Лабораторна робота №6
Знайомство з системою контролю версій git

I. Підготовка до лабораторної роботи

Ознайомитись з теоретичною та практичною частиною наведеною нижче для виконання даної лабораторної роботи

II. Теоретичні відомості

1. Про контроль версій

Що таке контроль версій, та навіщо він вам потрібен? Контроль версій - це система, котра веде протокол змін в файлі або кількох файлах увесь час для того, щоб ви мали можливість викликати певні версії пізніше.

Якщо ви графічний або веб-дизайнер і ви хочете зберегти кожну версію малюнку або шару, то використання системи контролю версій (надалі - СКВ) буде дуже розсудливим в цьому випадку. СКВ надає можливість робити наступне: повертати файли до попереднього вигляду, повертати цілий проект до попереднього вигляду, переглядати зміни, зроблені за увесь час, дивитися, хто востаннє змінював щось, що могло призвести до проблеми, хто вирішив задачу та коли й багато іншого. Використання СКВ також означає що, якщо ви щось зламаєте або загубите файли, ви зможете легко усе відновити. До того ж, ви все це отримуєте за дуже маленькі накладні витрати.

Якщо декілька розробників працюють над один й тим самим додатком, іноді одним і тим самим файлом або навіть у суміжних строках, Git дозволяє фіксувати зміни паралельно за допомогою декілька бранчей (branch), а потім зливати усі зміні в один бранч, наприклад основну кодову базу. Альтернативою без контролю версій було би збереження змін у кожного розробника локально, а потім змішування змін вручну на девайсі одного із них. Окремою перевагою є можливість порівняти зміни за допомогою утиліти diff (або git diff), що дозволяє бачити лише зміни без порівняння усіх файлів.

Уміння писати грамотний код – це лише частина роботи програміста. Йому також необхідно вміти використовувати різні інструменти, що дозволяють оптимізувати, полегшити роботу. Розробка програмного забезпечення (ПЗ) ведеться групами розробників, кожен з яких може як створювати свої файли з вихідним кодом, так і змінювати створені іншими файли. Для контролю змін вихідних файлів, зберігання різних версій ПЗ розробники застосовують системи контролю версіями (СКВ). Основними завданнями, які виконують СКВ, є:

- зберігання файлів в репозиторії;
- підтримка перевірки файлів в репозиторії;
- створення різних варіантів одного документа, так званої гілки, із загальною історією змін до точки розгалуження і з різними – після неї;
- знаходження конфліктів при зміні вихідного коду і забезпечення си- нхронізації при роботі в середовищі з багатьма користувачами розробки.
- відстеження авторів змін;
- надання інформації про те, хто і коли додав або змінив конкретний набір рядків у файлі;
- ведення журналу змін, в який користувачі можуть записувати пояснення про те, що і чому вони змінили в цій версії;
- контроль прав доступу користувачів, дозволяючи або забороняючи читання або зміну даних, залежно від того, хто запитує цю дію.

Локальна система контролю версій

Багато хто в якості методу контролю версій вибирає просте копіювання файлів в іншу директорію. Такий підхід дуже розповсюджений, тому що він такий простий, але він, як правило, й призводить до повного краху. Це так легко забути, що ви не в тій директорії, та випадково записати не той файл, або перезаписати не ті файли, котрі хотіли. Для вирішення цієї задачі програмісти вже давно

розробили локальні СКВ, котрі мають просту базу даних для збереження усіх змін файлів, котрі знаходяться під контролем системи (див. рис. 1.1).

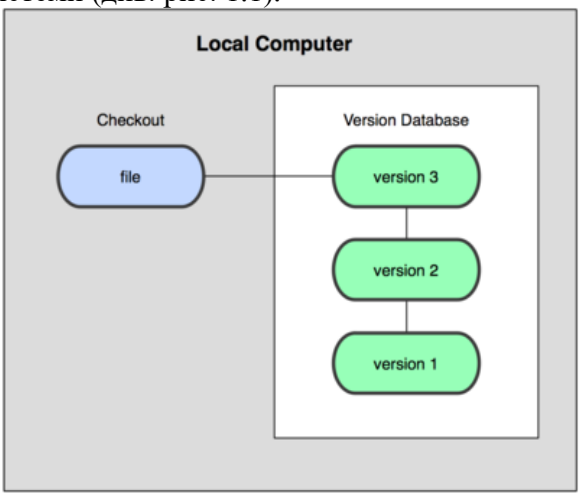


Рисунок 1.1 — Схема локальної СКВ

Одним з найбільш популярних інструментів СКВ є система rcs, котра досі поширена на багатьох ПК нині. Навіть популярна операційна система Mac OS X має утіліту rcs, котра встановлюється разом з Developer Tools. Робота цього інструменту базується на збереженні наборів патчів (це різниця між файлами) від одніє ревізії до іншої в спеціальному форматі на диску; це дає можливість потім відтворити вигляд будь-якого файлу в будь-який момент часу, накладаючи патчі.

Централізовані системи контролю версій

Наступна важлива проблема полягала в потребі співпрацювати з розробниками на інших машинах. Для вирішення цієї проблеми були розроблені централізовані системи контролю версій (ЦСКВ). Ці системи, такі як CVS, Subversion та Perforce, мають єдиний сервер, котрий містить усі версії файлів, а клієнти отримують копії файлів з цього центрального місця. Протягом багатьох років це було стандартом для СКВ. (див. рис. 1.2).

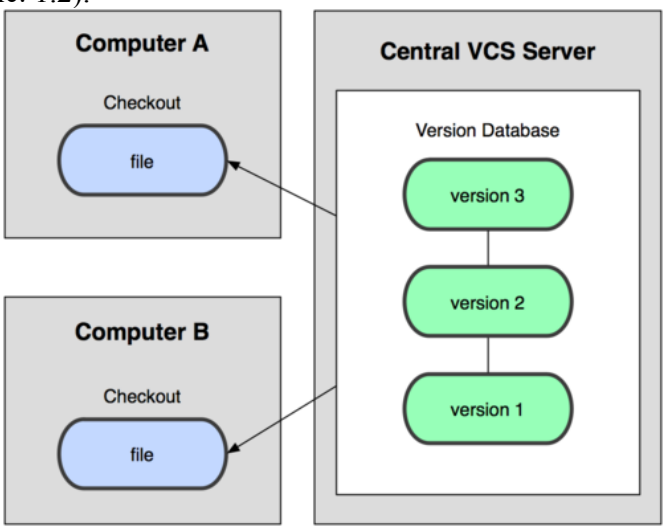


Рисунок 1.2 — Схема централізованого СКВ

Такі системи пропонують багато переваг, особливо над локальними СКВ. Для прикладу, кожен має постійний доступ до інформації: хто що зробив у проекті. Адміністратори мають добре розгалужений контроль над тим, хто що має може робити; і адміністрування ЦСКВ більш легке, ніж надання прав доступу до локальної бази данних кожному користувачу.

Однак, такий підхід також має деякі серйозні недоліки. Найбільш очевидний: централізований сервер є місце вразливості системи. Якщо цей сервер вийде з ладу на годину, то протягом цієї години

ніхто не зможе співпрацювати з іншими або зберегти зміни у файлах нової версії до будь-якого проекту, над котрим працює. Якщо жорсткий диск з централізованою базою даних вийде з ладу та не буде збережено резервної копії, ви втратите абсолютно усе - повну історію проекту, за виключенням тих окремих знімків, котрі збереглися на машинах користувачів. Локальні СКВ також схильні до тієї ж проблеми: коли повна історія проекту зберігається в одному місці, існує ризик втратити усе.

Розподілені системи контролю версій

І тут настає черга розподілених систем контролю версій (РСКВ). В РСКВ (таких як Git, Mercurial, Bazaar або Darcs) клієнт не лише отримує останній знімок файлів, а й повне дзеркало репозиторія. В цьому випадку, якщо будь-який сервер "помре" та інші системи, котрі працювали з ним, любий клієнтський репозиторій може бути використаний для відновлення його копії на сервері. Кожен знімок є насправді цілою копією усіх даних (див. рис. 1.3).

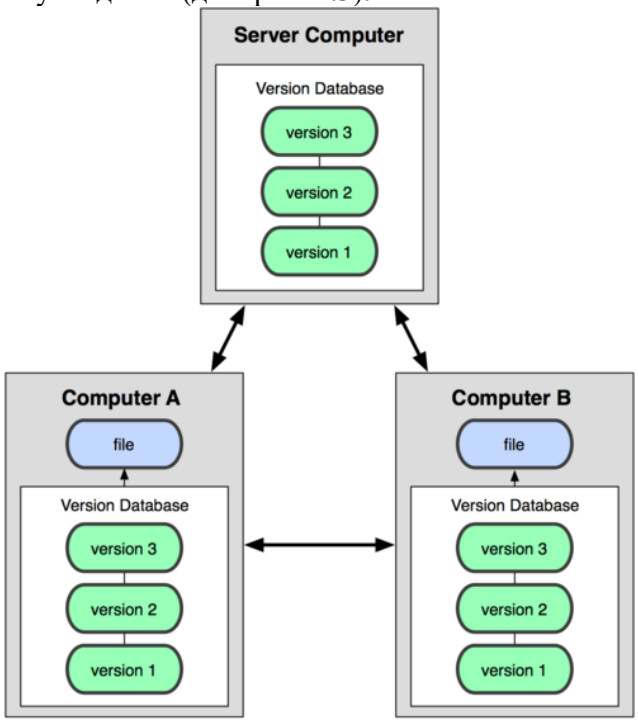


Рисунок 1.3 — Схема розподілених СКВ

Крім того, багато цих систем дуже добре надають доступ до декількох існуючих віддалених репозиторіїв, з котрими можна працювати, таким чином можна одночасно співпрацювати з різними групами розробників в різний спосіб над одним й тим же проектом. Це дає можливість використати декілька типів робочих проектів, що неможливо в ЦСКВ як в ієрархічній моделі.

2. Системи контролю версій на прикладі git

Однією з найбільш популярних СКВ, особливо серед розробників Open-source проектів, на сьогодні є Git – розподілена система керування версіями файлів. Проект був створений Лінусом Торвальдсом для управління розробкою ядра Linux, перша версія випущена 7 квітня 2005 року.

Структура

Git не є централізованим сервером як файлові сервера. Замість цього він є розподіленим (distributed), тобто він може зберігатися одночасно локально та на серверах сервісів таких як GitHub, BitBucket, GitLab тощо. Виглядає він як звичайна директорія з файлам, за виключенням наявності директорії “.git”, що зберігає зміни за допомогою різниць тобто diff-ів. Тобто якщо в історії змін є великий файл, то Git не зберігає його копії, а тільки різницю між його станом до й після зміни. Ця директорія називається репозиторій.

Після деяких змін стан файлів фіксується за допомогою комітів (commit). Коміт це контрольна точка у історії змін, наприклад коміт включає у себе зміну 3 файлів, створення 2 інших файлів, видалення одного файлу.

Для організації комітів використовуються бранчі та теги. Бранчі - це вказівники на певний коміт, що змінюється при додавання інших. Вони використовуються при розробці версії або певної фічі. Коли наближається час релізу, розробник створює тег, що на відміну від бранча не може бути зміненим. Теги звичайно мають маркування як і версії ПЗ.

Операції

Git дозволяє полегшити життя розробника за допомогою повністю або частково автоматичних операцій:

– операції commit/push/pull/fetch забезпечують додавання та синхронізацію змін між локальним та віддаленими репозиторіями;

– операції merge/rebase/cherry-pick надають можливість змішувати зміни між бранчами;

– операції status/log надають можливість слідкувати за змінами у проекті;

– операції checkout/revert/reset забезпечують відміну змін та навігацію між комітами, тегами або бранчами.

Додаткові функції Git сервісів

Крім звичайних команд, Git сервера такі як GitHub, GitLab тощо мають можливості:

– створення та зберігання релізів, тобто файлів для використання ПЗ (наприклад .exe файл для встановлення на Windows);

– створення issues, тобто тікетів на будь-яку проблему або нову фічу для ПЗ;

– створення pull request (merge request для GitLab), що дозволяє перед додаванням змін у основний бранч надати іншому розробнику можливість перевірити ваш код, запропонувати поліпшення тощо.

3. Початок роботи з Git

Для того щоб приступити до використання системи контролю версій (СКВ) Git, а також познайомитися з порядком її застосування, потрібно завантажити та встановити дистрибутив на комп'ютері та виконати налаштування Git.

Завантаження Git

Для завантаження *Git* треба перейти на сайт http://git-scm.com/. На сторінці буде посилання на актуальну на теперішній час версію дистрибутива (рис. 1.4).

Рисунок 1.4 – Вибір версії Git

Інсталяція Git

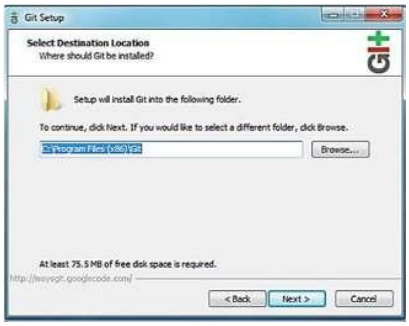
При інсталяції необхідно указати каталог для установки й деякі параметри (рис. 1.5 а – ж).


Git Setup
Welcome to the Git Setup Wizard
This will install Git version 1.7.9-preview20120201 on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel

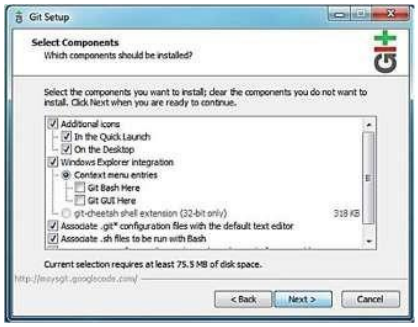
а


Git Setup
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
GNU General Public License
Version 2, June 1991
Preamble
The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public
< Back
Next >
Cancel

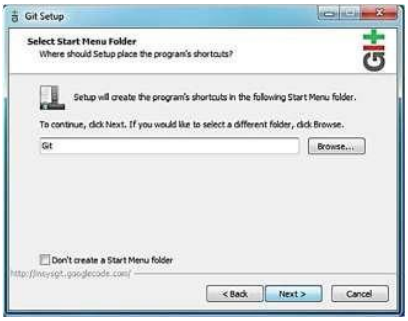
б


Git Setup
Select Destination Location
Where should Git be installed?
Setup will install Git into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
C:\Program Files (x86)\Git
Browse...
At least 75.5 MB of free disk space is required.
< Back
Next >
Cancel

в


Git Setup
Select Components
Which components should be installed?
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.
Additional icons
In the Quick Launch
On the Desktop
Windows Explorer integration
Context menu entries
Git Bash Here
Git GUI Here
git-cheetah shell extension (32-bit only)
318 KB
Associate .git* configuration files with the default text editor
Associate .sh files to be run with Bash
Current selection requires at least 75.5 MB of disk space.
< Back
Next >
Cancel

г


Git Setup
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
Git
Browse...
Don't create a Start Menu folder
< Back
Next >
Cancel

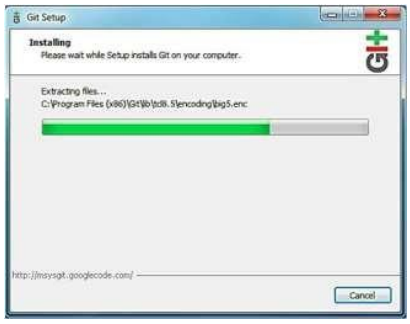
д


Git Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git Bash only
This is the most conservative choice if you are concerned about the stability of your system. Your PATH will not be modified.
Run Git from the Windows Command Prompt
This option is considered safe and no conflicts with other tools are known. Only Git will be added to your PATH. Use this option if you want to use Git from a Cygwin Prompt (make sure to not have Cygwin's Git installed).
Run Git and included Unix tools from the Windows Command Prompt
Both Git and its accompanying Unix tools will be added to your PATH.
Warning: This will override Windows tools like find.exe and sort.exe. Select this option only if you understand the implications.
< Back
Next >
Cancel

е


Git Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
< Back
Next >
Cancel

є


Git Setup
Installing
Please wait while Setup installs Git on your computer.
Extracting files...
C:\Program Files (x86)\Git\lib\tcl8.5\encoding\big5.enc
Cancel

ж

ж

Рисунок 1.5 – Вибір параметрів для установки *Git*

Налаштування Git

Після установки на робочому столі повинен з'явитися ярлик Git Bash. Після запуску системи контролю версій з'явиться консоль (рис. 1.6).

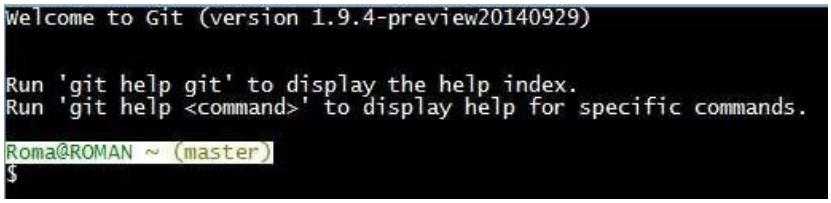


Рисунок 1.6 – Консоль системи контролю версій *Git*

Короткий словник термінів, які будуть використані у практичних роботах з вивчення Git.

Робоче дерево (Working tree) – будь-яка директорія у файловій системі, яка пов'язана з репозиторієм (що можна бачити за наявності в ній піддиректорії «.git»). Включає в себе всі файли і піддиректорії.

Коміт (Commit) – у ролі іменника: «моментальний знімок» робочого дерева в якийсь момент часу. У ролі дієслова: комітити (закомітити) – додавати Коміт в репозиторій.

Репозиторій (Repository) – це набір комітів, тобто просто архів минулих станів робочого дерева проекту.

Гілка (Branch) – просто ім'я для комітів, також зване посиланням (reference). Визначає походження – «родовід» комітів, і таким чином, є типовим представником «гілки розробки».

Checkout – операція перемикання між гілками або відновлення файлів робочого дерева.

Злиття (Merge) – поєднання змін у гілках.

Звичайно, слова на зразок «закомітити» є сленгом, проте у середовищі розробників вони дуже популярні, тому в цих роботах також будуть використовуватися.

III. Завдання на лабораторну роботу

У ході роботи виконаємо найтиповіші дії з репозиторієм.

1. Створити локальний репозиторій.
2. Додати файл у репозиторій.
3. Змінити файл у репозиторії (commit).
4. Видалити файл з репозиторія (commit).
5. Виконати checkout та створити гілку у репозиторії.
6. З'єднати гілки (операція merge).
7. Отримати історію ревізій (change log) по будь-якому файлу.

Створити локальний репозиторій

Для того щоб створити локальний репозиторій, необхідно додати нову директорію для роботи, перейти до неї. Командою git init треба проініціалізувати порожній репозиторій (рис. 1.8).

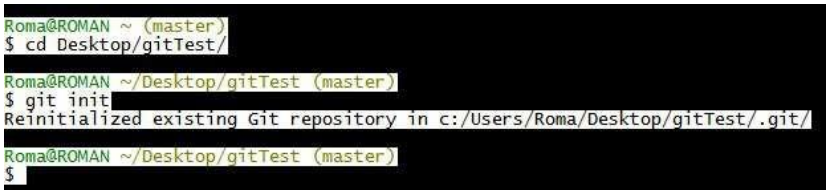


Рисунок 1.8 – Ініціалізація порожнього репозиторію

Додати файл у репозиторій

Створити у новій директорії порожній файл example.txt. Перейти уконсоль та ввести команду git status (рис. 1.9).

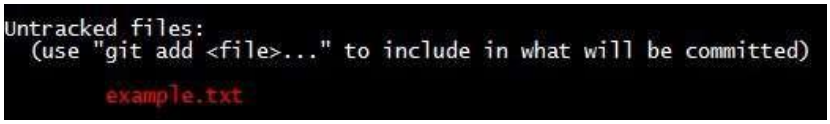


Рисунок 1.9 – Створення файлу example.txt

Командою git add example.txt додається файл під версійний контроль. Якщо потрібно додати усі нові файли у директорії, то використовується команда git add .

Після цього треба виконати commit (закомітити) (рис. 1.10). за допомогою команди git commit -m 'Added file'

Рисунок 1.10 – Додавання файлу під версійний контроль

Змінити файл у репозиторії (commit)

Додати в файл example.txt рядок “Hello World” та зберегти файл. Далі повторюються дії щодо додавання файлу до репозиторію (див. попередній пункт).

Видалити файл з репозиторію (commit)

Командою git rm example.txt видаляється файл з репозиторію. Після цього виконуємо commit («комітимося») – git commit -m 'removed file' (рис. 1.11).

Рисунок 1.11 – Видалення файлу з репозиторію

Виконати checkout та створити гілку у репозиторії

Командою git checkout -b development переключаємося на нову гілку з назвою development (рис. 1.12). Принципи створення нових гілок (розгалуження) буде розглянуто далі.

Рисунок 1.12 – Перемикання між гілками (checkout)

З'єднати гілки (merge)

Командою git merge master development з'єднуються гілки development та master (рис.1 13).

Рисунок 1.13 – З'єднання гілок

Отримати історію ревізій (change log) за будь-яким файлом

Командою git log отримуємо історію ревізій (change log) за файлом example.txt (рис. 1.14).

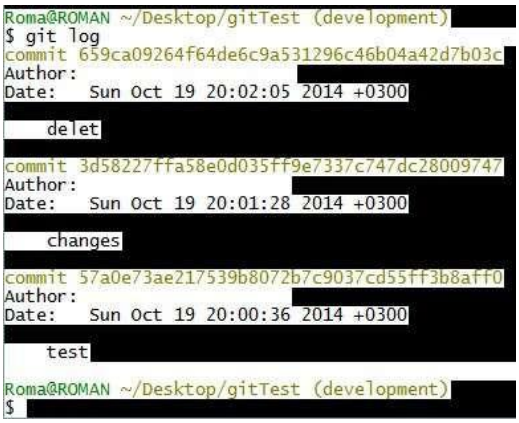


Рисунок 1.14 – Отримання історії ревізій за файлом example.txt

IV. Контрольні питання

1. Для чого потрібні системи контролю версіями (СКВ)?
2. Які завдання виконують СКВ?
3. На які групи за архітектурою побудови поділяються СКВ?
4. Що таке «Робоче дерево»?
5. Пояснити термін «коміт».
6. Пояснити терміни Checkout та Merge. У чому їхня різниця?
7. Як створити локальний репозиторій?
8. Як видалити файл з репозиторію?
9. Як отримати історію ревізій за будь-яким файлом у репозиторії?