Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №1
з дисципліни «Технології розроблення програмного забезпечення»
Тема: «Системи контроля версій. Розподілена система контролю версій Git»

Виконав:
студент групи ІА-34
Тунік О.

Перевірив:
асистент кафедри ІСТ
Мягкий М.Ю.

Київ 2025

Зміст

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

Теоретичні відомості

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 мати 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. Однак вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю електронних документів, що безперервно змінюються.

Історія розвитку

Перший етап розвитку систем контролю версій охоплював роботу з окремими файлами в локальному середовищі. На початку використовували лише копіювання проєктів у різні папки чи архіви, що фактично не було повноцінним контролем версій. У 1982 році з’явилася RCS, яка ввела ідею збереження змін у вигляді дельт, однак вона працювала тільки з текстовими файлами та не мала централізованого репозиторію.

У 90-х роках почалася епоха централізованих систем, де ключову роль відіграли CVS та SVN. Вони забезпечили появу центральних репозиторіїв і дали можливість командної роботи через сервер. CVS започаткувала багато ідей, а SVN удосконалив підхід, зробивши систему надійнішою та простішою у використанні. Проте обмеженість у роботі з гілками та конфліктами згодом стала причиною переходу до децентралізованих рішень.

Наступним етапом стали розподілені системи, серед яких виділилися ClearCase, Git та Mercurial. Вони надали кожному розробнику повну копію репозиторію та дозволили працювати автономно. Git, створений Лінусом Торвальдсом у 2005 році, став ключовим інструментом завдяки швидкості, надійності та зручному управлінню гілками. Починаючи з 2010-х, розвиток перейшов у площину хмарних платформ, де GitHub, GitLab і Bitbucket інтегрували контроль версій із CI/CD, DevOps-практиками та інструментами для глобальної співпраці.

Робота з Git

Робота з Git може виконуватися як через командний рядок, так і за допомогою візуальних оболонок. Командний рядок надає повний набір команд і дозволяє складати складні сценарії, тому ним найчастіше користуються програмісти. Графічні оболонки, такі як Git Extensions, SourceTree, GitKraken чи GitHub Desktop, спрощують роботу завдяки наочному відображенню гілок і змін, однак зазвичай підтримують лише найпоширеніші операції.

Основна ідея Git полягає в тому, що кожен розробник має власний локальний репозиторій, у який фіксує свої зміни, а потім синхронізує його з іншими за допомогою віддаленого репозиторію. Для цього передбачено набір базових команд: git clone для отримання копії проєкту, git commit для фіксації змін, git fetch і git pull для завантаження нових версій, git push для відправки власних змін, а також git merge для злиття гілок. Цей інструментарій дозволяє забезпечити ефективну співпрацю між членами команди.

Типовий робочий процес у Git виглядає так: розробник спершу клонує репозиторій, створює нову гілку під задачу і вносить у неї зміни. У процесі роботи він періодично синхронізує локальні дані з центральним репозиторієм, щоб уникати конфліктів. Після завершення задачі відбувається злиття робочої гілки з основною та фінальна синхронізація з віддаленим репозиторієм, що дозволяє інтегрувати зміни в проєкт і зробити їх доступними для всієї команди.

Хід роботи

1. Ініціалізуємо репозиторій:

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_
technologies
$ mkdir gitTest

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies
$ cd gitTest/

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest
$ git init
Initialized empty Git repository in D:/Smegal/Documents/Universities/KPI/Software_dev
_technologies/gitTest/.git/
```

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (master)
$ git commit -m "init" --allow-empty
[master (root-commit) 6ca89de] init
```

2. Створюємо 2 додаткові гілки різними способами, щоб в сумі було 3:

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (master)
$ git branch br1

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (master)
$ git checkout -b br2
Switched to a new branch 'br2'

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br2)
$ git branch
  br1
* br2
  master
```

3. Додаємо файл, фіксуємо різне його наповнення на різних гілках, щоб симулювати конфлікт при майбутньому об’єднанні:

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br2)
$ echo "dfgssdhf" > f2.txt

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br2)
$ git add .
warning: in the working copy of 'f2.txt', LF will be replaced by CRLF the next time G
it touches it

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br2)
$ git commit -m "f2 create"
[br2 7c6c7a3] f2 create
 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br2)
$ git checkout br1
Switched to branch 'br1'

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ echo "111111" > f2.txt
```

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ git add .
warning: in the working copy of 'f2.txt', LF will be replaced by CRLF the next time G
it touches it

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ git commit -m "f2 create"
[br1 3bea909] f2 create
 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt
```

4. Здійснюємо злиття зі стисненням комітів в один та вирішенням конфліктів:

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ git merge --squash br2
Auto-merging f2.txt
CONFLICT (add/add): Merge conflict in f2.txt
Squash commit -- not updating HEAD
Automatic merge failed; fix conflicts and then commit the result.

smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ vi f2.txt
```

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ git add .
```

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ git commit -m "squash"
[br1 346c448] squash
 1 file changed, 1 insertion(+), 1 deletion(-)
```

5. Переглядаємо історію репозиторію, щоб побачити результати наших дій:

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_techn
ologies/gitTest (br1)
$ git log --all --graph
* commit 346c4484984c3b808f0948b6d1297f06b7a1de16 (HEAD -> br1)
| Author: Oleksandr Tunik <70912192+Smegalex@users.noreply.github.com>
| Date:   Sat Sep 20 12:41:11 2025 +0100
|
|     squash
|
* commit 3bea909060a0f331911c62bf2965d9ca0cd4f41b
| Author: Oleksandr Tunik <70912192+Smegalex@users.noreply.github.com>
| Date:   Sat Sep 20 12:37:54 2025 +0100
|
|     f2 create
|
| * commit 7c6c7a365095fe9c3fce4861b07af93550736913 (br2)
|/  Author: Oleksandr Tunik <70912192+Smegalex@users.noreply.github.com>
|   Date:   Sat Sep 20 12:37:15 2025 +0100
|
|       f2 create
|
* commit 6ca89de6c05b4cc52929d651c5f90ce3c5acb40a (master)
  Author: Oleksandr Tunik <70912192+Smegalex@users.noreply.github.com>
  Date:   Sat Sep 20 12:35:33 2025 +0100

      init
```

Додаткове завдання

Повторимо кроки 1-3, але далі, замість злиття зі стисненням комітів в один, об’єднаємо гілки з вирішенням конфліктів і без збереження історії однієї з них, використувавши перебазування.

4. Робимо перебазування, вирішуємо конфлікт:

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_
technologies/gitTest (br2)
$ git rebase br1
Auto-merging f2.txt
CONFLICT (add/add): Merge conflict in f2.txt
error: could not apply e6a3915... f2 create
```

```
vi f2.txt
```

```
git add .
git rebase --continue
```

5. Переглядаємо історію репозиторію, щоб побачити результати наших дій:

```
smega@SilmerStarBeast MINGW64 /d/Smegal/Documents/Universities/KPI/Software_dev_
technologies/gitTest (br2)
$ git log --all --graph
* commit 068ea2ae734a65679cba8d84a6c0f55adfea469b (HEAD -> br2)
| Author: Oleksandr Tunik <70912192+Smegalex@users.noreply.github.com>
| Date:   Sat Sep 20 20:49:32 2025 +0100
|
|     f2 create
|
* commit 8d1a3c1bcc43a9a4623da7baa41afe233b6759ca (br1)
| Author: Oleksandr Tunik <70912192+Smegalex@users.noreply.github.com>
| Date:   Sat Sep 20 20:50:21 2025 +0100
|
|     f2 create on br1
|
* commit 346f2f9002498a7f6c3d0fc05bf15f7d72d15481 (master)
  Author: Oleksandr Tunik <70912192+Smegalex@users.noreply.github.com>
  Date:   Sat Sep 20 20:48:51 2025 +0100

      init
```

Висновки

У ході виконання лабораторної роботи дослідили особливості використання команд git merge, git merge --squash та git rebase. Створили 2 додаткові гілки, свідомо зафіксували на них файли що призведуть до конфлікту при майбутніх об’єднаннях. Об’єднали гілки 2 різними способами, вирішили конфлікти, проаналізували результати.

У результаті роботи отримали практичний досвід роботи з різними стратегіями об’єднання гілок, закріпили навички використання Git, на практиці їх відпрацювали.