpi.ua>
  Date:   Sat Sep 13 11:55:14 2025 +0300
```

1. git log --all --graph

   Виводить історію комітів з графічним представленням:

   - видно розгалуження *test1* і *test2*;

- відображається початковий коміт, окремі коміти для файлів *f1.txt* і *f2.txt*;
- видно коміт злиття з вирішенням конфлікту.

3 Висновки

У процесі виконання лабораторної роботи ми ознайомилися з основними командами Git, навчилися створювати репозиторій, коміти, працювати з гілками, виконувати злиття та вирішувати конфлікти. Практично відпрацювали робочий процес з Git, включно з переглядом історії комітів.

4 Відповіді на питання

1. Що таке система контролю версій (СКВ)?

Це програмне забезпечення, яке відслідковує зміни у файлах, зберігає історію та дозволяє співпрацювати над проектами.

2. Відмінності між розподіленою та централізованою СКВ.

У централізованій є один центральний сервер; у розподіленій кожен має повну копію репозиторію.

3. Різниця між stage та commit в Git.

Stage — підготовлена область для змін; commit — фіксація цих змін у репозиторії.

4. Як створити гілку в Git?

Командою git branch branch_name або git switch -c branch_name, або git ckeckout -b branch_name

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

Використати git clone <url>.

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

Конфлікт виникає, коли в різних гілках зміни торкаються одного й того ж рядка. Створюється шляхом редагування файлу у двох гілках. Вирішується вручну редагуванням і git add, git merge --continue.

7. Коли використовуються merge, rebase, cherry-pick?

- merge — об’єднання двох гілок в одну;
- rebase — переписує історію гілки так, ніби всі її коміти були зроблені поверх іншої гілки;
- cherry-pick — вибіркове перенесення окремих комітів з однієї гілки в іншу.

8. Як переглянути історію змін Git репозиторію в консолі?

Командою git log.

9. Як створити гілку в Git без команди git branch?

За допомогою git checkout -b branch_name або git switch -c branch_name.

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

git add .

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

git add <folder_name>/

12. Як переглянути перелік гілок у репозиторії?

git branch

13. Як видалити гілку?

git branch -d <name> (якщо злиття виконано), або git branch -D <name> (примусово).

14. Способи створення гілки та відмінності між ними.

- git branch <name> — створює гілку без перемикання;
- git checkout -b <name> або git switch -c <name> — створює та одразу перемикає.