

Управління версіями програмного забезпечення за допомогою Git

ЗМІСТ

1	ЗНАЙОМСТВО З РОЗПОДІЛЕНОЮ СИСТЕМОЮ КЕРУВАННЯ ВЕРСІЯМИ GIT	4
1.1	Підготовка до виконання роботи	4
1.2	Основи Git	4
1.2.1	Зліпки файлової системи	4
1.2.2	Локальні операції	6
1.2.3	Контроль цілісності даних	7
1.2.4	Дані тільки додаються	7
1.2.5	Стани файлів	8
1.3	Виконання роботи	9
1.3.1	Початкове налаштування	9
1.3.2	Створення сховища	10
1.3.3	Запис змін в репозиторій	11
1.3.4	Перегляд історії комітів	22
1.3.5	Скасування змін	23
1.4	Вимоги до звіту	Ошибка! Закладка не определена.
1.5	Питання для самоперевірки	28
2	ДОКУМЕНТУВАННЯ ВИМОГ І ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ЗА ДОПОМОГОЮ МОВИ UML. РОБОТА З ГІЛКАМИ В СИСТЕМІ GIT	31
2.1	Підготовка до виконання роботи	Ошибка! Закладка не определена.
2.2	Документування вимог	Ошибка! Закладка не определена.
2.2.1	Користувальницькі історії	Ошибка! Закладка не определена.

2.2.2 Сценарії використання.....	Ошибка! Закладка не определена.
2.2.3 Детальні вимоги	Ошибка! Закладка не определена.
2.3 Проектування архітектури системи	Ошибка! Закладка не определена.
2.4 Робота з гілками в системі Git	31
2.4.1 Розгалуження і злиття.....	31
2.4.2 Конфлікти при злитті	36
2.5 Вимоги до звіту	Ошибка! Закладка не определена.
2.6 Питання для самоперевірки	39

1 ЗНАЙОМСТВО З РОЗПОДІЛЕНОЮ СИСТЕМОЮ КЕРУВАННЯ ВЕРСІЯМИ GIT

1.1 Підготовка до виконання роботи

1. Створити на диску каталог (наприклад, D:\GIT_PRACTICE) і помістити в нього підкаталог зі створеними в попередніх роботах файлами тощо (наприклад, D:\GIT_PRACTICE\analysis).

2. Встановити Git, для чого необхідно завантажити exe-файл інсталятора зі сторінки проекту (<https://gitforwindows.org/>) і запустити його. Після установки буде доступна для використання, як консольна версія, так і стандартна графічна. Рекомендується використовувати Git тільки з командної оболонки, що входить до складу встановленої системи, оскільки тільки таким чином будуть доступні всі команди, які використовуються, в тому числі, і в даній роботі.

Для установки Git в інших операційних системах, необхідно звернутися до інструкції:

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

1.2 Основи Git

1.2.1 Зліпки файлової системи

Основною відмінністю Git від будь-яких інших систем управління версіями (наприклад, Subversion і їй подібних) є те, яким чином в Git організовані дані. Більшість інших систем управління версіями зберігає дані у вигляді списку змін (патчів) для файлів. Такі системи (CVS, Subversion, Perforce, Bazaar і інші) представляють збережені дані у вигляді набору файлів і змін, зроблених для кожного з цих файлів в часі (рисунок 1.1).

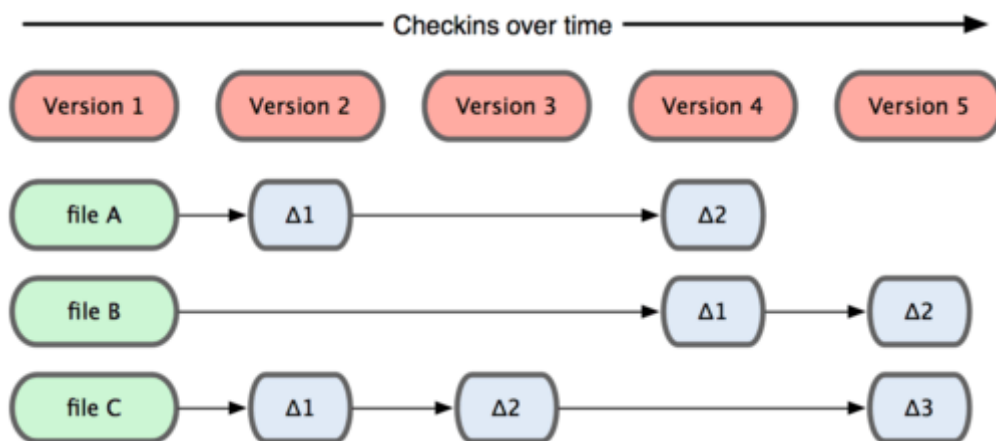


Рисунок 1.1

Замість того щоб зберігати дані у вигляді, представленому на рисунку 1.1, Git представляє збережені дані у вигляді набору зліпків невеликої файлової системи. Кожен раз, коли користувач фіксує поточну версію проекту, система управління версіями Git зберігає зліпок того, як виглядають всі файли проекту на поточний момент. Для підвищення ефективності, в разі, якщо файл не був змінений, Git не зберігає файл знову, а створює посилання на збережений раніше файл. Даний підхід зображений на рисунку 1.2.

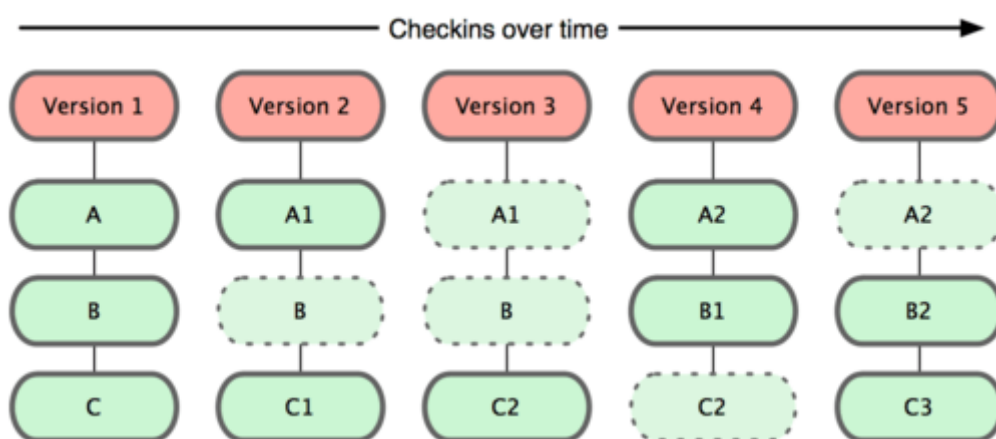


Рисунок 1.2

Дана особливість відрізняє Git практично від усіх інших систем управління версіями. Внаслідок чого, створення Git зажадало перегляду практично всіх аспектів управління версіями, які інші системи перейняли від своїх попередниць. Таким чином, Git нагадує невелику файлову систему з потужними інструментами, що працюють поверх неї, ніж звичну систему управління версіями.

1.2.2 Локальні операції

Для більшості операцій в системі Git необхідні тільки локальні файли і ресурси, тобто зазвичай інформація з інших комп'ютерів в мережі не потрібна. Оскільки вся історія проекту зберігається локально на диску, призначеному для користувача комп'ютера, більшість операцій виконуються практично миттєво.

Для демонстрації історії проекту Git не завантажує її з сервера, а просто читає її безпосередньо з локального сховища конкретного користувача, який запросив демонстрацію історії проекту. Якщо необхідно переглянути зміни між поточною версією файлу і версією, зробленої місяць тому, Git може обчислити різницю локально, замість того, щоб запитувати різницю у сервера системи керування версіями або завантажувати стару версію файлу і тільки потім здійснювати локальне порівняння.

Локальне виконання операцій означає, що лише малу частину операцій не можна виконати без доступу до мережі або VPN. У разі якщо користувач хоче попрацювати, не маючи доступу до мережі, наприклад, перебуваючи в літаку або поїзді, він може продовжувати робити комміти (фіксувати зміни проекту), а потім відправити їх на сервер, як тільки стане доступна мережа. Аналогічно, якщо VPN-клієнт не працює, все одно можна продовжувати роботу.

У багатьох інших системах управління версіями повноцінна локальна робота неможлива або вкрай незручна. Наприклад,

використовуючи Perforce, практично нічого не можна зробити без з'єднання з сервером. Працюючи з Subversion і CVS, користувач може редагувати файли, але зберегти зміни в локальну базу даних неможливо, оскільки вона відключена від центрального сховища.

1.2.3 Контроль цілісності даних

Перед тим, як будь-який файл буде збережений, Git обчислює його контрольну суму, яка використовується в якості індексу даного файлу. Тому неможливо змінити вміст файлу або каталогу так, щоб зміни не були виявлені системою Git. Дана функціональність є важливою складовою Git. Якщо інформація буде втрачена або пошкоджена при передачі, Git завжди це виявить.

Механізм, який використовується в Git для обчислення контрольних сум, називається SHA-1 хешем. Це рядок з 40 шістнадцяткових символів (0-9 і af), що обчислюється на основі вмісту файлу або структури каталогу. SHA-1 хеш виглядає приблизно наступним чином:

24b9da6552252987aa493b52f8696cd6d3b00373

При роботі з Git, такі хеші зустрічаються всюди, оскільки вони дуже широко використовуються в системі Git. Фактично, в своїй базі даних Git зберігає все не по іменах файлів, а по хешам їх вмісту.

1.2.4 Дані тільки додаються

Практичні всі дії, що здійснюються користувачем в Git, тільки додають дані в базу. Дуже складно змусити систему видалити дані або зробити щось, що не можна відмінити. Можна, як і в будь-який інший системі управління версіями, втратити дані, які ще не були збережені, але як тільки вони будуть зафіксовані, їх дуже складно втратити, особливо якщо зміни регулярно відправляються в центральний репозиторій. Тому, при використанні системи Git, можна експериментувати, не боячись щось серйозно пошкодити в проекті.

1.2.5 Стани файлів

Найважливіше, що необхідно знати про Git, це те, що в системі файли можуть перебувати в одному з трьох станів:

- 1) «зафіксований» – файл вже збережений в локальному репозиторії;
- 2) «змінений» – файл, який був змінений, але ще не зафіксований;
- 3) «підготовлений» – змінений файл, зазначений для включення в наступний комміт.

Таким чином, в проектах, що використовують Git, є три області (рисунок 1.3):

1) каталог Git (Git directory) – місце, де Git зберігає метадані та базу даних об'єктів користувальницького проекту; це найбільш важлива частина Git і саме вона копіюється, коли виконується клонування сховища з сервера;

2) робочий каталог (working directory) – витягнута з бази копія певної версії проекту; ці файли витягуються з бази даних в каталозі Git і поміщаються на диск для того, щоб їх можна було переглядати і редагувати;

3) область підготовлених файлів (staging area) – це файл, що зазвичай зберігається в каталозі Git, який містить інформацію про те, що повинно увійти в наступний комміт; іноді його називають індексом.

Стандартний робочий процес з використанням системи управління версіями Git виглядає приблизно наступним чином (рисунок 1.3):

1. Користувач вносить зміни в файли в своєму робочому каталозі.
2. Користувач готує файли, додаючи їх зліпки в область підготовлених файлів.
3. Користувач робить комміт, який бере підготовлені файли з області підготовлених файлів (індексу) і поміщає їх в каталог Git на постійне зберігання.

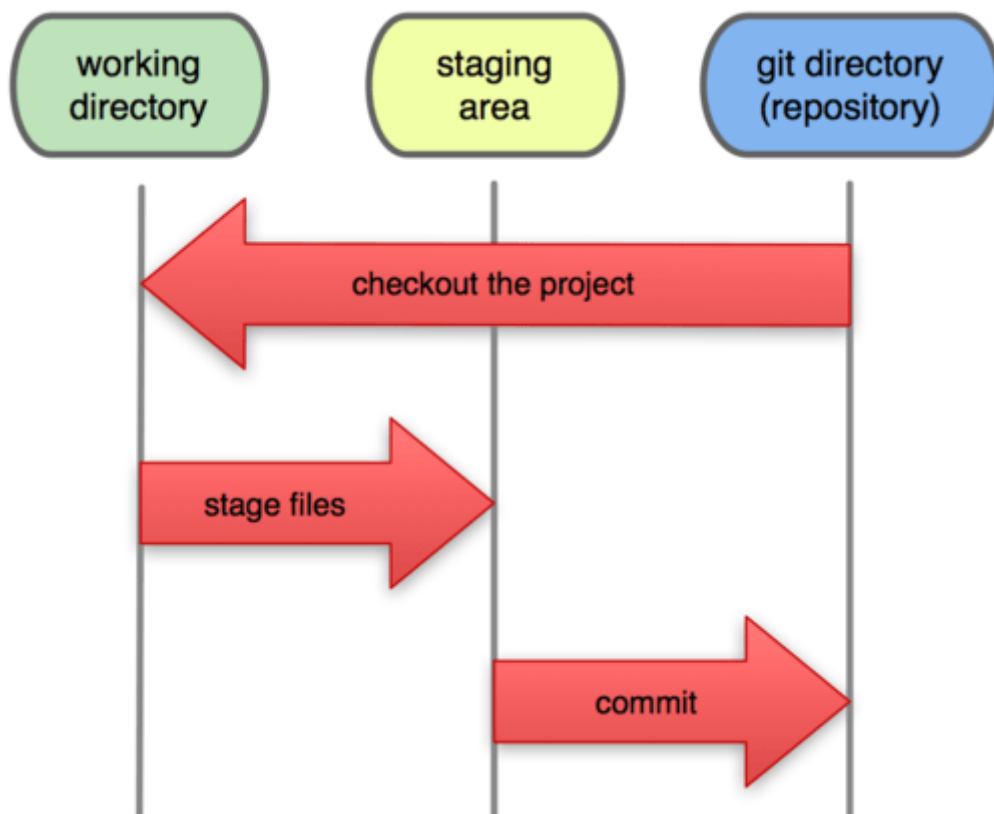


Рисунок 1.3

Якщо робоча версія файлу збігається з версією в каталозі Git, файл вважається зафіксованим. Якщо файл змінений, але доданий в область підготовлених даних, він підготовлений. Якщо ж файл змінився після вивантаження з бази, але не був підготовлений, то він вважається зміненим.

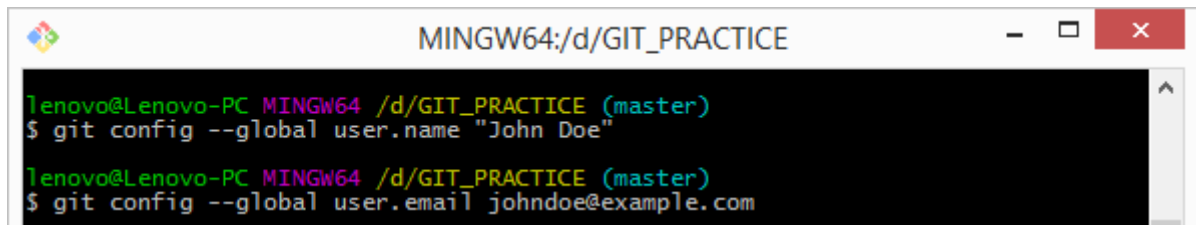
1.3 Виконання роботи

1.3.1 Початкове налаштування

До складу системи Git входить утиліта `git config`, яка дозволяє переглядати і встановлювати параметри, що контролюють всі аспекти роботи Git і його зовнішній вигляд.

Перше, що необхідно зробити після установки Git – вказати ім'я та адресу електронної пошти. Це необхідно, оскільки кожен комміт містить

цю інформацію, і вона включена в комміти, що передаються користувачем, і не може бути далі змінена. Запустити Git Bash і ввести (рисунок 1.4):



```
MINGW64:/d/GIT_PRACTICE
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git config --global user.name "John Doe"
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git config --global user.email johndoe@example.com
```

Рисунок 1.4

У разі якщо зазначена опція `--global`, то ці налаштування достатньо зробити тільки один раз. Якщо для якихось окремих проектів необхідно вказати інше ім'я або електронну пошту, можна виконати ті ж команди без параметра `--global` в каталозі з потрібним проектом.

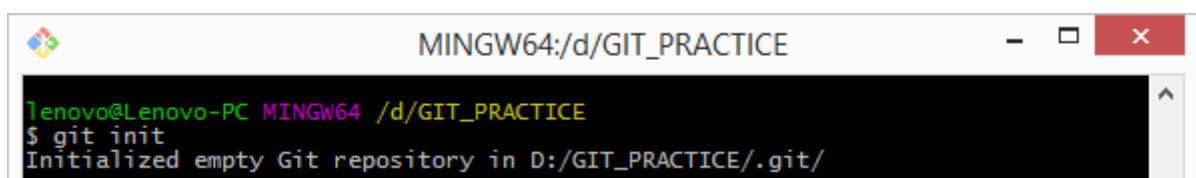
Більш детальну інформацію про налаштування Git можна отримати за адресою:

<https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup>

1.3.2 Створення сховища

Існує два основні підходи до створення сховища в Git. Перший підхід – імпорт в Git вже існуючого проекту або каталогу. Другий підхід – клонування вже існуючого сховища з сервера. Як видно з пункту 1.1, в даній роботі буде розглянуто перший підхід до створення сховища. Клонування існуючого сховища з сервера буде розглянуто пізніше.

Перейти в проектний каталог (D: \ GIT_PRACTICE) і в командному рядку (ПКМ, пункт Git Bash Here) ввести (рисунок 1.5):



```
MINGW64:/d/GIT_PRACTICE
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE
$ git init
Initialized empty Git repository in D:/GIT_PRACTICE/.git/
```

Рисунок 1.5

Команда `git init` створює в поточному каталозі новий підкаталог з ім'ям `.git`, що містить всі необхідні файли сховища Git. На цьому етапі проект все ще не знаходиться під версійним контролем.

1.3.3 Запис змін в репозиторій

Так як тепер є репозиторій Git і робоча копія файлів для проекту (`D:\GIT_PRACTICE\analysis`), необхідно робити деякі зміни і фіксувати знімки (або зліпки) стану (snapshots) цих змін в репозиторії кожен раз, коли проект досягає стану, який б хотілося зберегти.

Необхідно пам'ятати, що кожен файл в робочому каталозі може перебувати в одному з двох станів:

1) «відстежується» – файл, який був в останньому знімку стану проекту (що знаходиться під версійним контролем); він може бути незміненим, зміненим або підготовленим до коммітів;

2) «не відстежується» – будь-який файл в робочому каталозі, який не входив в останній знімок стану і не підготовлений до коммітів.

Коли репозиторій клонований, всі файли відстежуються і вони є незміненими, тому що вони тільки були клоновані (checked out), але не були відредаговані.

Як тільки файли будуть відредаговані, Git розглядатиме їх як змінені. Зміни необхідно індексувати (готувати до коммітів) і потім фіксувати, після чого цикл повторюється. Даний життєвий цикл зображений на рисунку 1.6.

У нашому випадку, проект був імпортований, а не клонований з іншого сховища, тому всі файли не відстежуються, тому що вони не входили в останній знімок стану (ще не було зроблено жодного знімка) і ще не були підготовлені до коммітів.

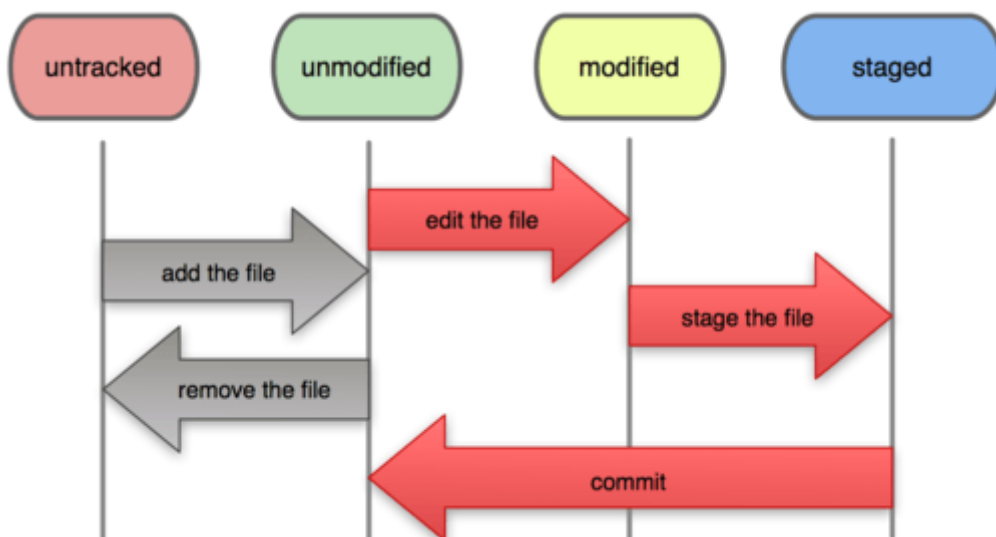


Рисунок 1.6

Для визначення, які файли, в якому стані знаходяться, використовується команда `git status` (рисунок 1.7):

```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

analysis/
```

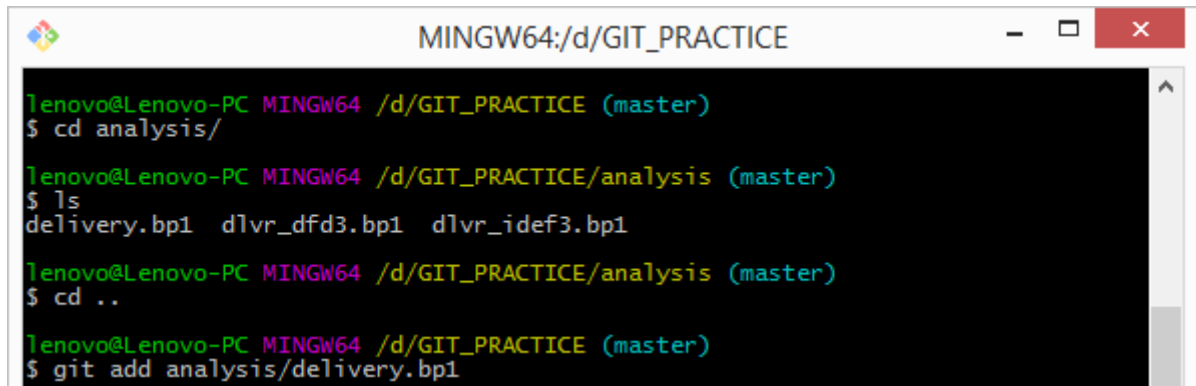
Рисунок 1.7

Зрозуміти, що каталог `analysis` не відстежується можна по тому, що він знаходиться в секції «Untracked files» у висновку команди `status`. Крім того, команда повідомляє користувачеві на який гілці (branch) він зараз перебуває. Поки що це завжди гілка `master` – це гілка за замовчуванням. Особливості роботи з гілками будуть розглянуті пізніше.

Система Git не стане додавати файли, що не відстежуються в комміти, поки користувач цього явно не вкаже. Це оберігає від

випадкового додавання в репозиторій бінарних файлів або будь-яких інших, які не планувалися додавати.

Для того щоб відстежувати новий файл, використовується команда `git add`. Щоб додати під версійний контроль один з файлів каталогу `analysis`, необхідно виконати наступне (рисунок 1.8):



```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cd analysis/

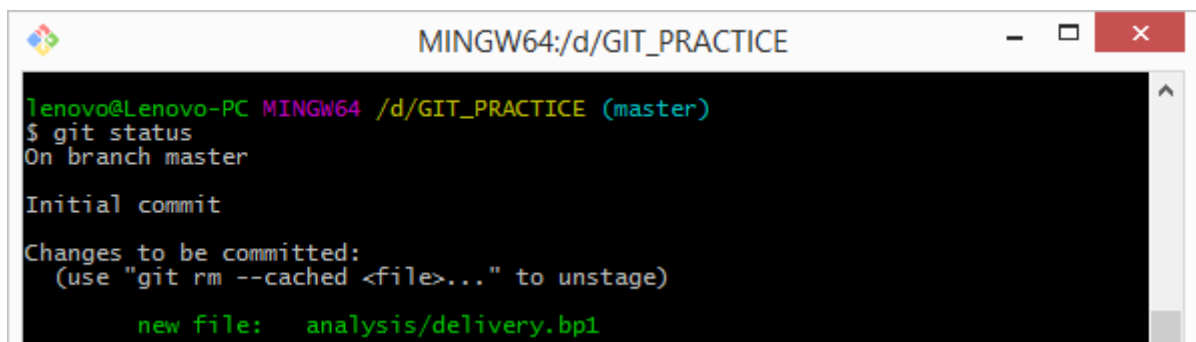
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ ls
delivery.bp1  dlvr_dfd3.bp1  dlvr_idef3.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ cd ..

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/delivery.bp1
```

Рисунок 1.8

Якщо знову виконати команду `status`, то буде видно, що файл `delivery.bp1` тепер відстежується і є індексованим (рисунок 1.9):



```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   analysis/delivery.bp1
```

Рисунок 1.9

Видно, що файл проіндексований тому, що він знаходиться в секції «Changes to be committed». Якщо комміт буде виконаний в цей момент, то версія файлу, що існувала на момент виконання команди `git add`, буде додана в історію знімків стану. Команда `git add` приймає як параметр шлях

до файлу або каталогу, якщо це каталог, команда рекурсивно додає (індексує) всі файли в даному каталозі.

Додати решту невідстежуваних файлів з каталогу `analysis` за допомогою команди `git add`, перевірити стан індексу за допомогою команди `git status` (рисунок 1.10):



```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

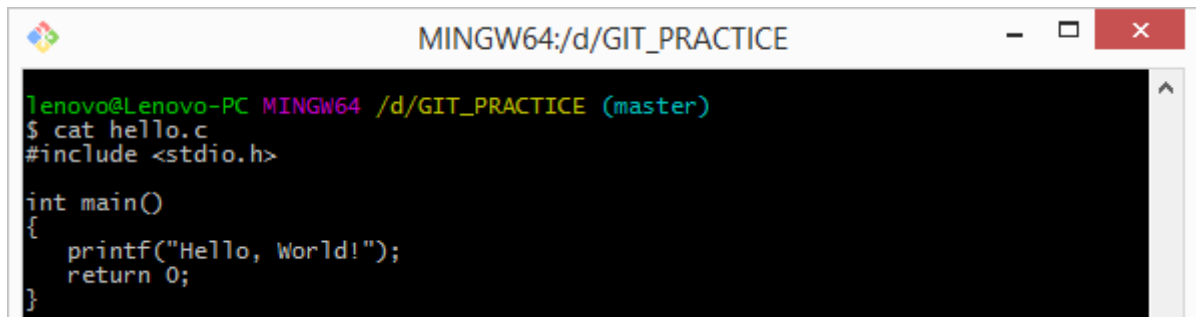
        new file:   analysis/delivery.bp1
        new file:   analysis/dlvr_dfd3.bp1
        new file:   analysis/dlvr_idef3.bp1
```

Рисунок 1.10

Найчастіше, в проєкті міститься група файлів, які не тільки не потрібно автоматично додавати в репозиторій, а й бачити в списку невідстежуваних. До таких файлів зазвичай ставляться автоматично генеруються файли (різні логи, результати складання програм і т.п.). В такому випадку можна створити файл `.gitignore` з перерахуванням шаблонів відповідних таких файлів. Детальну інформацію про формування `.gitignore` можна отримати за посиланням:

<https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository>

Створити в робочому каталозі файл `hello.c`, який, з якоїсь причини, не повинен відстежуватись (рисунок 1.11):

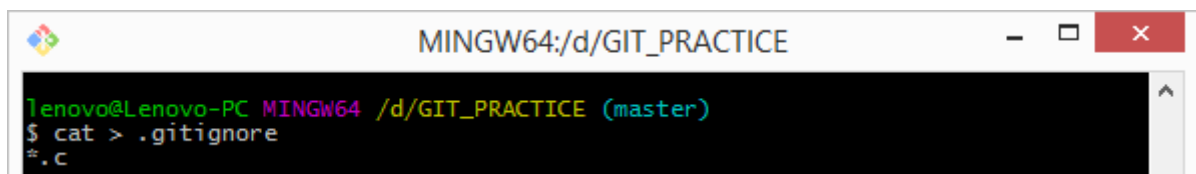
A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The prompt is "lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)". The user has entered the command "\$ cat hello.c". The output shows the content of the file: "#include <stdio.h>", "int main()", "{", " printf("Hello, World!");", " return 0;", "}".

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat hello.c
#include <stdio.h>

int main()
{
    printf("Hello, World!");
    return 0;
}
```

Рисунок 1.11

Створити файл з ім'ям .gitignore і наступним змістом, для виходу з режиму введення вмісту файлу натиснути Ctrl + D (рисунок 1.12):

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The prompt is "lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)". The user has entered the command "\$ cat > .gitignore". The output shows the first line of the file: "#.c".

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat > .gitignore
#.c
```

Рисунок 1.12

Єдиний рядок створеного файлу .gitignore наказує Git ігнорувати будь-які файли, що закінчуються на .c. У список можна також включати каталоги, які потрібно ігнорувати, закінчуючи шаблон символом слеша (/) для вказівки каталогу. Порожні рядки, а також рядки, що починаються з # (коментарі), ігноруються.

Перевірити коректність створення файлу .gitignore за допомогою команди git status (рисунок 1.13):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

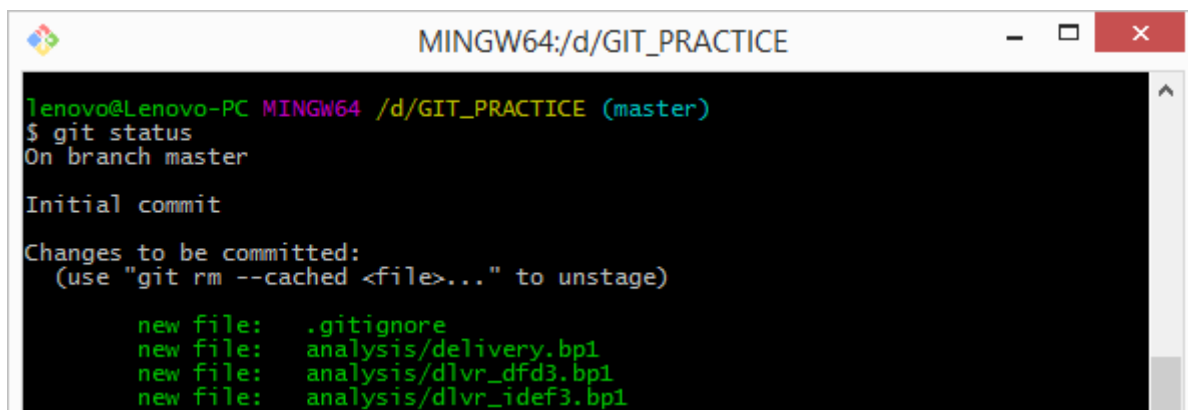
        new file:   analysis/delivery.bp1
        new file:   analysis/dlvr_dfd3.bp1
        new file:   analysis/dlvr_idef3.bp1

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
```

Рисунок 1.13

Як видно, створений раніше файл `hello.c` не відстежується, оскільки він відсутній у списку файлів секції «Untracked files», а доданий в індекс він не був. Файл `.gitignore` також необхідно підготувати до фіксації змін за допомогою команди `git add` (рисунок 1.14):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

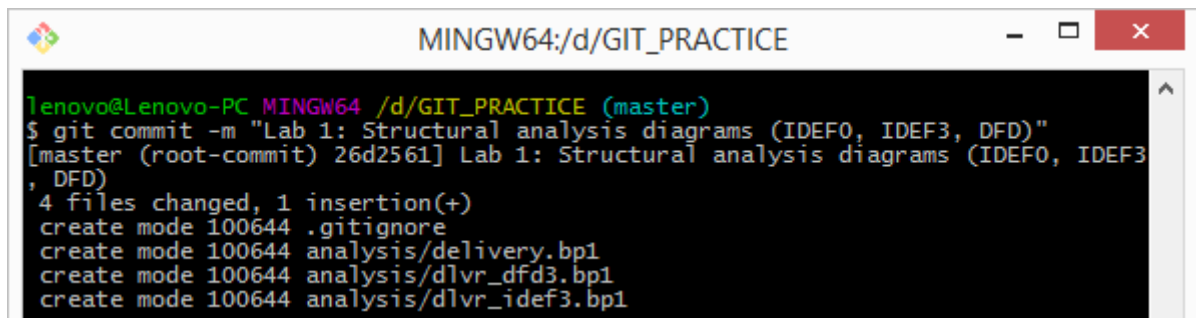
        new file:   .gitignore
        new file:   analysis/delivery.bp1
        new file:   analysis/dlvr_dfd3.bp1
        new file:   analysis/dlvr_idef3.bp1
```

Рисунок 1.14

Тепер, коли індекс налаштований так, як це було необхідно, можна зафіксувати зроблені зміни. Необхідно запам'ятати, що все, що до сих пір не було проіндексовано – будь-які файли, створені або змінені користувачем, і для яких не була виконана команда `git add` після моменту редагування – не увійде в коміт. Такі файли залишаться зміненими на

диску користувача. У нашому випадку видно, що все проіндексовано (рисунок 1.14) і готове до фіксації.

Для фіксації змін використовується команда `git commit`. Коментар до коммітів зазвичай набирається в командному рядку разом з командою `commit`, вказуючи після параметра `-m` (рисунок 1.15):

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The prompt is "lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)". The command entered is "\$ git commit -m 'Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)'". The output shows the commit hash "[master (root-commit) 26d2561]", the commit message, and a list of files changed: "4 files changed, 1 insertion(+)", followed by "create mode 100644 .gitignore", "create mode 100644 analysis/delivery.bp1", "create mode 100644 analysis/dlvr_dfd3.bp1", and "create mode 100644 analysis/dlvr_idef3.bp1".

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit -m "Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)"
[master (root-commit) 26d2561] Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)
4 files changed, 1 insertion(+)
create mode 100644 .gitignore
create mode 100644 analysis/delivery.bp1
create mode 100644 analysis/dlvr_dfd3.bp1
create mode 100644 analysis/dlvr_idef3.bp1
```

Рисунок 1.15

Є й інший спосіб – набрати `git commit` без параметрів. Ця команда відкриє текстовий редактор з коментарем за замовчуванням, який містить закоментований результат роботи команди `git status`, який можна видалити і набрати свій коментар або залишити для нагадування про те, що було зафіксовано.

Після того, як комміт був створений, була виведена інформація про гілку, на яку був виконаний комміт (`master`), яка контрольна сума SHA-1 у цього комміту (`26d2561`), скільки файлів було змінено (`4 files changed`), а також статистику по доданим / віддаленим рядкам в цьому комміт (рисунок 1.15).

Для того щоб видалити файл з Git, необхідно видалити його з файлів, що відстежуються (точніше, видалити його з індексу), а потім виконати комміт. Це дозволять зробити команда `git rm`, яка також видаляє файл з робочого каталогу, тому він не буде відзначений в наступний раз як такий, що не відстежується. Якщо ж просто видалити файл з робочого каталогу, то він буде показаний в секції «Changes not staged for commit» виведення

команди `git status`. І лише після виконання команди `git rm`, видалення файлу потрапить в індекс.

Якщо файл був змінений і вже проіндексований, необхідно використовувати примусове видалення з допомогою параметра `-f`.

Корисною функцією є видалення файлу з індексу, залишаючи його при цьому в робочому каталозі. Це особливо необхідно, коли користувач забув додати щось в файл `.gitignore` і помилково проіндексував, наприклад, великий файл з логами або проміжні файли компіляції. Для цього необхідно використовувати опцію `--cached`.

Припустимо, що файл DFD-діаграми з якоїсь причини необхідно видалити з індексу, але, в той же час, залишити доступним в робочому каталозі для подальшого редагування або виправлення виявлених помилок (рисунок 1.16):

A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The terminal shows the following commands and output:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git rm --cached analysis/dlvr_dfd3.bp1
rm 'analysis/dlvr_dfd3.bp1'

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    analysis/dlvr_dfd3.bp1

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    analysis/dlvr_dfd3.bp1
```

Рисунок 1.16

В результаті виконання команди `git rm` з опцією `--cached`, в наступному комміті даний файл буде видалений з каталогу `analysis` в репозиторії, але залишиться невідстежуваним системою Git і доступним для редагування в робочому каталозі.

Фіксацію видалення файлу `dlvr_dfd3.bp1` виконати другим способом (за допомогою команди `git commit` без параметрів), набравши коментар до коммітів в текстовому редакторі, попередньо видаливши створений за замовчуванням коментар (рисунок 1.17):



Рисунок 1.17

За замовчуванням в Git використовується зручний і лаконічний редактор Vim. Для переходу в режим редагування необхідно натиснути `i` або `Insert`, для повернення в режим команд – `Esc`. Зберегти файл і вийти з редактора можна за допомогою команди: `wq` (для примусового запису використовувати: `wq!`), Після чого буде показаний висновок команди `git commit` (рисунок 1.18):

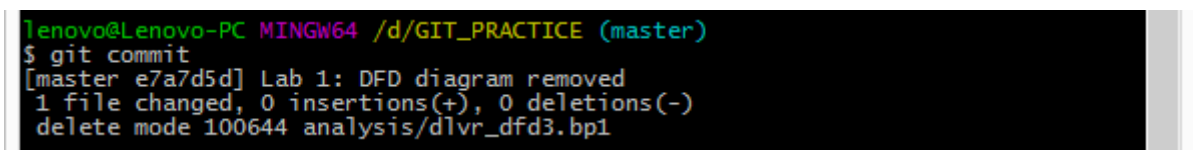
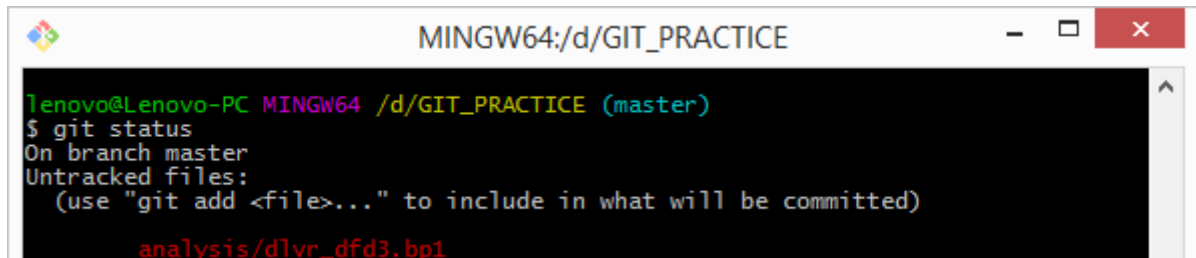


Рисунок 1.18

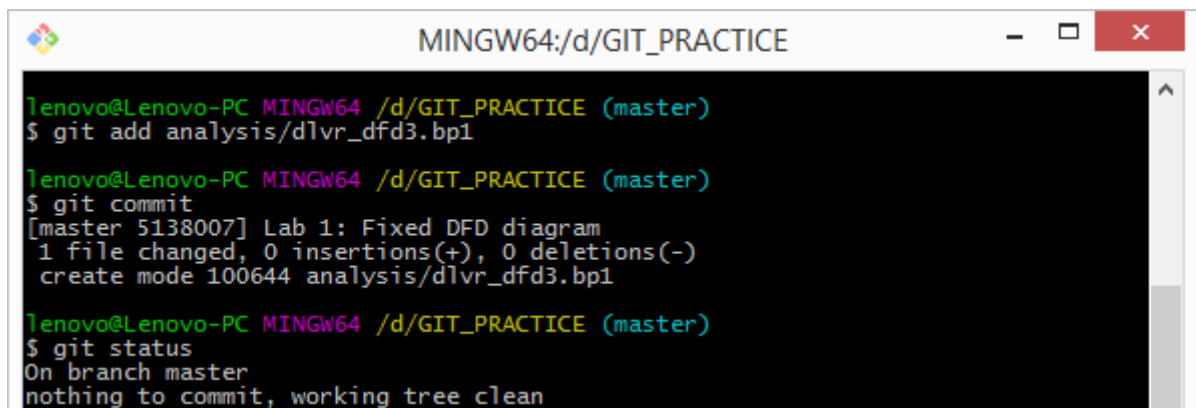
Набравши команду `git status`, можна переконатися, що «віддалений» файл все ще доступний для редагування в робочому каталозі і відзначений системою Git як такий, що не відстежується (рисунок 1.19):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
analysis/dlvr_dfd3.bp1
```

Рисунок 1.19

При бажанні можна внести в даний файл будь-які зміни (наприклад, виправити недоліки зазначені викладачем), проіндексувати його, і зафіксувати зміни за допомогою команди `commit` (для зручності можна використовувати редактор Vim для введення коментаря) як це показано на рисунку 1.20:



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/dlvr_dfd3.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit
[master 5138007] Lab 1: Fixed DFD diagram
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 analysis/dlvr_dfd3.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

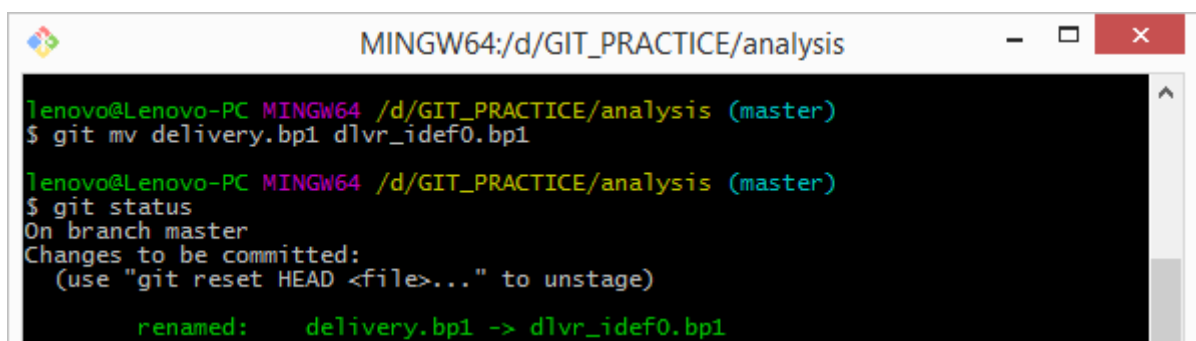
Рисунок 1.20

В команду `git rm` можна передавати файли, каталоги або шаблони. Наприклад, для видалення всіх файлів, що мають розширення `.bp1`, з каталогу `analysis`, можна використовувати команду `git rm analysis / \ *.bp1`.

На відміну від багатьох інших систем управління версіями, Git не відслідковує переміщення файлів явно. При перейменуванні файлу в Git в ньому не зберігається ніяких метаданих, які говорять про те, що файл був перейменований.

У разі якщо файл необхідно перейменувати або перемістити, необхідно використовувати команду `git mv`.

Наприклад, з тієї чи іншої причини потрібно перейменувати файл `delivery.bp1` в `dlvr_idef0.bp1` (рисунок 1.21):



```
MINGW64:/d/GIT_PRACTICE/analysis
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ git mv delivery.bp1 dlvr_idef0.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    delivery.bp1 -> dlvr_idef0.bp1
```

Рисунок 1.21

Після того, як команда `git mv` буде виконана, якщо перевірити стан області підготовлених файлів за допомогою команди `status`, буде видно, що Git проіндексував перейменування файлу (рисунок 1.21).

Однак, це еквівалентно виконанню наступних команд:

```
mv delivery.bp1 dlvr_idef0.bp1
```

```
git rm delivery.bp1
```

```
git add dlvr_idef0.bp1
```

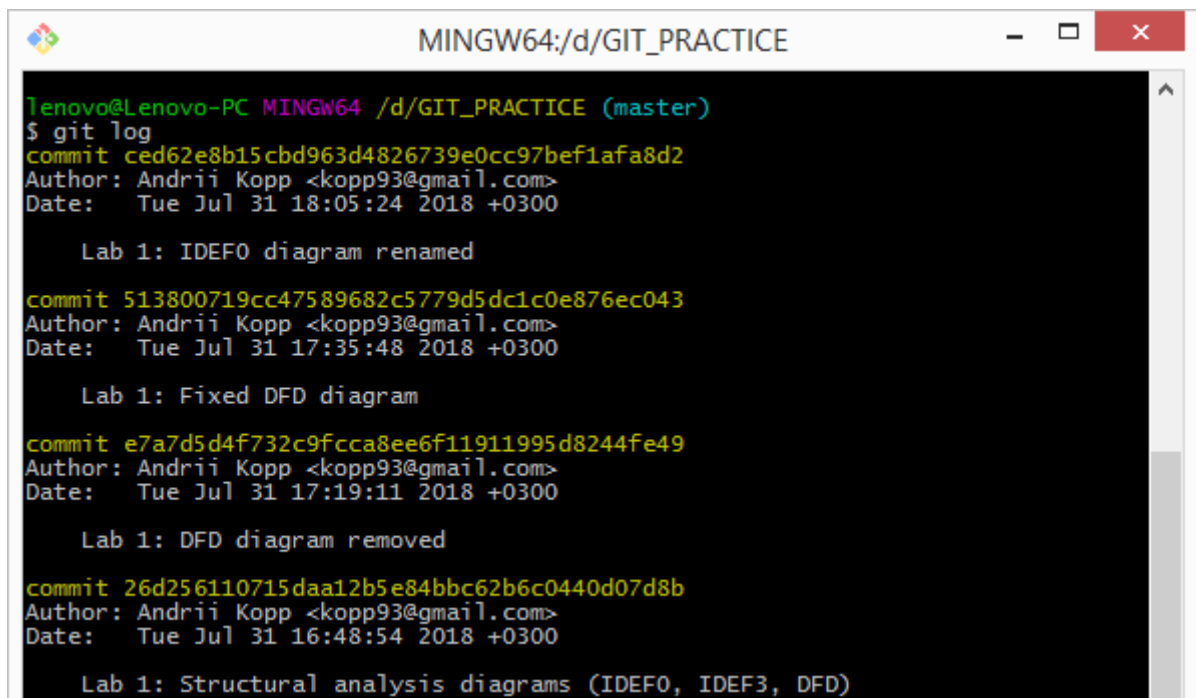
Система Git неявно визначає, що відбулося перейменування, тому неважливо яким способом буде перейменований файл.

Після того, як файл був перейменований, зміни необхідно знову зафіксувати за допомогою команди `git commit`.

1.3.4 Перегляд історії коммітів

Для перегляду історії коммітів використовується простий і в той же час потужний інструмент – команда `git log`. Дана команда особливо корисна, коли користувач клонує репозиторій з уже існуючою історією коммітів і хоче дізнатися, що ж відбувалося з цим репозиторієм.

В результаті виконання `git log` буде виведений список коммітів, створених в даному репозиторії, в зворотному хронологічному порядку (рисунок 1.22):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit ced62e8b15cbd963d4826739e0cc97bef1afa8d2
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEF0 diagram renamed

commit 513800719cc47589682c5779d5dc1c0e876ec043
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:35:48 2018 +0300

    Lab 1: Fixed DFD diagram

commit e7a7d5d4f732c9fcca8ee6f11911995d8244fe49
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:19:11 2018 +0300

    Lab 1: DFD diagram removed

commit 26d256110715daa12b5e84bbc62b6c0440d07d8b
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 16:48:54 2018 +0300

    Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)
```

Рисунок 1.22

Один з найбільш корисних параметрів команди `log` – це `-p`, який показує дельту (різницю), привнесену кожним коммітів. Також можна використовувати `-2`, що обмежить висновок до 2-х останніх записів.

Більш докладно про команду `git log` можна прочитати за посиланням:

<https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History>

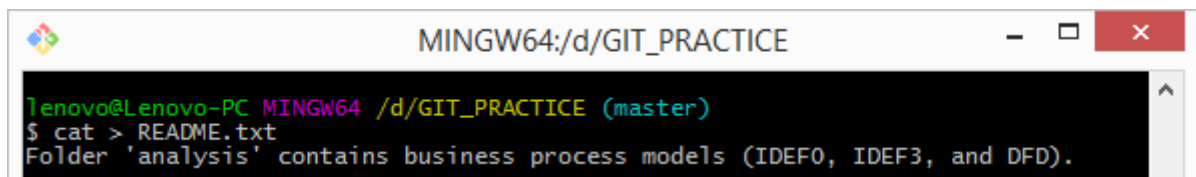
1.3.5 Скасування змін

На будь-якій стадії роботи може виникнути необхідність що-небудь скасувати. Однією з небагатьох ситуацій в Git, коли можна втратити свою роботу, якщо зробити щось неправильно, є така, коли не завжди можливо скасувати самі скасування.

Одна з типових відмін відбувається тоді, коли користувач робить комміт занадто рано, забувши додати якісь файли або ввівши не той коментар до коммітів. Якщо необхідно зробити цей комміт ще раз, можна виконати `git commit` з опцією `--amend`.

Якщо після останнього комміту не було ніяких змін, то стан проекту буде абсолютно таким же і все, що буде потрібно змінити, це коментар до коммітів в редакторі.

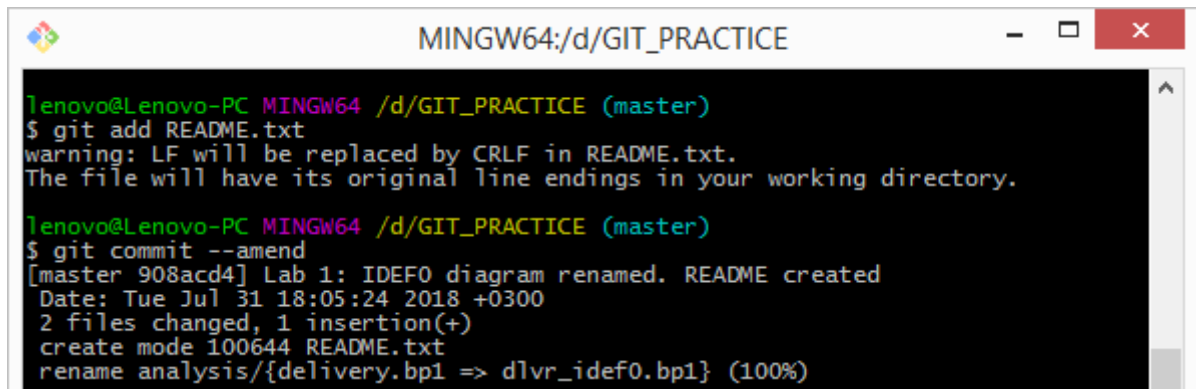
Наприклад, перед тим як виконати останній комміт (фіксація перейменування файлу IDEF0-діаграми), необхідно було також проіндексувати файл README.txt такого змісту (рисунок 1.23):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat > README.txt
Folder 'analysis' contains business process models (IDEF0, IDEF3, and DFD).
```

Рисунок 1.23

Тепер же, за допомогою наступних команд, можна додати в попередній комміт файл README.txt і відредагувати коментар до коммітів (рисунок 1.24):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add README.txt
warning: LF will be replaced by CRLF in README.txt.
The file will have its original line endings in your working directory.

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit --amend
[master 908acd4] Lab 1: IDEF0 diagram renamed. README created
Date: Tue Jul 31 18:05:24 2018 +0300
2 files changed, 1 insertion(+)
create mode 100644 README.txt
rename analysis/{delivery.bp1 => dlvr_idef0.bp1} (100%)
```

Рисунок 1.24

Перевірити, чи дійсно був змінений останній комміт, а не створений новий, можна за допомогою команди `git log` (рисунок 1.25):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 908acd45edc20615c35c9d247aaddff482553c51
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEF0 diagram renamed. README created

commit 513800719cc47589682c5779d5dc1c0e876ec043
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:35:48 2018 +0300

    Lab 1: Fixed DFD diagram

commit e7a7d5d4f732c9fcca8ee6f11911995d8244fe49
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:19:11 2018 +0300

    Lab 1: DFD diagram removed

commit 26d256110715daa12b5e84bbc62b6c0440d07d8b
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 16:48:54 2018 +0300

    Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)
```

Рисунок 1.25

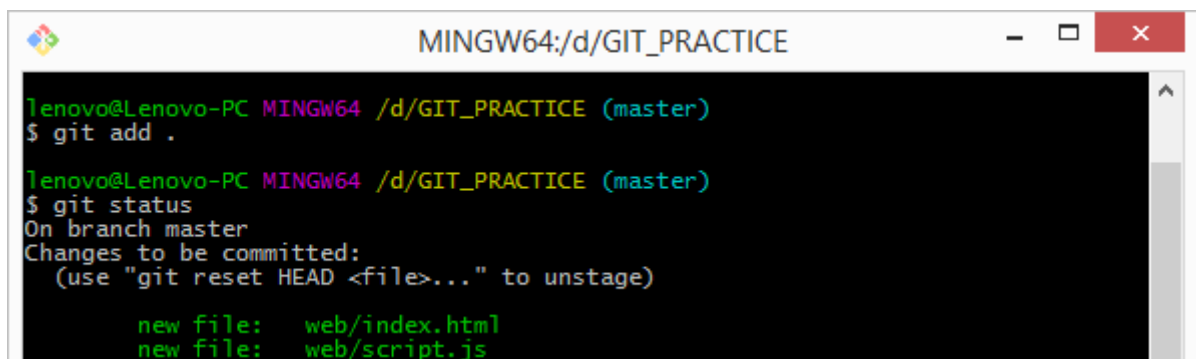
Для демонстрації того, як скасувати індексацію файлу, необхідно створити два файли `index.html` і `script.js` в каталозі `web` (рисунок 1.26):

A terminal window titled 'MINGW64:/d/GIT_PRACTICE' showing the creation of two files. The first command is 'cat web/index.html' which outputs an HTML document with a title 'Index' and a div with id 'hello'. The second command is 'cat web/script.js' which outputs a JavaScript snippet that sets the innerHTML of the 'hello' div to 'Hello World!'.

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat web/index.html
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello"></div>
<script src="script.js"></script>
</body>
</html>
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat web/script.js
var layout = document.getElementById("hello");
layout.innerHTML = "Hello World!";
```

Рисунок 1.26

Для індексації всіх невідстежуваних файлів проекту, в Git можна використовувати команду `git add .` (рисунок 1.27):

A terminal window titled 'MINGW64:/d/GIT_PRACTICE' showing the execution of 'git add .' and 'git status'. The output of 'git status' shows that two new files, 'web/index.html' and 'web/script.js', are staged for commit.

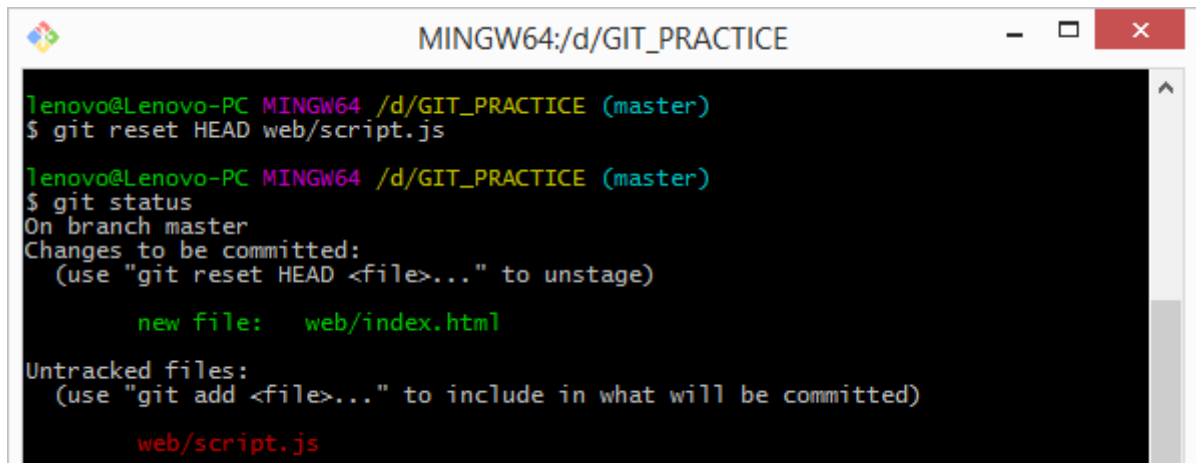
```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add .

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   web/index.html
    new file:   web/script.js
```

Рисунок 1.27

Але що, якщо ці два файли необхідно було записати в два окремих комміти? Висновок команди `git status` підказує, що скасувати індексацію одного з двох файлів можна за допомогою команди `git reset` (рисунок 1.28):

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The terminal shows the following commands and output:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git reset HEAD web/script.js

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

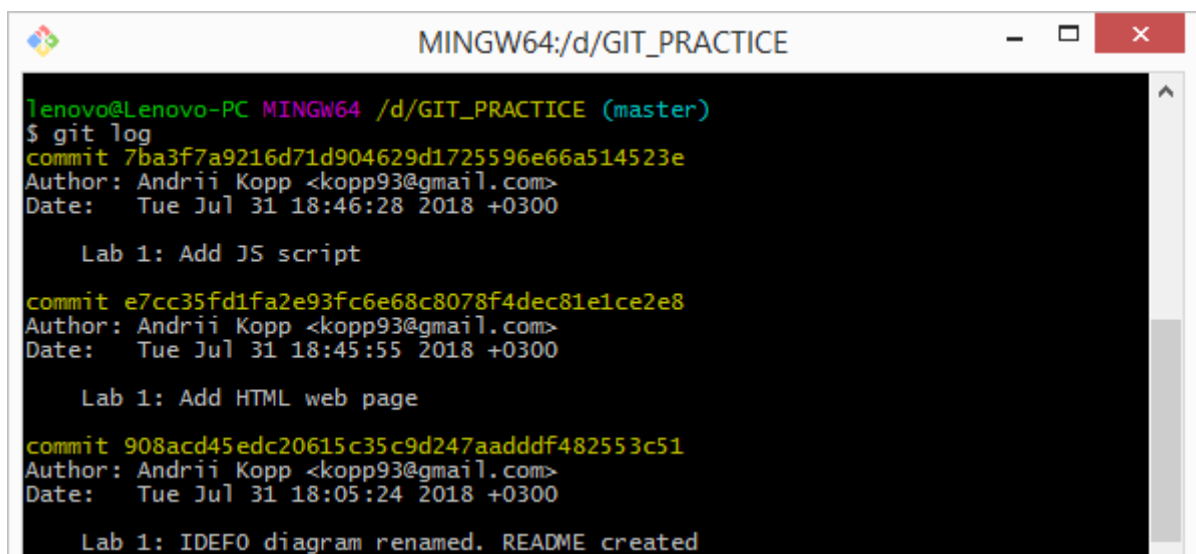
        new file:   web/index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        web/script.js
```

Рисунок 1.28

Тепер, коли другий створений файл (script.js) вважається не індексованим, можна виконати фіксацію першого файлу (index.html), додати в область підготовлених файлів другий файл, і виконати його фіксацію в окремому комміті (рисунок 1.29).

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The terminal shows the output of the 'git log' command:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 7ba3f7a9216d71d904629d1725596e66a514523e
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:46:28 2018 +0300

    Lab 1: Add JS script

commit e7cc35fd1fa2e93fc6e68c8078f4dec81e1ce2e8
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:45:55 2018 +0300

    Lab 1: Add HTML web page

commit 908acd45edc20615c35c9d247aadddf482553c51
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEFO diagram renamed. README created
```

Рисунок 1.29

Припустимо, що напис «Hello World» на веб-сторінці «Index» потрібно відображати у вигляді заголовка першого рівня. Для цього внесемо відповідні зміни в файл index.html (рисунок 1.30):

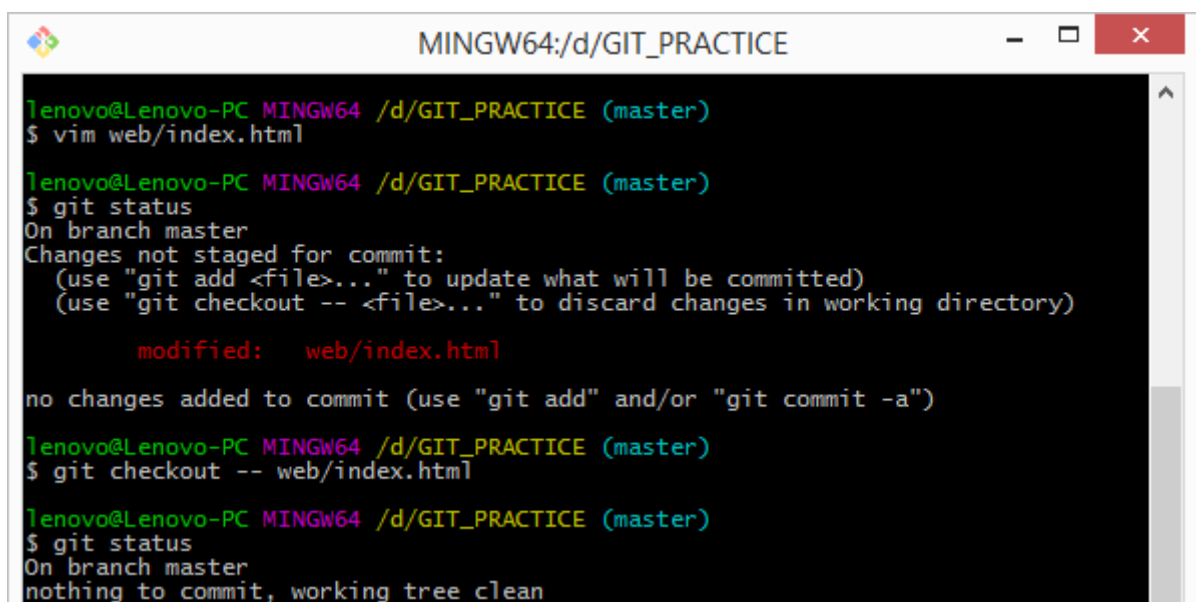
A screenshot of a MINGW64 terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The terminal shows a text editor with the following HTML code:

```
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
  <h1><div id="hello"></div></h1>
<script src="script.js"></script>
</body>
</html>
```

The status bar at the bottom indicates the file path is '/d/GIT_PRACTICE/web/index.html', the editor is in 'dos' mode, the time is '18:37 31/07/2018', and the cursor is at line 7, column 32-39. The prompt is ':wq!|'.

Рисунок 1.30

Але тепер виявилося, що залишати дані зміни не потрібно. Для швидкої відміни змін, повернення файлу в той стан, в якому він перебував під час останнього комміту, можна скористатися командою `git checkout` (рисунок 1.31):

A screenshot of a MINGW64 terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The terminal shows the following commands and output:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ vim web/index.html

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   web/index.html

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

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git checkout -- web/index.html

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 1.31

Для перевірки того, що файл повернувся в той стан, в якому він був під час останнього комміту, необхідно переглянути його вміст (рисунк 1.32):

A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The prompt is 'lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)'. The command '\$ cat web/index.html' has been executed, displaying the following HTML code:

```
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello"></div>
<script src="script.js"></script>
</body>
</html>
```

Рисунок 1.32

Необхідно розуміти, що `git checkout` – небезпечна команда: всі зроблені зміни в цьому файлі пропали – поверх нього був просто скопійований інший файл. Ніколи не слід використовувати цю команду, якщо немає повної впевненості, що цей файл не потрібен. Більш переважними способами є приховання (`stash`) і розгалуження. Ці способи будуть розглянуті пізніше.

1.5 Питання для самоперевірки

1. У чому полягає основна відмінність Git від інших систем управління версіями?
2. Що таке знімки файлової системи в Git і для чого вони призначені?
3. У чому полягає особливість зберігання історії проекту в системі Git? Основні переваги і недоліки даної системи?
4. Що таке комміт (фіксація змін в проекті) і для чого він призначений?

5. Яким чином система Git здійснює контроль цілісності даних? Що таке SHA-1 хеш?

6. Яким чином система Git фіксує дії, які виконуються користувачем? У чому перевага такого способу організації змін?

7. В яких станах можуть перебувати файли в системі Git? Коротко опишіть кожний з цих станів.

8. Які області зберігання файлів існують в проектах, що використовують Git як систему управління версіями?

9. Опишіть основні етапи робочого процесу з використанням системи управління версіями Git. В яких випадках файл вважається зміненим, підготовленим або зафіксованим?

10. Яким чином здійснюється початкове налаштування системи Git? Призначення команди `git config`.

11. Для чого призначена опція `--global` команди `git config`?

12. Які існують способи створення сховища в Git? Призначення команди `git init`.

13. Чим відстежуємі файли в робочому каталозі відрізняються від невідстежуваних?

14. Для чого призначена команда `git status`? Яка інформація виводиться при виконанні даної команди?

15. В якій гілці відбувається робота в системі Git за замовчуванням?

16. Для чого призначена команда `git add`? Які основні особливості використання даної команди?

17. Яким чином можна уникнути індексації небажаних файлів (логи, результати складання програм і т.п.)?

18. Для чого призначена команда `git commit`? Які основні особливості використання даної команди?

19. Яка інформація буде виведена в результаті виконання команди `git commit`?

20. Яким чином можна видалити файл з Git? Що робити в разі, якщо файл вже був проіндексований? Як видалити файл з індексу, але залишити в робочому каталозі?

21. За допомогою якої команди можна видалити всі файли з робочого каталогу, що мають певне розширення?

22. Яким чином здійснюється переміщення або перейменування файлів в системі Git? Яка команда для цього використовується? Які існують альтернативні способи виконання даних операцій?

23. Для чого призначена команда `git log`? Які параметри даної команди можна використовувати для виведення більш детальної інформації?

24. Яким чином в системі Git можна повторно виконати останній коміт з урахуванням необхідних змін? За яких умов можлива така операція?

25. Яким чином можна скасувати індексацію файлу? Як проіндексувати всі невідстежуємих файли?

26. Для чого призначена команда `git checkout`? Чому цю команду необхідно використовувати з обережністю?

2 ДОКУМЕНТУВАННЯ ВИМОГ І ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ЗА ДОПОМОГОЮ МОВИ UML. РОБОТА З ГІЛКАМИ В СИСТЕМІ GIT

2.4 Робота з гілками в системі Git

2.4.1 Розгалуження і злиття

В результаті роботи в гілці master, використовуваної системою Git за замовчуванням, вже є кілька коммітів (рисунок 2.12).

A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The terminal shows the output of the 'git log' command. It lists four commits in descending order. Each commit entry includes a commit hash, the author's name and email, the date, and a description of the commit. The commits are: 1. 'commit d53a9ee238e95d8cf89d2ab6a056550d718dd020' by Andrii Kopp, dated Mon Aug 6 11:55:57 2018, with description 'Lab 2: UML models'. 2. 'commit ea3206b1343ffd67916f7edd6387790ab1926c95' by Andrii Kopp, dated Mon Aug 6 11:54:42 2018, with description 'Lab 2: User stories list'. 3. 'commit 7ba3f7a9216d71d904629d1725596e66a514523e' by Andrii Kopp, dated Tue Jul 31 18:46:28 2018, with description 'Lab 1: Add JS script'. 4. 'commit e7cc35fd1fa2e93fc6e68c8078f4dec81e1ce2e8' by Andrii Kopp, dated Tue Jul 31 18:45:55 2018, with description 'Lab 1: Add JS script'. The terminal window has a standard Windows-style title bar with minimize, maximize, and close buttons.

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit d53a9ee238e95d8cf89d2ab6a056550d718dd020
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Mon Aug 6 11:55:57 2018 +0300

    Lab 2: UML models

commit ea3206b1343ffd67916f7edd6387790ab1926c95
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Mon Aug 6 11:54:42 2018 +0300

    Lab 2: User stories list

commit 7ba3f7a9216d71d904629d1725596e66a514523e
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:46:28 2018 +0300

    Lab 1: Add JS script

commit e7cc35fd1fa2e93fc6e68c8078f4dec81e1ce2e8
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:45:55 2018 +0300
```

Рисунок 2.12

Уявімо, що після того, як було виконано документування вимог і проектування архітектури створюваної системи, далі необхідно перейти до роботи над створенням прототипу інформаційної системи.

Так як створення прототипу є відокремленою завданням в рамках традиційного життєвого циклу розробки програмного забезпечення (рисунок 2.13), необхідно створити нову гілку в системі управління версіями Git і працювати на ній.

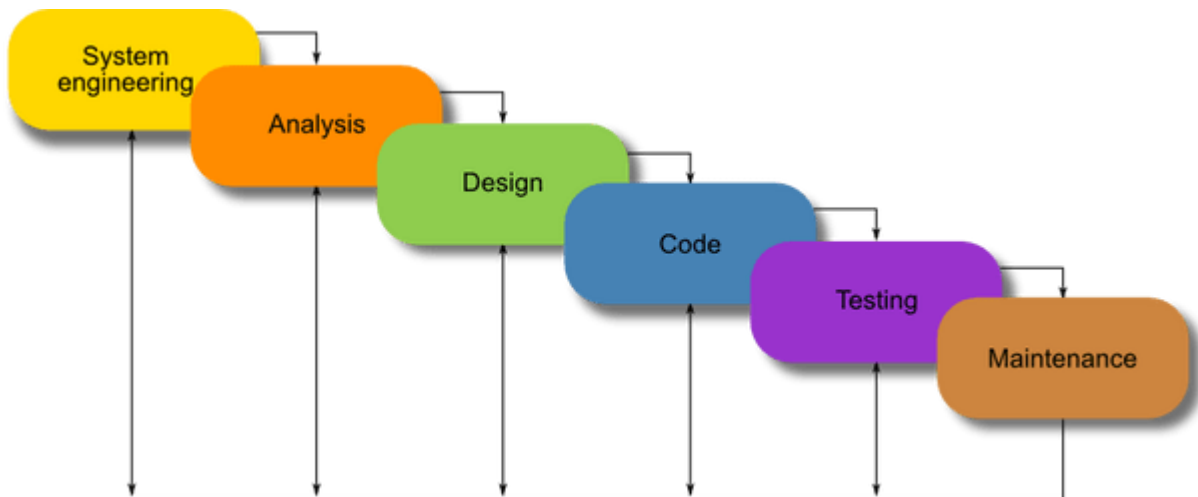


Рисунок 2.13

Для створення нової гілки і переходу на неї, необхідно використовувати команду `git checkout` з ключем `-b` (рисунок 2.14):

```
MINGW64:/d/GIT_PRACTICE
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git checkout -b design
Switched to a new branch 'design'
```

Рисунок 2.14

Виконання команди `checkout` з ключем `-b` є скороченням для двох команд:

```
git branch design # створення нової гілки
git checkout design # перехід на нову гілку
```

Після створення, нова гілка вказує на той же коміт, що і гілка `master`, оскільки ніяких змін в гілці `design` ще не було зафіксовано (рисунок 2.15).




```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (design)
$ git log
commit d53a9ee238e95d8cf89d2ab6a056550d718dd020
Author: Andrii Kopp <kopp93@gmail.com>
Date: Mon Aug 6 11:55:57 2018 +0300

    Lab 2: UML models
```

Рисунок 2.15

Звичайно ж, під час роботи над прототипом створюваної системи будуть зроблені і зафіксовані деякі зміни. Наприклад, з'явилася необхідність використовувати заголовок першого рівня для повідомлення, що виводиться на сторінці index.html (рисунок 2.16):



```
index.html - Notepad
File Edit Format View Help
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<h1>
<div id="hello"></div>
</h1>
<script src="script.js"></script>
</body>
</html>
```

Рисунок 2.16

Внесені зміни необхідно зафіксувати. Після коммітів гілка design, в якій виконувалася робота, буде вказувати вже на останній комміт, пов'язаний з додаванням заголовка в файл index.html, тобто зрушиться вперед, щодо гілки master (рисунок 2.17).

A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The prompt is 'lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (design)'. The user has entered '\$ git log'. The output shows two commits. The first commit has hash '1ae63c2d58d9735318f91ab7f36b2ff40465bc62', author 'Andrii Kopp <kopp93@gmail.com>', and date 'Mon Aug 6 12:40:09 2018 +0300'. Its message is 'Lab 2: H1 for index message'. The second commit has hash 'd53a9ee238e95d8cf89d2ab6a056550d718dd020', the same author, and date 'Mon Aug 6 11:55:57 2018 +0300'. Its message is 'Lab 2: UML models'.

Рисунок 2.17

Припустимо, що з якоїсь причини треба було змінити колір виведеного повідомлення на сторінці index.html на червоний. Причому, зробити це необхідно в такий спосіб:

1. Переконавшись, що всі зміни на гілці design зафіксовані, переключитися на гілку master (рисунок 2.18):

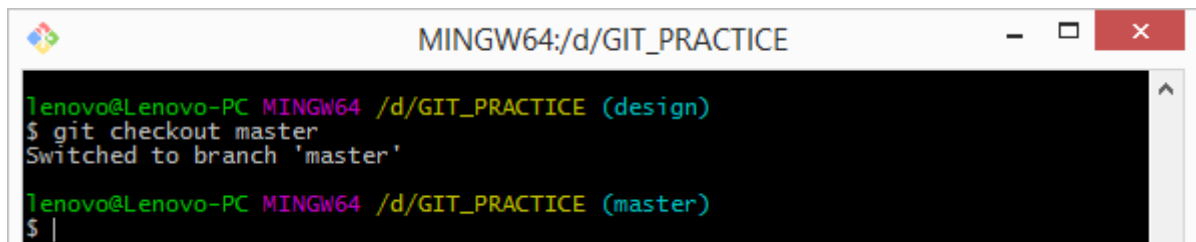
A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The prompt is 'lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (design)'. The user has entered '\$ git checkout master'. The output is 'Switched to branch 'master''. The prompt now shows '(master)'. The user has entered '\$ |'.

Рисунок 2.18

2. Створити гілку, в якій буде виконуватися робота (рисунок 2.19):

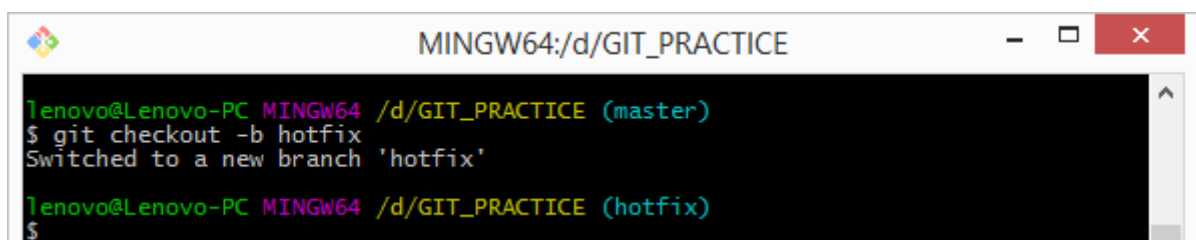
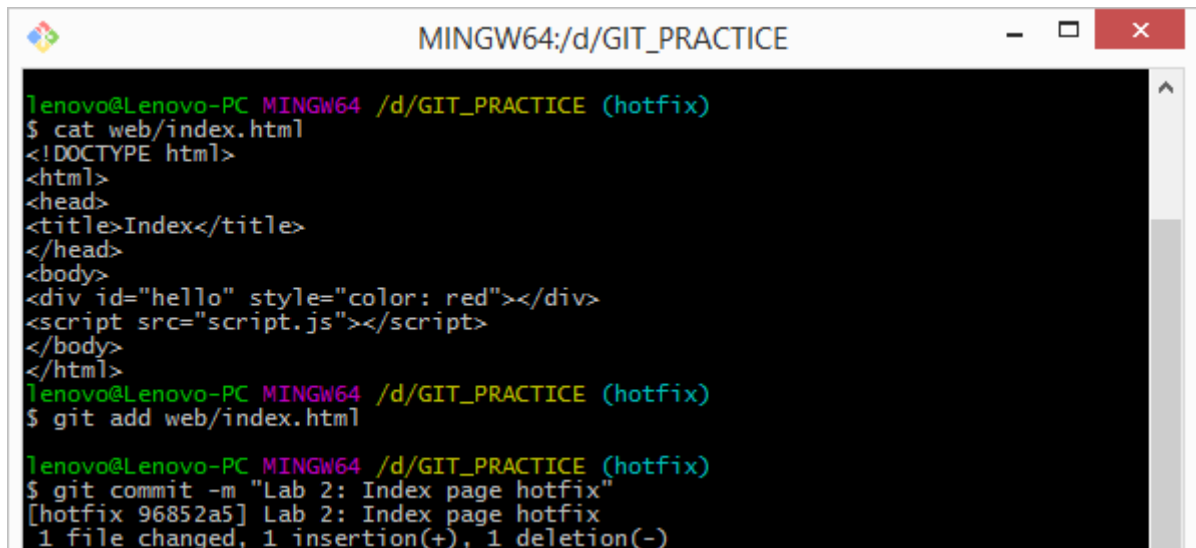
A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The prompt is 'lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)'. The user has entered '\$ git checkout -b hotfix'. The output is 'Switched to a new branch 'hotfix''. The prompt now shows '(hotfix)'. The user has entered '\$'.

Рисунок 2.19

3. Внести необхідні зміни в файл index.html і зробити комміт з відповідним коментарем (рисунок 2.20):



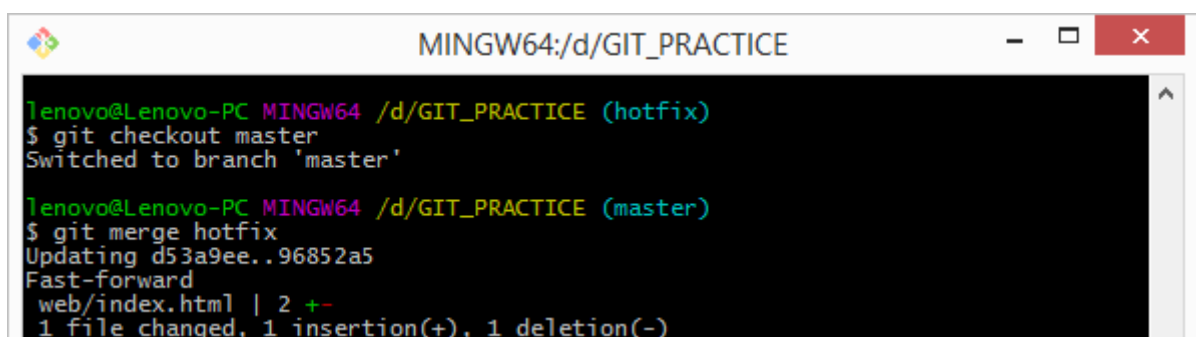
```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ cat web/index.html
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello" style="color: red"></div>
<script src="script.js"></script>
</body>
</html>
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ git add web/index.html

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ git commit -m "Lab 2: Index page hotfix"
[hotfix 96852a5] Lab 2: Index page hotfix
1 file changed, 1 insertion(+), 1 deletion(-)
```

Рисунок 2.20

4. Злити зміни в гілку master, щоб включити їх до проекту, за допомогою команди git merge (рисунок 2.21):



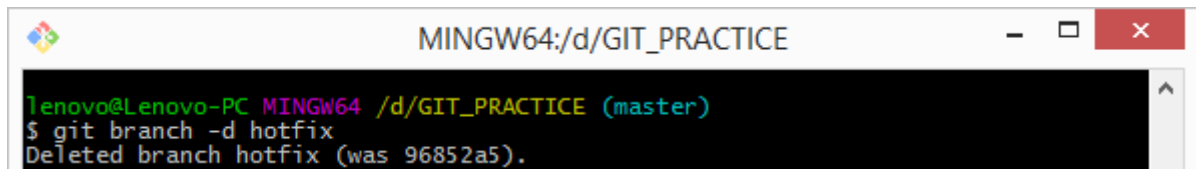
```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (hotfix)
$ git checkout master
Switched to branch 'master'

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git merge hotfix
Updating d53a9ee..96852a5
Fast-forward
 web/index.html | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

Рисунок 2.21

5. Видалити непотрібну більше гілку hotfix (гілка master після злиття вказує на те ж місце, так як система Git просто перемістила показник вперед через відсутність розбіжних змін, які потрібно було б зливати воедино) за допомогою команди git branch з опцією -d (рисунок 2.22):



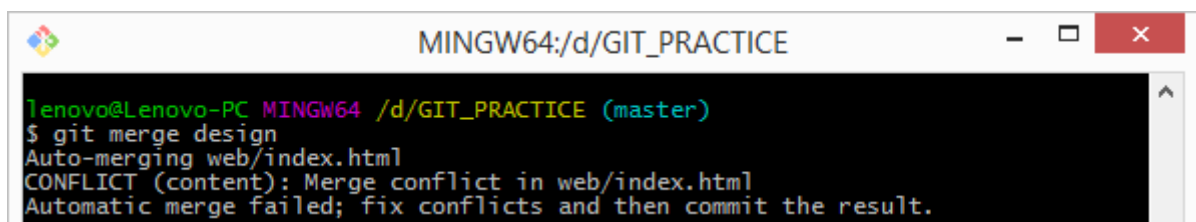
```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git branch -d hotfix
Deleted branch hotfix (was 96852a5).
```

Рисунок 2.22

Після того як проблема вирішена, можна повернутися назад до гілки design і продовжити роботу. Однак необхідно пам'ятати, що робота, зроблена на гілці hotfix, не включена в комміти на гілці design. Якщо необхідно, гілку master можна злити в гілку design допомогою команди `git merge master`. Крім того, інтеграцію змін можна відкласти до тих пір, поки зміни на гілці design НЕ буде вирішено включити в основну гілку проекту master.

2.4.2 Конфлікти при злитті

Процес злиття гілок не завжди проходить гладко. У нашому випадку рішення задачі в гілці design змінює той же файл (`index.html`), що і гілка hotfix, в результаті чого буде отримано конфлікт злиття (рисунок 2.23):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git merge design
Auto-merging web/index.html
CONFLICT (content): Merge conflict in web/index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Рисунок 2.23

Система Git не створить новий комміт для злиття гілок, а призупинить цей процес до тих пір, поки користувач не вирішить конфлікт. Для перегляду файлів, які не пройшли злиття, необхідно виконати команду `git status` (рисунок 2.24):



```
MINGW64:/d/GIT_PRACTICE
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master|MERGING)
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

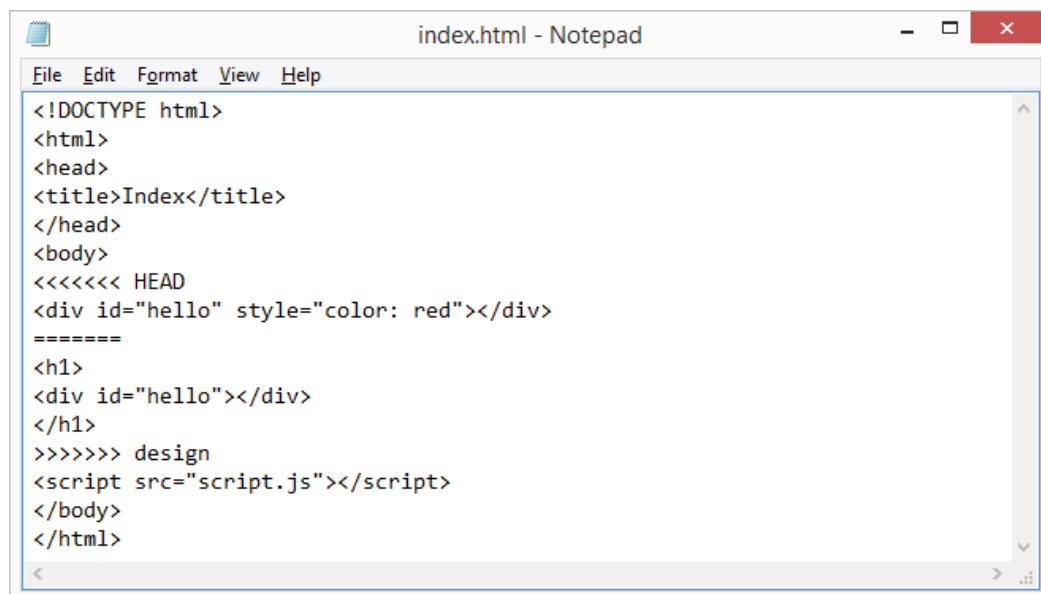
Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   web/index.html

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

Рисунок 2.24

Система Git додає стандартні маркери до файлів (тут це index.html), які мають конфлікт (unmerged), так що можна відкрити такі файли вручну і вирішити ці конфлікти. Файл index.html буде виглядати наступним чином (рисунок 2.25):



```
index.html - Notepad
File Edit Format View Help
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<<<<<<< HEAD
<div id="hello" style="color: red"></div>
=====
<h1>
<div id="hello"></div>
</h1>
>>>>>> design
<script src="script.js"></script>
</body>
</html>
```

Рисунок 2.25

У файлі index.html все, що вище ===== це версія з HEAD (гілка master, так як саме на ній була виконана команда merge). Все, що знаходиться нижче – версія в гілці design. Для вирішення конфлікту необхідно або вибрати одну з цих частин, або якось об'єднати вміст на свій

розсуд. Наприклад, конфлікт може бути вирішений таким чином (рисунок 2.25):

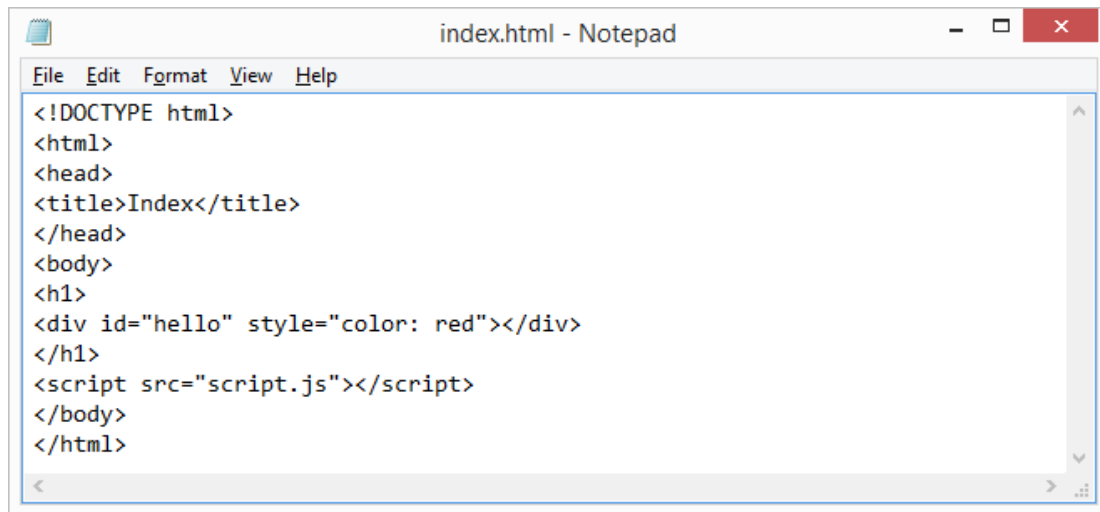


Рисунок 2.25

Після вирішення конфліктів, для кожного конфліктного файлу необхідно виконати команду `git add`. Індексуювання для системи Git буде означати, що всі конфлікти дозволені (рисунок 2.26):

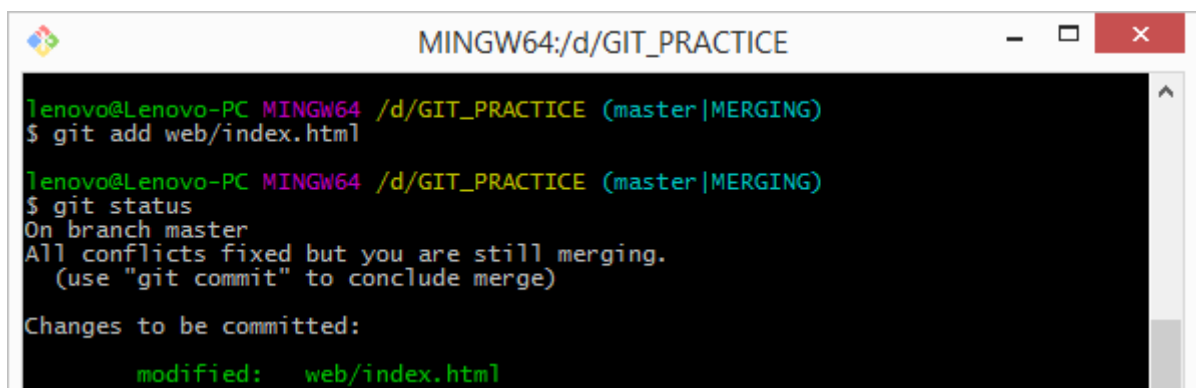
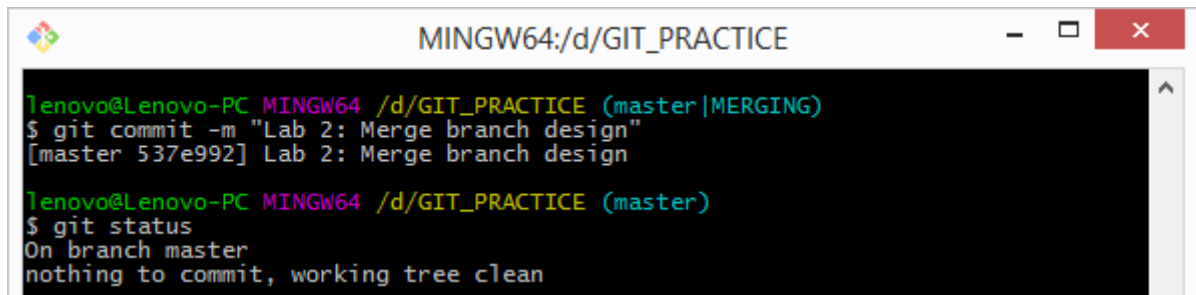


Рисунок 2.26

Упевнившись, що всі файли, що мали конфлікти, були проіндексовані, можна виконати `git commit` (рисунок 2.27):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master|MERGING)
$ git commit -m "Lab 2: Merge branch design"
[master 537e992] Lab 2: Merge branch design

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 2.27

У коментарі до комітів рекомендується вказувати інформацію про те, як було вирішено конфлікт, якщо це не очевидно і може бути корисно для інших користувачів в майбутньому.

2.6 Питання для самоперевірки

15. За допомогою якої команди можна створити нову гілку в системі Git?
16. Яка команда в системі Git використовується для переходу між гілками?
17. Для чого використовується ключ -b команди git checkout?
18. Для чого використовується команда git merge?
19. Яким чином можна видалити гілку в системі Git?
20. З яких причин можуть виникнути конфлікти при злитті гілок в системі Git?
21. Яким чином можна переглянути список файлів, які не пройшли злиття?
22. Як система Git «допомагає» вирішити конфлікти в файлах, які не пройшли злиття?
23. Що необхідно зробити для завершення процесу злиття гілок після того, як всі конфлікти були дозволені?
24. Як уникнути виникнення конфліктів при злитті гілок?