ЛАБОРАТОРНА РОБОТА №3
ОСНОВИ GIT

3.1 Мета роботи

Вивчити основні команди по роботі з локальним репозиторієм GIT.

3.2 Основні теоретичні відомості

ОСНОВНІ ХАРАКТЕРИСТИКИ GIT

Переваги:

1. Надійна система порівняння ревізій та перевірки коректності даних, заснована на алгоритмі хешируванния SHA1 (Secure Hash Algorithm 1).
2. Гнучка система розгалуження проектів і злиття гілок між собою.
3. Наявність локального репозиторію,
4. Висока продуктивність і швидкість роботи.
5. Зручний і інтуїтивно зрозумілий набір команд.
6. Безліч графічних оболонок, що дозволяють швидко і якісно вести роботи з Git'ом.
7. Можливість робити контрольні точки.
8. Широка розповсюдженність, легка доступність і якісна документація.
9. Гнучкість системи.
10. Універсальний мережевий доступ з використанням протоколів http, ftp, rsync, ssh та ін.

Недоліки

1. Unix - орієнтованість.
2. Можливі (але надзвичайно низькі) збіги хеш - коду відмінних за змістом ревізій.
3. Не відстежуються зміни окремих файлів
4. При початковому (першому) створенні сховища та синхронізації його з іншими розробниками, буде потрібно досить тривалий час для скачування даних, особливо, якщо проект великий, так як потрібно скопіювати на локальний комп'ютер весь репозиторій.

АРХІТЕКТУРА GIT

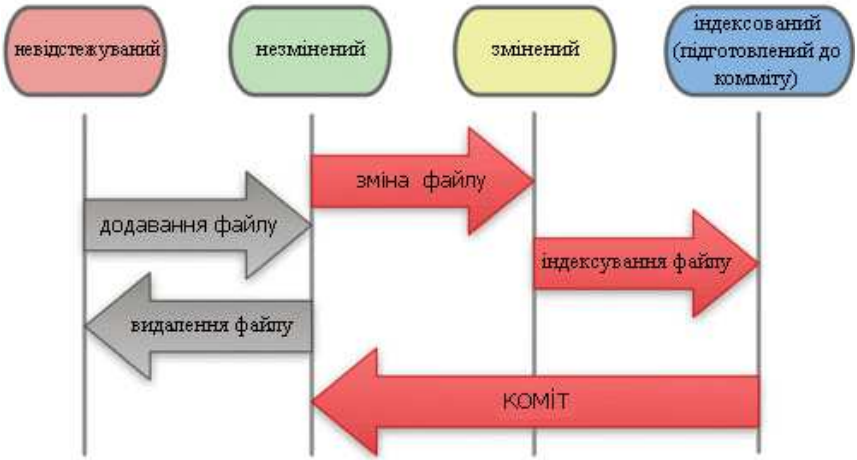
• SHA1 (Secure Hash Algorithm 1)

- Об’єкти Git
  - Blob (Binary Large Object)
  - Дерево (Tree)
  - Reference
  - Гілка (Head, Branch)
  - Тег (tag) /мітка
  - light tag
  - annotated tag

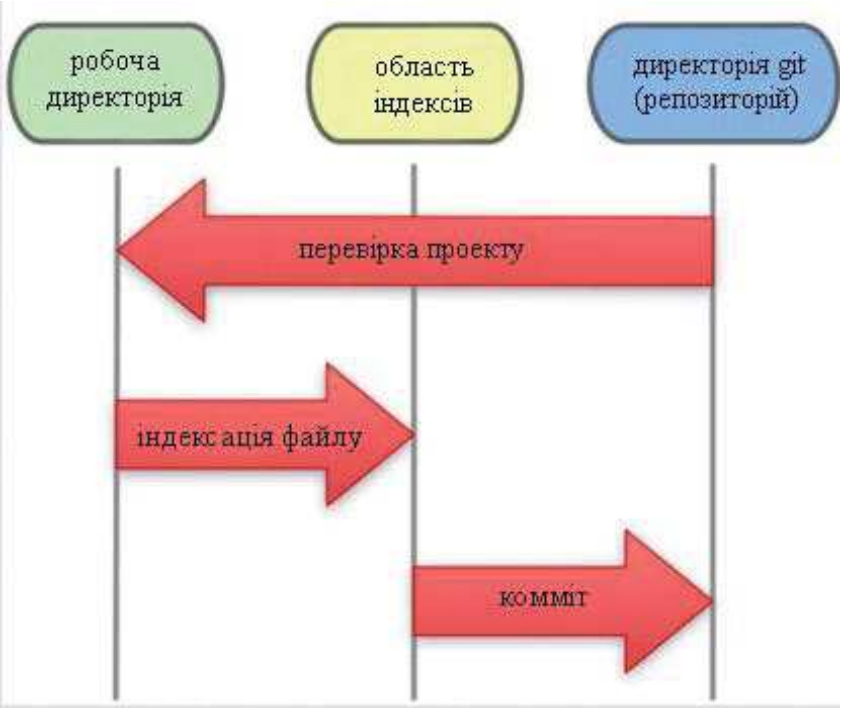
ТРИ СТАНИ ФАЙЛУ

- зафиксований
- змінений
- підготовлений

ЖИТТЕВИЙ ЦИКЛ ФАЙЛУ



GIT ПРОЕКТ



РОЗГАЛУЖЕННЯ

- git branch
- git branch <ім’я_гілки>
  - -d < ім’я_гілки >
  - -D < ім’я_гілки >
  - -m < ім’я_гілки >
  - --contains <ссылка_на_коммит>
- git checkout <имя_ветви>
  - -f <ім’я_гілки> ‘
  - -b <ім’я_гілки>
- git merge <гілка>
- git merge <ім’я_віддаленого_репозиторія>/<гілка_віддаленого_репозиторія>

3.3 Завдання на лабораторну роботу

3.3.1 Ознайомитися з теоретичними відомостями щодо основних команд GIT.

3.3.2 Виконати свій варіант завдання, аналогічно нульовому варіанту:

Варіант 0 (див. приклад)

Ініціалізувати каталог з майбутнім проектом в git. Створити файл .gitignore з описом ігнорованих файлів (*. [oa], * ~).

Керівник проекту вносить свої дані в git, створює файл в якому прописує структуру майбутнього проекту, з коментарями щодо завдань іншим членам команди. Виконує комміт для фіксації внесених змін.

Перший член команди проекту вносить свої дані в git і створює свою гілку. Відкриває файл проекту, виконує завдання прописане керівником команди. Виконує комміт для фіксації внесених змін і повертається в гілку master.

Другий член команди проекту вносить свої дані в git і створює свою гілку, відкриває файл проекту, виконує завдання прописане керівником команди. Виконує комміт для фіксації внесених змін і повертається в гілку master.

Керівник проекту зливає гілки і перевіряє результат.

Відобразити всі результати роботи за допомогою gitk.

Варіант 1

Ініціалізувати каталог з майбутнім проектом в git. Створити файл .gitignore з описом ігнорованих файлів (*. [oa], * ~). Користувач вводить свої даних, створює папку проекту, створює в ній файл з 2 завданнями. Виконує комміт для фіксації внесених змін. Створює нову гілку, відкриває файл і виконує першу задачу, виконує комміт. Створює ще одну підгілку, відкриває файл і виконує друге завдання, виконує комміт. Повертається в гілку master. Переглядає історію змін, отримує хеш всіх змін і по хеш відкочується до першого комміту. Відобразити всі результати роботи за допомогою gitk.

Варіант 2

Ініціалізувати каталог з майбутнім проектом в git. Створити файл .gitignore з описом ігнорованих файлів (*. [oa], * ~). Користувач вводить свої даних, створює папку проекту, створює в ній файл із завданнями. Виконує комміт для фіксації внесених змін. Створює

нову гілку, відкриває файл і виконує завдання, виконує комміт. Повертається в гілку майстер, додає аліаси для branch, checkout, status, history. За допомогою алиасів переглядає всі гілки проекту, його поточний статус та історію коммітів.

Варіант 3

Ініціалізувати папку з майбутнім проектом в git. Створити файл .gitignore з описом ігнорованих файлів (*. [oa], * ~). Користувач вводить свої дані, створює папку проекту, створює в ній файл із завданнями. Виконує комміт для фіксації внесених змін. Створює нову гілку, відкриває файл і виконує завдання, виконує комміт. Повертається в гілку master. Переглядає історію коммітів використовуючи одностроковий формат. Вносить зміни у файл, виконує комміт. Переглядає історію, використовуючи різні формати контролю відображення записів.

Варіант 4

Ініціалізувати папку з майбутнім проектом в git. Створити файл .gitignore з описом ігнорованих файлів (*. [oa], * ~).

Керівник проекту вносить свої дані в git, створює кілька файлів в яких прописує структуру майбутнього проекту, з коментарями щодо завдань іншим членам команди. Виконує комміт для фіксації внесених змін.

Перший член команди проекту вносить свої дані в git і створює свою гілку. Відкриває проекти і виконує завдання (у декількох файлах) прописане керівником. Виконує комміт для фіксації внесених змін і повертається в гілку master.

Другий член команди проекту вносить свої дані в git і створює свою гілку, відкриває проект проекту, переглядає його і видаляє один з файлів. Виконує комміт для фіксації внесених змін і повертається в гілку master.

Керівник проекту зливає гілки, створює новий файл у проекті, вносить до нього зміни і виконує Ком. Переглядає історію коммітів і виконує останні зміни. Відобразити всі результати роботи за допомогою gitk.

3.3.3 Виконати аналіз отриманих результатів.

3.3.4 Оформити звіт та відповісти на контрольні питання.

2.3 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Результати виконання свого варіанту завдання, аналогічно нульовому варіанту

Виконання роботи (варіант 0)

Ініціалізувати каталог з майбутнім проектом в git. Створити файл .gitignore з описом ігнорованих файлів (*. [oa], * ~).

Керівник проекту вносить свої дані в git, створює файл в якому прописує структуру майбутнього проекту, з коментарями щодо завдань іншим членам команди. Виконує комміт для фіксації внесених змін.

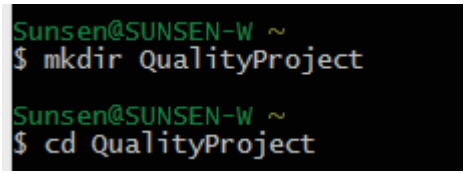
Перший член команди проекту вносить свої дані в git і створює свою гілку. Відкриває файл проекту, виконує завдання прописане керівником команди. Виконує комміт для фіксації внесених змін і повертається в гілку master.

Другий член команди проекту вносить свої дані в git і створює свою гілку, відкриває файл проекту, виконує завдання прописане керівником команди. Виконує комміт для фіксації внесених змін і повертається в гілку master.

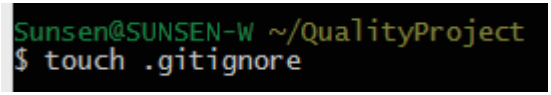
Керівник проекту зливає гілки і перевіряє результат.

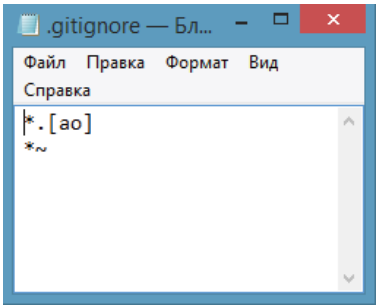
Відобразити всі результати роботи за допомогою gitk.

1. Створюємо каталог з майбутнім проектом та переходимо у нього



2. Створюємо файл .gitignore з описом ігнорованих файлів





3. Створюємо репозиторій для проекта

```
Sunsen@SUNSEN-W ~/QualityProject
$ git init
Initialized empty Git repository in c:/Users/Sunsen/QualityProject/.git/
```

4. Додаємо у нього файли

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git add .
```

5. Робимо коміт

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git commit -m "First commit"
[master (root-commit) 56c1c0f] First commit
 1 file changed, 2 insertions(+)
 create mode 100644 .gitignore
```

6. Перший користувач(керівник проекту) виконує наступні дії:

6.1 Вносить свої дані (ім’я и emil)

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git config --global user.name "teamLeader"

Sunsen@SUNSEN-W ~/QualityProject (master)
$ git config --global user.email "teamleader.en@gmail.com"
```

6.2 Створює файл з описом структури проекту

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ touch quality.cpp
```

quality — Блокнот

Файл Правка Формат Вид Справка

```
#include "mainwindow.h"
#include <cmath>
#include "ui_mainwindow.h"

int n1, n2, m, N1, N2, N, R;

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

//задания для partyOne
//написать методы для подсчета количества строк кода и пустых строк



//задания для partyTwo
//написать методы для подсчета количества комментариев и строк кода(общее кол-во строк – комментарии – пустые строки)
```

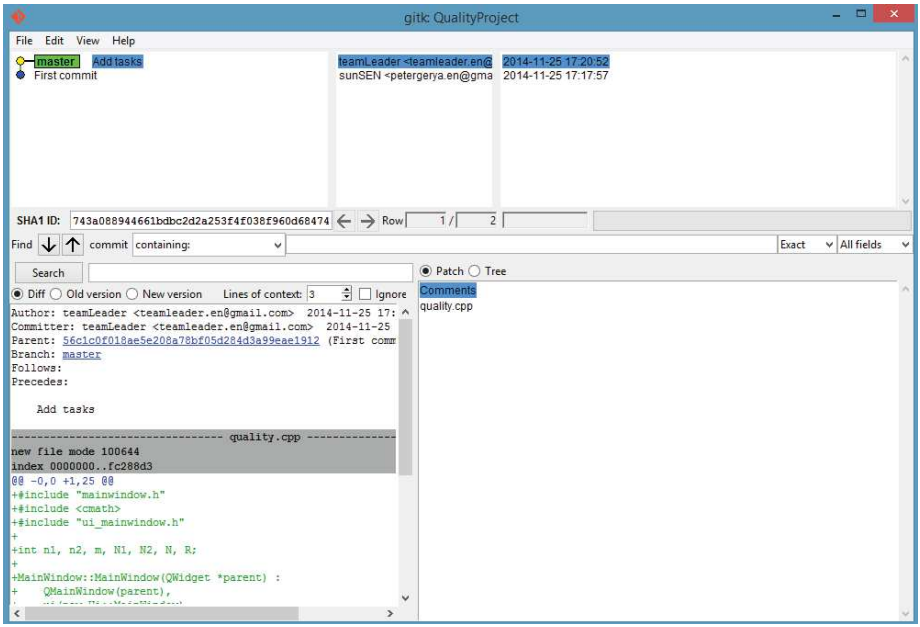
6.3 Додає усі файли до репозиторію

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git add .
```

6.4 Виконує коміт

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git commit -m "Add tasks"
[master 743a088] Add tasks
 1 file changed, 25 insertions(+)
 create mode 100644 quality.cpp
```

При виконанні команди gitk отримуємо:



7. Перший член проекту виконує наступні дії:

7.1 Вносить свої дані (ім’я та emil)

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git config --global user.name "crewmanOne"

Sunsen@SUNSEN-W ~/QualityProject (master)
$ git config --global user.email "crewman.one@gmail.com"
```

7.2 Створює свою гілку та переходить у неї

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git branch crewmanOne

Sunsen@SUNSEN-W ~/QualityProject (master)
$ git checkout crewmanOne
Switched to branch 'crewmanOne'
```

7.3 Виконує своє завдання, яке прописано керівником команди

quality.cpp | <Выберите символ> | # Строка: 1, Столбец: 1

```
18
19  //задания для crewmanOne
20  //написать методы для подсчета количества строк кода и пустых строк
21
22  //количество строк кода
23  void MainWindow::on_SLOC_clicked()
24  {
25      QString str=ui->plainTextEdit->toPlainText();
26          int n=str.count("\n");
27          ui->label->setText(QString::number(n+1));
28  }
29
30  // количество пустых строк
31  void MainWindow::on_EmptyLines_clicked()
32  {
33      QString str=ui->plainTextEdit->toPlainText();
34          int n=str.count("\n\n");
35          ui->label_2->setText(QString::number(n));
36  }
```

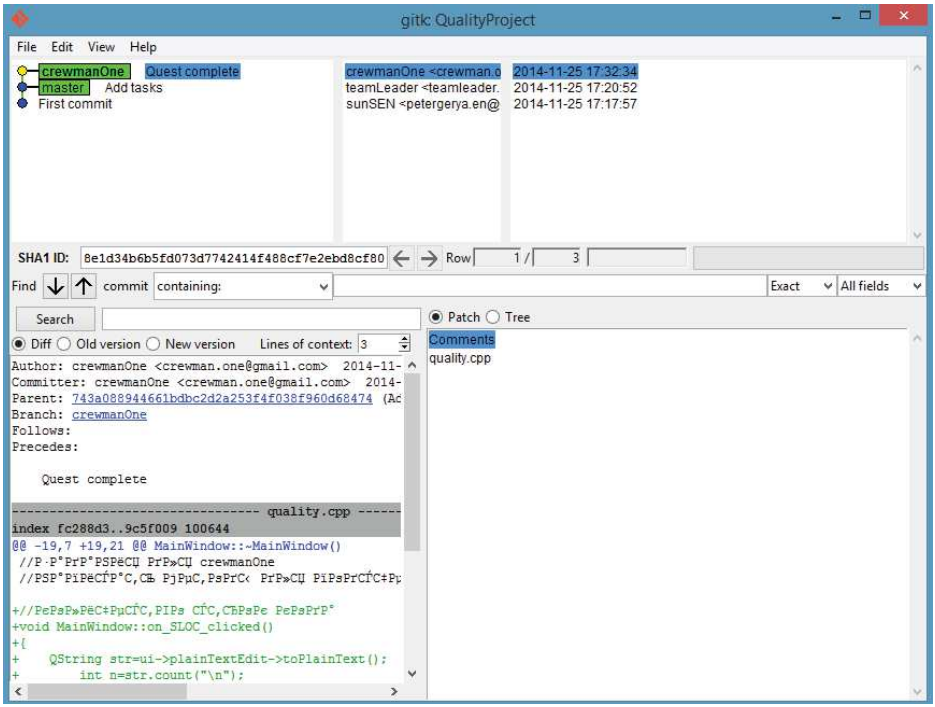
7.4 Додає усі файли до репозиторію

```
Sunsen@SUNSEN-W ~/QualityProject (crewmanOne)
$ git add .
```

7.5 Робить коміт

```
Sunsen@SUNSEN-W ~/QualityProject (crewmanOne)
$ git commit -m "Quest complete"
[crewmanOne 8e1d34b] Quest complete
 1 file changed, 14 insertions(+)
```

При виконанні команди gitk отримуємо:



7.6 Повертається у гілку майстра

```
Sunsen@SUNSEN-W ~/QualityProject (crewmanOne)
$ git checkout master
Switched to branch 'master'
```

8. Другий член проекту виконує наступні дії:

8.1 Вносить свої дані (ім’я та emil)

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git config --global user.name "crewmanTwo"

Sunsen@SUNSEN-W ~/QualityProject (master)
$ git config --global user.email "crewman.two@gmail.com"
```

8.2 Дивиться назви усіх існуючих гілок, щоб визначитись з назвою своєї гілки

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git branch
  crewmanOne
* master
```

8.3 Створює свою гілку та переходить у неї

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git branch crewmanTwo

Sunsen@SUNSEN-W ~/QualityProject (master)
$ git checkout crewmanTwo
Switched to branch 'crewmanTwo'
```

8.4 Виконує своє завдання прописане керівником проекту

quality.cpp <Выберите символ> Строка: 1, Столбец: 1

```
23
24  //задания для crewmanTwo
25  //написать методы для подсчета количества комментариев и строк кода(общее кол-во строк - коммент
26
27  //количество комментариев
28  void MainWindow::on_Comments_clicked()
29  {
30      QString str=ui->plainTextEdit->toPlainText();
31          int n=str.count("\n//");
32          ui->label_3->setText(QString::number(n));
33          int k=str.count("\n");
34          ui->label_5->setText(QString::number(((float)n)/(k+1)*100)+"%");
35  }
36
37  //количество строк кода(общее кол-во строк - комментарии - пустые строки)
38
39  void MainWindow::on_OnlyCode_clicked()
40  {
41      QString str=ui->plainTextEdit->toPlainText();
42      int n1=str.count("\n//");
43      int n2=str.count("\n\n");
44      int n=str.count("\n");
45      ui->label_4->setText(QString::number(n-n1-n2+1));
46
47  }
48
```

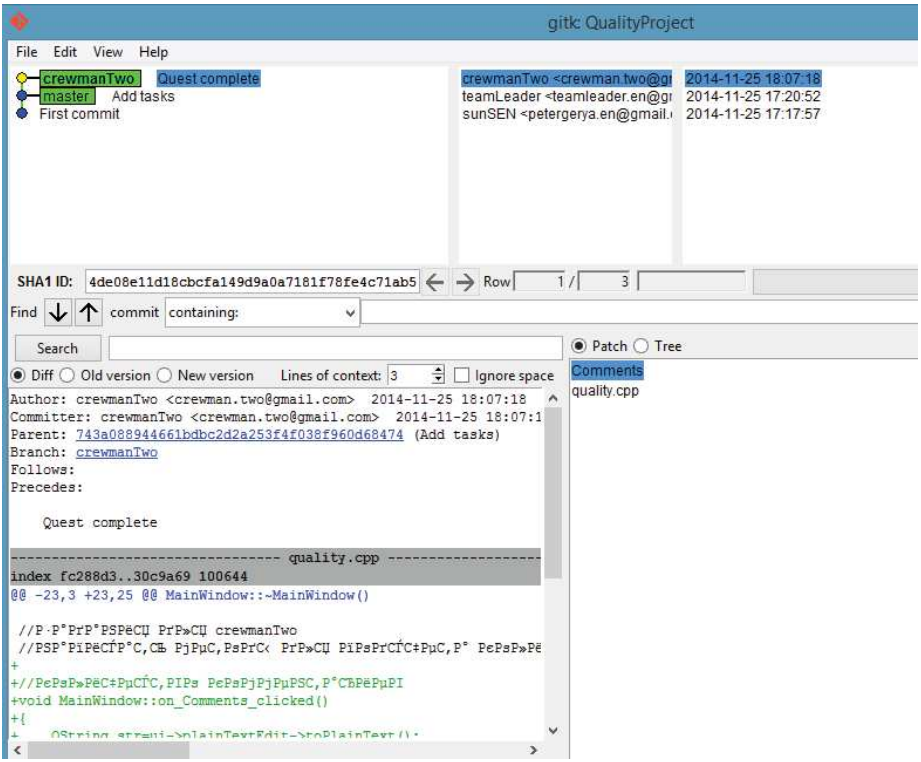
8.5 Додає усі файли до репозиторію

```
Sunsen@SUNSEN-W ~/QualityProject (crewmanTwo)
$ git add .
```

8.6 Виконує коміт

```
Sunsen@SUNSEN-W ~/QualityProject (crewmanTwo)
$ git commit -m "Quest complete"
[crewmanTwo 4de08e1] Quest complete
 1 file changed, 22 insertions(+)
```

При виконанні команди gitk отримуємо:



8.7 Повертається у гілку майстра

```
Sunsen@SUNSEN-W ~/QualityProject (crewmanTwo)
$ git checkout master
Switched to branch 'master'
```

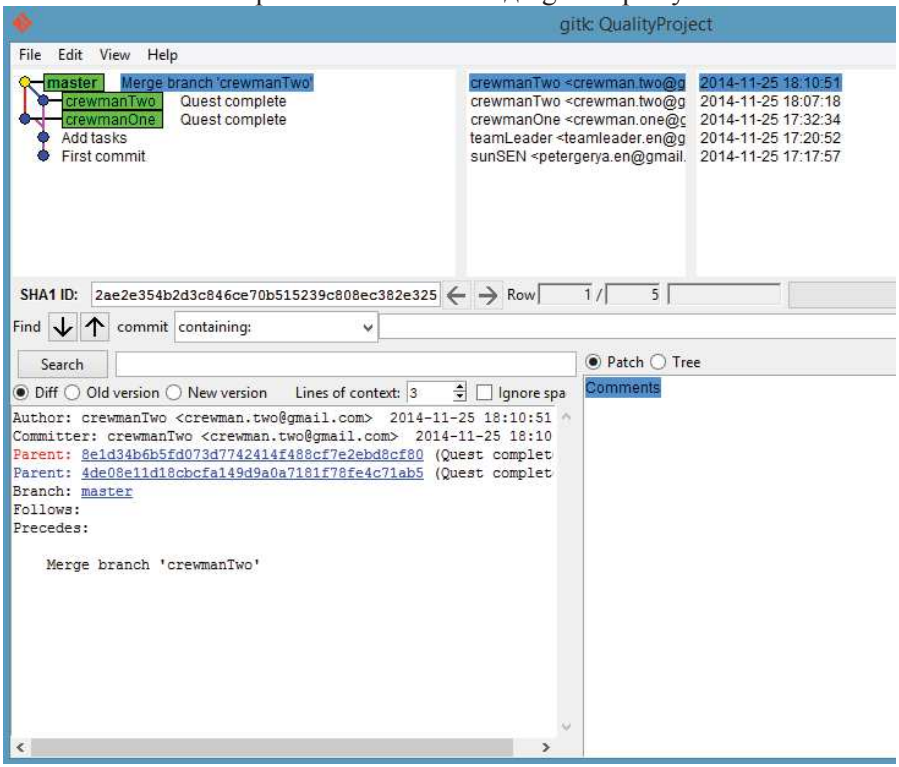
9. Керівник проекту зливає усі гілки разом

```
Sunsen@SUNSEN-W ~/QualityProject (master)
$ git merge crewmanOne
Updating 743a088..8e1d34b
Fast-forward
 quality.cpp | 14 ++++++++++++++
 1 file changed, 14 insertions(+)

Sunsen@SUNSEN-W ~/QualityProject (master)
$ git merge crewmanTwo
Auto-merging quality.cpp
Merge made by the 'recursive' strategy.
 quality.cpp | 22 ++++++++++++++++++++++
 1 file changed, 22 insertions(+)
```

10. Перевіряє результат

При виконанні команди gitk отримуємо:



Файл quality.cpp має вигляд:

```
MainWindow::~MainWindow()
{
    delete ui;
}

//задания для crewmanOne
//написать методы для подсчета количества строк кода и пустых строк

//количество строк кода
void MainWindow::on_SLOC_clicked()
{
    QString str=ui->plainTextEdit->toPlainText();
        int n=str.count("\n");
        ui->label->setText(QString::number(n+1));
}

// количество пустых строк
void MainWindow::on_EmptyLines_clicked()
{
    QString str=ui->plainTextEdit->toPlainText();
        int n=str.count("\n\n");
        ui->label_2->setText(QString::number(n));
}

//задания для crewmanTwo
//написать методы для подсчета количества комментариев и строк кода(общее кол-во строк - комментарии - пустые строки)

//количество комментариев
void MainWindow::on_Comments_clicked()
{
    QString str=ui->plainTextEdit->toPlainText();
        int n=str.count("\n//");
        ui->label_3->setText(QString::number(n));
        int k=str.count("\n");
        ui->label_5->setText(QString::number(((float)n)/(k+1)*100)+"%");
}

//количество строк кода(общее кол-во строк - комментарии - пустые строки)

void MainWindow::on_OnlyCode_clicked()
{
    QString str=ui->plainTextEdit->toPlainText();
    int n1=str.count("\n//");
    int n2=str.count("\n\n");
    int n=str.count("\n");
    ui->label_4->setText(QString::number(n-n1-n2+1));

}
```

3.4.3. Аналіз отриманих результатів.

3.4.5. Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

3.5 Контрольні запитання

3.5.1. Як створити Git-репозиторій?

3.5.2. Як додати під версійний контроль усі файли?

3.5.3. У скількох станах можуть знаходиться файли в репозиторії Git?

3.5.4. Яка команда використовується для визначення стану файлів в репозиторії?

3.5.5. Для чого використовується файл .gitignore?

3.5.6. Якою командою переглянути не проіндексовані зміни, зроблені після останнього коміту?

3.5.7. Якою командою виконується фіксація змін?

3.5.8. Як видалити файл з Git-репозиторію?

3.5.9. Як перемістити файл у Git-репозиторії?

3.5.10. Як переглянути історію комітів?

3.5.11. Як змінити останній комміт?

3.5.12. Яка команда створить нову гілку в Git-репозиторії?

3.5.13. Як об’єднати гілку А та Б?