

Міністерство освіти і науки України
Національний університет «Запорізька Політехніка»

Кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 2

з дисципліни «Soft skills, групова динаміка та комунікації» на тему:
«ГРУПОВА РОБОТА З СИСТЕМАМИ КЕРУВАННЯ ВЕРСІЯМИ»

Виконал(а):

студент групи КНТ-113сп

І. А. Щедровський

Прийняв(а)

асистент:

А. В. Бєлова (Д. А. Каврін)

2022

1 ГРУПОВА РОБОТА З СИСТЕМАМИ КЕРУВАННЯ ВЕРСІЯМИ

1.1 Мета роботи

1.1.1 Вивчити основні можливості систем керування версіями на прикладі системи Subversion.

1.1.2 Вивчити основні можливості системи керування версіями Git та порівняти їх з можливостями Subversion.

1.1.3 Навчитися використовувати можливості систем керування версіями Subversion та Git для групової роботи.

1.2 Завдання роботи

1.2.1 Ознайомитися з теоретичними відомостями, необхідними для виконання роботи.

1.2.2 Ознайомитися з основними можливостями систем керування версіями Subversion та Git.

1.2.3 Використовуючи систему керування версіями Subversion (для пунктів 1.2.3-1.2.15), створити каталог проекту та базову структуру репозиторію.

1.2.4 Створити файли проекту у каталозі. Файли проекту можуть містити код проекту або бути текстовими файлами.

1.2.5 Відредагувати файли проекту і зберегти версії.

1.2.6 Створити нову гілку та переключити робочу копію на неї. 1.2.7 Відредагувати файли проекту.

1.2.8 Переключити робочу копію на піддиректорію trunk. 1.2.9 Злити зміни між гілками.

1.2.10 Вилучити гілку.

1.2.11 Додати до репозиторію проект на основі даних одного з проектів з відкритим вихідним кодом.

1.2.12 В якості такого проекту можна використати, наприклад, Notepad++ (svn://svn.code.sf.net/p/notepad-plus/code/trunk notepad-plus code) або будь-який власний.

- 1.2.13 Внести зміни в файли проекту.
- 1.2.14 Відправити зміни до репозиторію та пояснити отримані результати.
- 1.2.15 Пункти завдань 1.2.6–1.2.14 виконати за допомогою клієнту та командного рядка.
- 1.2.16 Використовуючи систему керування версіями Git (для пунктів 1.2.16–1.2.25), створити репозиторій на основі одного з раніше розроблених проектів.
- 1.2.17 Заборонити автоматичне додання в репозиторій файлів з розширенням *.exe.
- 1.2.18 Внести зміни в текст проекту та зафіксувати їх, для підпису використовуючи власні дані.
- 1.2.19 Переглянути різницю між новою версією проекту та початковою.
- 1.2.20 Створити власне віддалене сховище, використовуючи сервіс GitHub.
- 1.2.21 Налаштувати локальне сховище для синхронізації з віддаленим та відправити локальну версію на сервер.
- 1.2.22 Переглянути історію проекту та сторінку проекту через web-інтерфейс.
- 1.2.23 Внести зміни в текст проекту та зафіксувати їх.
- 1.2.24 Узгодити локальну версію репозиторію з сервером.
- 1.2.25 Пункти завдань 1.2.17–2.3.25 виконати за допомогою графічної оболонки та командного рядка.
- 1.2.26 Оформити звіт з роботи.
- 1.2.27 Відповісти на контрольні питання.

1.3 Короткі теоретичні відомості

Системи керування версіями надають важливий інструментарій для роботи з файлами, що є корисним зокрема для командної роботи. Система контролю версій фактично контролює процес створення версій файлу. За допомогою такої системи у підсумку можна зокрема повертатися до попередніх версій файлу, фіксувати те, які саме зміни відбулись і відповідно визначати, яка саме версія

файлу потрібна користувачеві. дозволяє відслідковувати зміни, які вносять різні користувачі у відповідні файли. Даний тип систем дуже важливий стосовно застосування для розробки програмного забезпечення, надаючи таким чином можливість зберігати і контролювати вихідні коди програми. Проте тільки таким типом даних використання даних систем не обмежується, фактично розширюючи його до будь-яких типів файлів.

Репозиторій (сховище) – місце, де зберігаються та підтримуються будь-які дані, при чому зберігаються як самі файли, так і історія змін цих файлів у відповідності з особливостями конкретної системи контролю версій [2]. Дані у репозиторії знаходяться у вигляді дерева файлової системи.

1.4 Копії екранних форм з результатами виконання завдання та тексти файлів у декількох ревізіях

Subversion

Для роботи з subversion спочатку потрібно створити новий репозиторій. Це показано на рисунку 1.1.

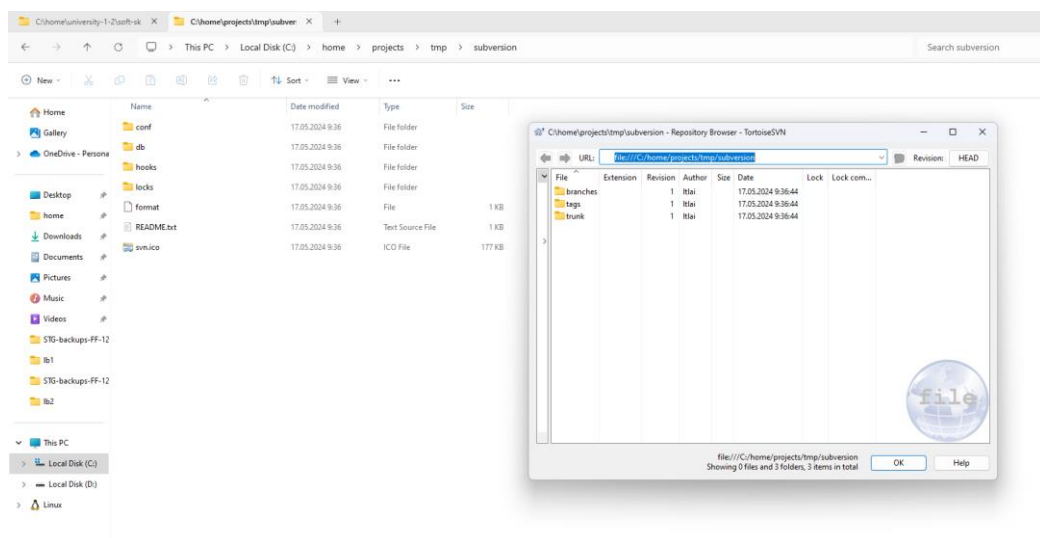


Рисунок 1.1 – Створення нового репозиторію в Subversion

Створення копії репозиторію через checkout, це показано на рисунку 1.2.

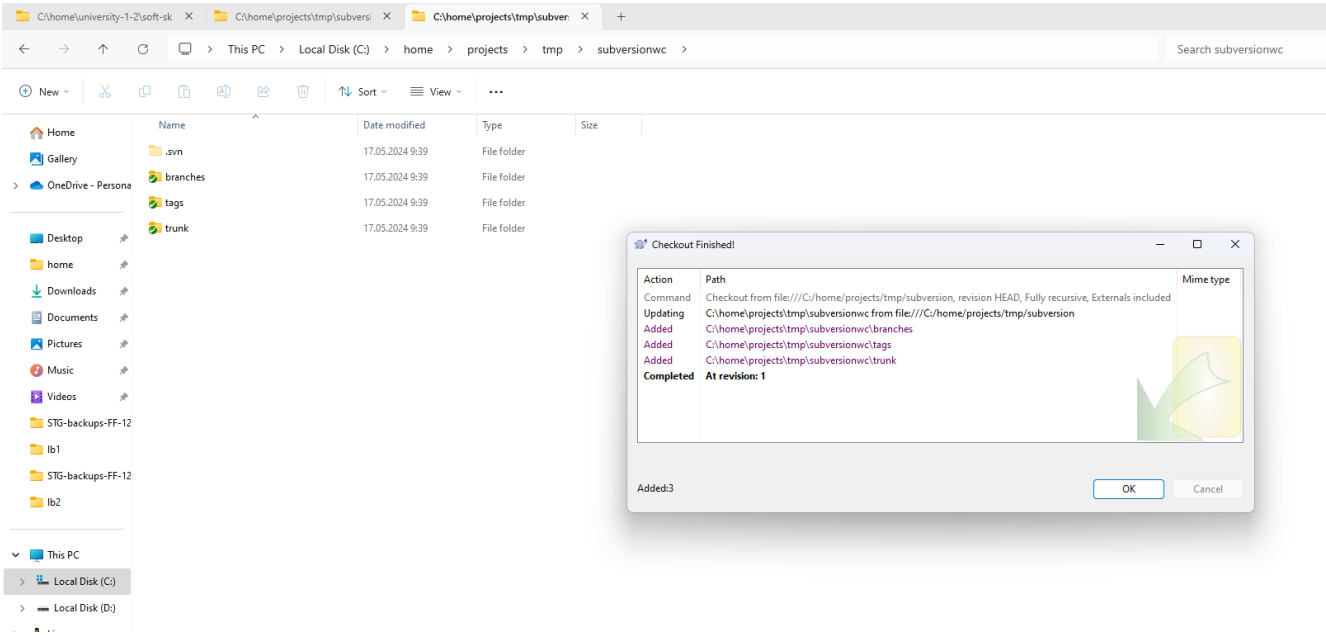


Рисунок 1.2 – Створення робочої копії репозиторію через checkout

Далі я створив два файли «a» та «b» та зробив комміт. Це показано на рисунках 1.3 та 1.4.

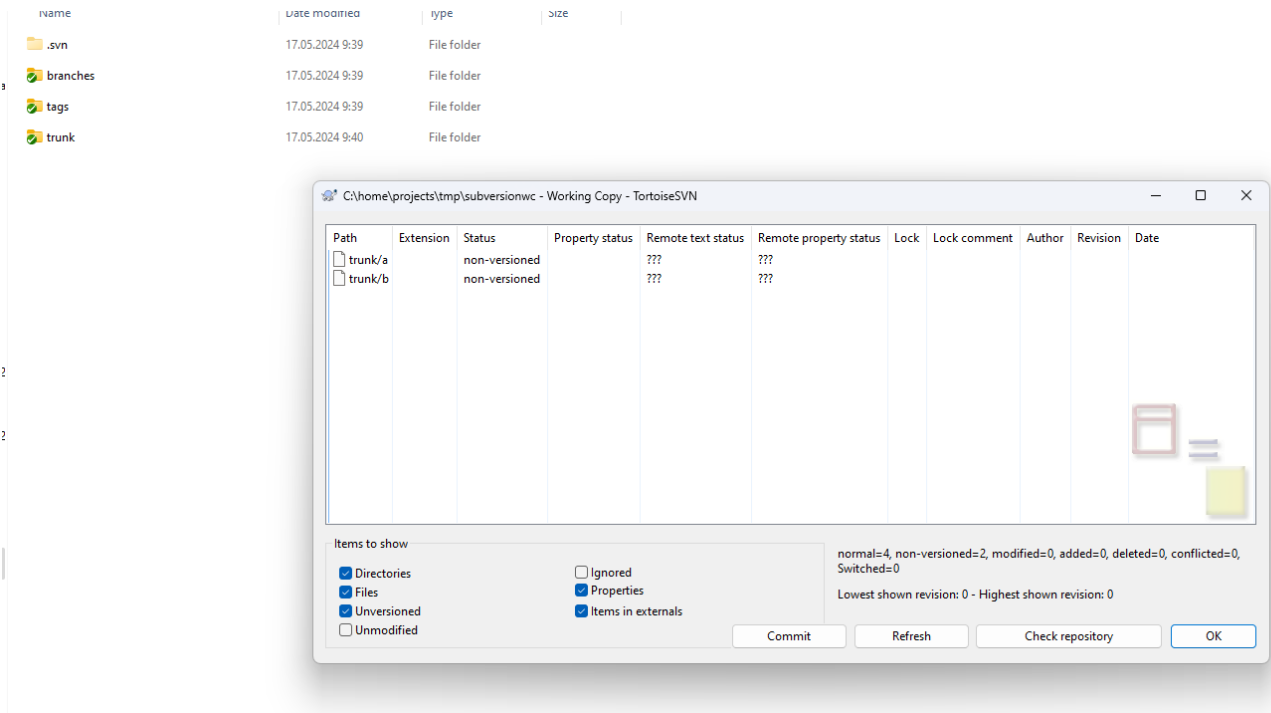


Рисунок 1.3 - Створення файлів a та b

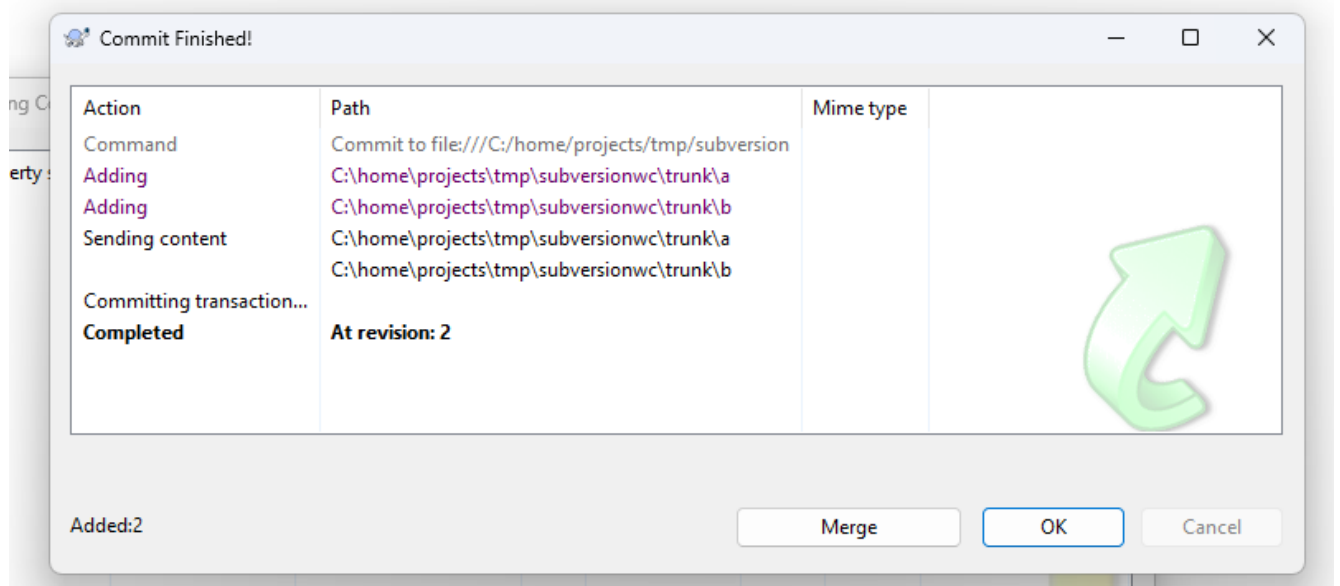


Рисунок 1.4 – Комміт файлів а та б

Далі я створив гілку first-branch, створив на ній файл «с» та зробив комміт. Це показано на рисунках 1.5 та 1.6

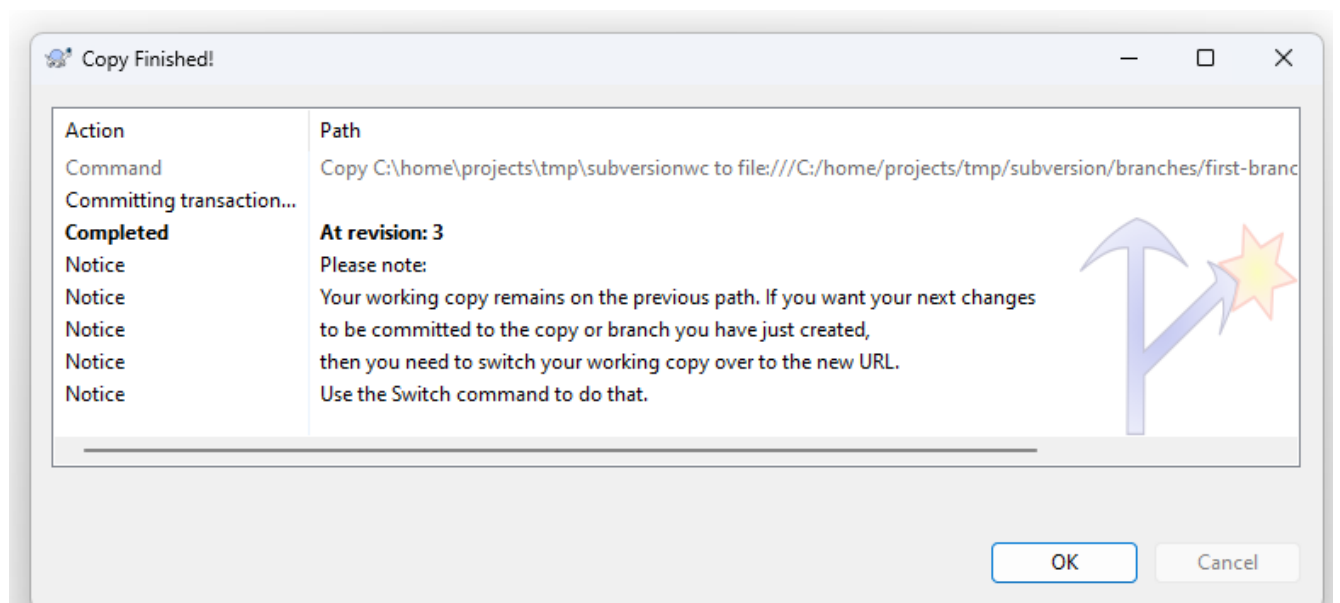


Рисунок 1.5 – Створення гілки first-branch

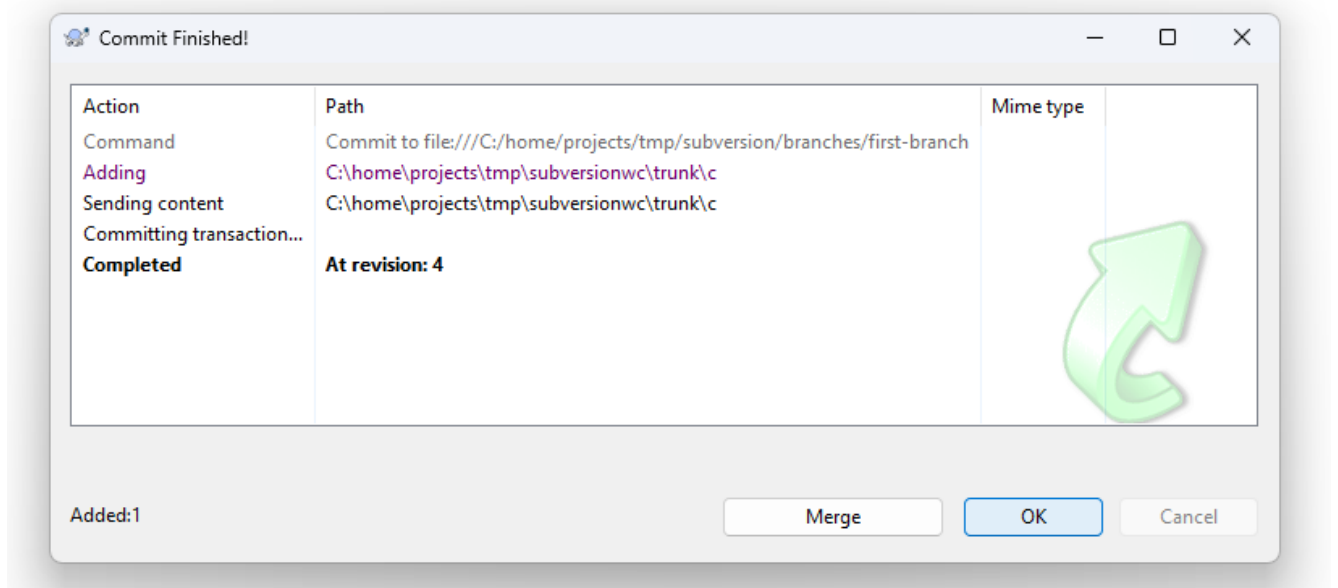


Рисунок 1.6 – Створення файлу с в гілці first-branch

Далі був виконаний merge з `first-branch` в `thunk`, це показано на рисунку 1.7

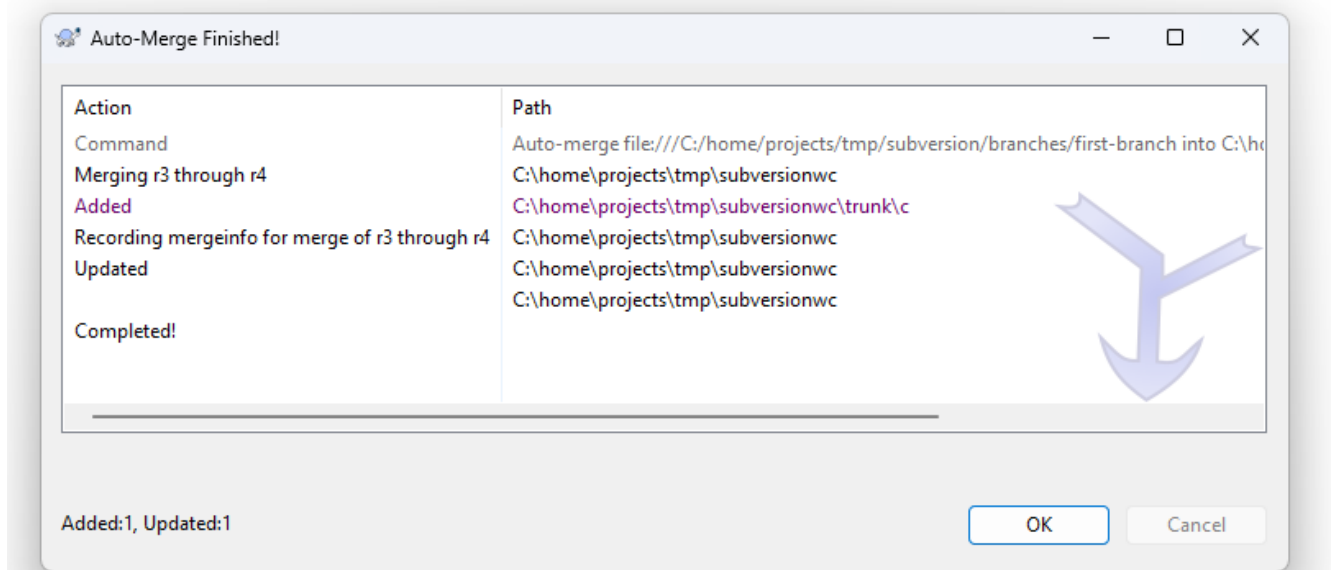


Рисунок 1.7 – Merge first-branch гілки в thunk

На рисунку 1.8 показаний загальний лог змін







Revision	Actions	Author	Date	Message
6		Itlai	17 травня 2024 р. 10:07:23	delete first-branch
5		Itlai	17 травня 2024 р. 10:07:05	merge first branch to thunk
4		Itlai	17 травня 2024 р. 10:04:10	tmp: create C on first branch
3		Itlai	17 травня 2024 р. 10:02:51	create first branch
2		Itlai	17 травня 2024 р. 10:02:06	tmp: A and B
1		Itlai	17 травня 2024 р. 10:01:28	Imported folder structure

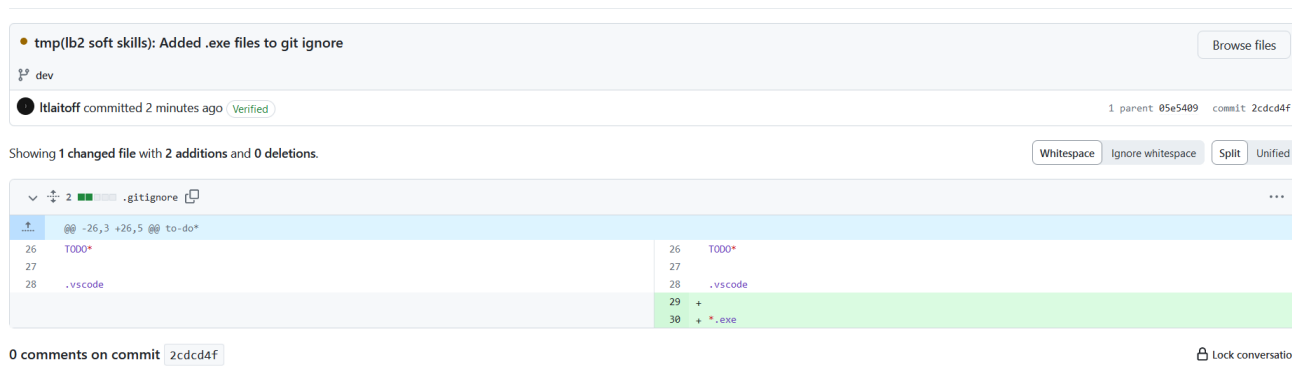
Рисунок 1.8 – Повний лог змін

Git

Для виконання завдання з git я взяв свій досить старий проєкт – hacker-news

Щоб заборонити .exe файли в git repository достатньо в файл `.gitignore` додати `*.exe`. Це показано на рисунку 1.1

Commit



The screenshot shows a commit interface for a repository named 'dev'. The commit message is 'tmp(lb2 soft skills): Added .exe files to git ignore'. The commit is by 'Itlaitoff' and is verified. The commit hash is '2cdcd4f'. The interface shows a diff for the file '.gitignore'. The diff shows two additions: 'T000*' and '*.exe'. The file '.vscode' is also shown. The interface includes buttons for 'Browse files', 'Whitespace', 'Ignore whitespace', 'Split', and 'Unified'. There are 0 comments on the commit.

Рисунок 1.1 – Додавання *.exe до gitignore

Для створення нових змін ми можемо використати:

- `git status` - Для перегляду work tree та staging area
- `git commit` - Для створення комітів
- `git push` - Для відправки змін в основний репозиторій

Звичайно, що це дуже базовий перелік команд і кожна з цих команд має багато параметрів які також можна використовувати

Наприклад, ми можемо підписувати коміти через gpg ключ, щоб потім можна було перевірити хто саме зробив цей комміт, ви або хтось, хто видає себе за вас

На рисунку 1.2 показано застосування команд для створення коміту

```
ltlai@ltlaitoff-win MINGW64 /c/home/projects
$ cd hacker-news/

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$ git s^C

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$ nvim .gitignore

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$ git s
On branch dev
Your branch is up to date with 'origin/dev'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$ git add .

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$ git s
On branch dev
Your branch is up to date with 'origin/dev'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$ git commit -S -m"tmp(lb2 soft skills): Added .exe files to git ignore"
[dev 2cdcd4f] tmp(lb2 soft skills): Added .exe files to git ignore
1 file changed, 2 insertions(+)

ltlai@ltlaitoff-win MINGW64 /c/home/projects/hacker-news (dev)
$ git p
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
```

Рисунок 1.2 – Створення коміту

1.5 Висновки

В ході виконання лабораторної роботи були досліджені основні можливості систем керування версіями на прикладі Subversion та Git, а також проведено їх порівняння.

Subversion (SVN):

- Центральний репозиторій, де зберігаються всі версії файлів.
- Проста структура команд для роботи з версіями.
- Підтримка атомарних комітів.
- Можливість блокування файлів для запобігання конфліктів.
- Підтримка різноманітних гачків (hooks) для автоматизації дій.

Git:

- Розподілена система керування версіями, де кожен користувач має повну копію історії репозиторію.
- Висока швидкість виконання операцій завдяки локальному збереженню даних.
- Підтримка створення гілок і злиття (branching and merging) з мінімальними витратами.
- Підтримка інструментів для інтеграції з CI/CD системами.
- Гнучка система команд для управління версіями та обробки конфліктів.

В процесі роботи було встановлено, що Git надає більше можливостей для гнучкого управління версіями та забезпечує ефективну підтримку розподіленої роботи в командах, в той час як Subversion краще підходить для централізованого управління проектами.

Навчання використанню систем керування версіями Subversion та Git дозволило отримати практичні навички роботи з ними, що є важливим для організації ефективної групової роботи та управління проектами в різних умовах. В цілому, Git та Subversion мають свої переваги та недоліки, і вибір між ними залежить від конкретних потреб проекту та вимог команди.

Контрольні запитання:

2.5.22 Яким чином виконати порівняння комітів та що представляє собою результат?

Для порівняння комітів у Git можна використовувати команду `git diff`. Ця команда показує різницю між двома комітами або між комітом і робочим каталогом. Результатом є набір змін, які відбулися між вибраними комітами або між комітом і поточним станом файлів.

2.5.23 Для чого виконується злиття гілки та яка команда використовується для цього?

Злиття гілок в Git виконується для об'єднання змін з однієї гілки в іншу. Це може бути корисно, коли розробляються функції відокремлено від основної розробки. Команда для злиття гілок - `git merge`. Вона дозволяє об'єднати зміни з однієї гілки в іншу, створюючи новий коміт, який містить зміни з обох гілок.

2.5.24 Яким чином відбувається розв'язання конфліктів у Git?

Конфлікти в Git виникають тоді, коли система не може автоматично об'єднати зміни під час злиття гілок. Розв'язання конфліктів включає в себе ручне вирішення суперечок у файлі або файлах. Це можна зробити за допомогою текстового редактора або спеціальних інструментів для вирішення конфліктів, після чого файли позначаються як вирішені, і злиття може бути завершено за допомогою команди `git merge --continue`.