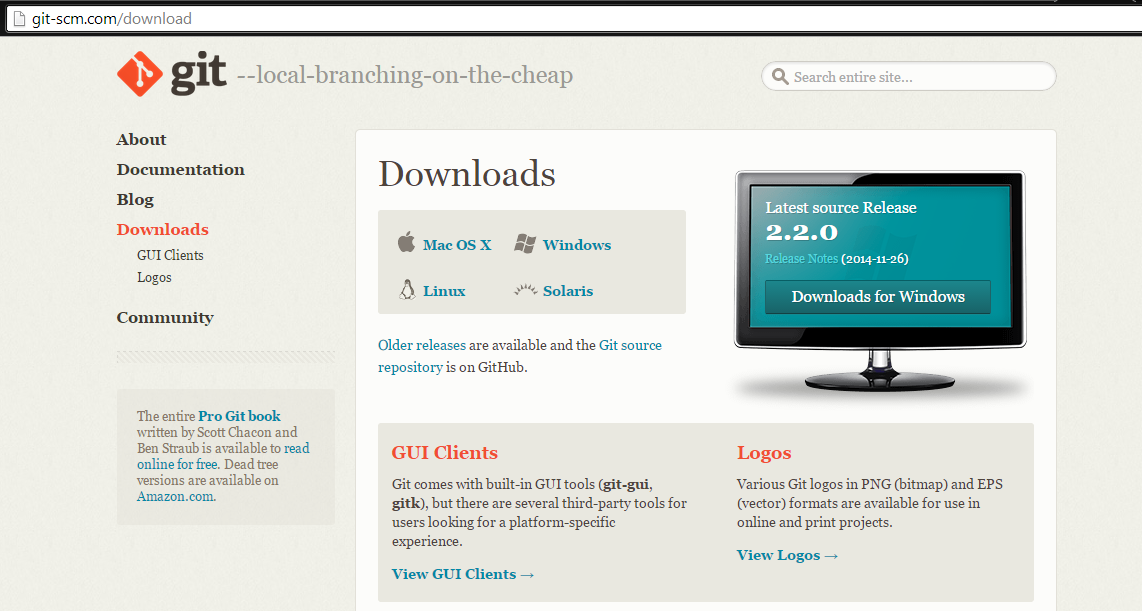
**Встановлення Git**

Спочатку необхідно перейти на сайт git-scm.com і завантажити git для своєї операційної системи:

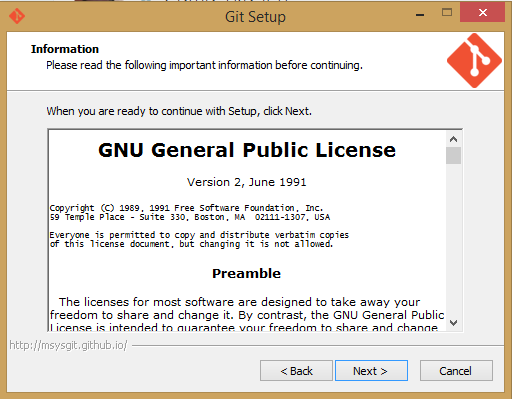


Встановлення Git для ОС Windows

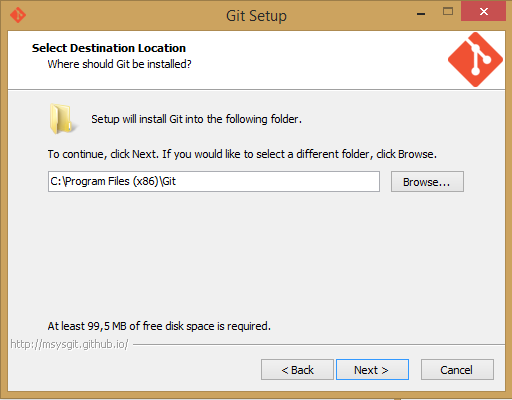
Запускаємо завантажений файл “Git-1.9.4-preview20140929.exe”



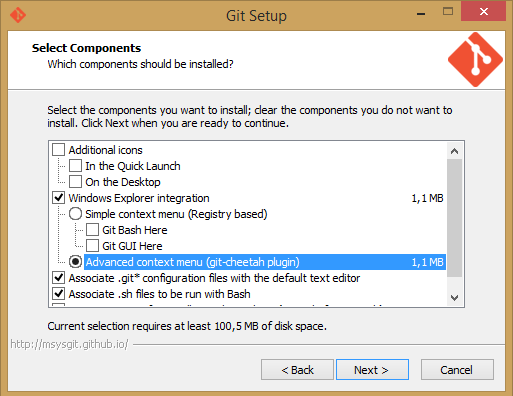
Натискаємо кнопку “Next” і ознайомлюємося з ліцензією



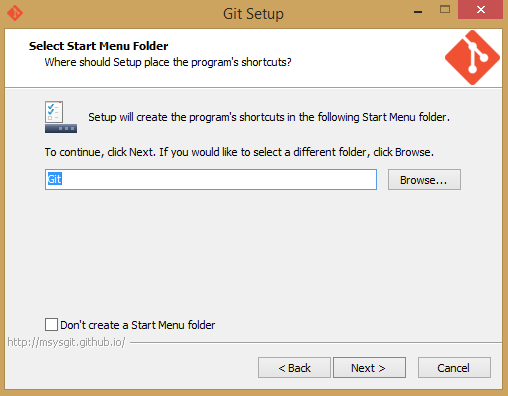
Натискаємо кнопку “Next” і вказуємо шлях встановлення



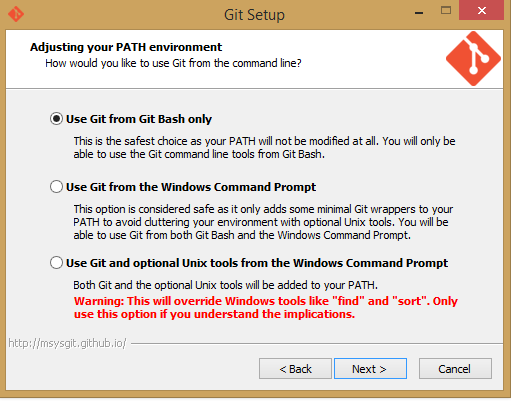
Натискаємо кнопку “Next”, обираємо необхідні елементи (радимо залишити все як було)



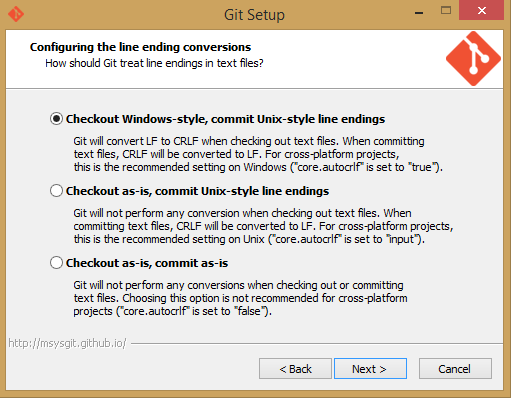
Натискаємо кнопку “Next” і обираємо каталог в якому будуть зберігатись проекти для роботи з git



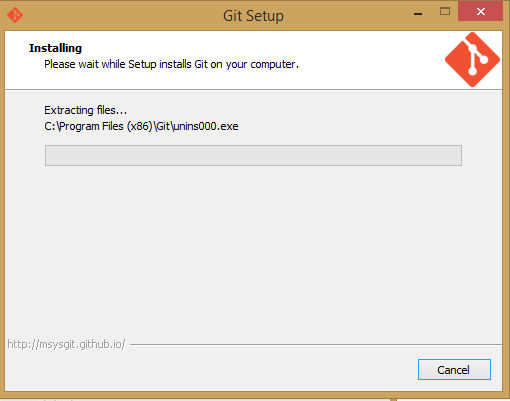
Натискаємо кнопку “Next” і обираємо зовнішній вигляд командного рядку git



Натискаємо кнопку “Next” і обираємо як будуть завершуватись файли git



І залишається дочекатись, коли програма завершить встановлення.



Натиснення кнопки «Finish» закінчує встановлення.



Встановлення завершено.

Встановлення Git для ОС Linux

Якщо у Вас Fedora:

$ yum install git-core

Якщо у Вас дистрибутив, заснований на Debian, наприклад, Ubuntu:

$ apt-get install git

**Варіанти завдань**

**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.

**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.

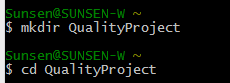
Перший член команди проекту вносить свої дані в git і створює свою гілку. Відкриває файл проекту, виконує завдання прописане керівником команди. Виконує комміт для фіксації внесених змін і повертається в гілку master.

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.

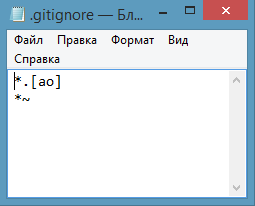
Відобразити всі результати роботи за допомогою gitk.

1. Створюємо каталог з майбутнім проектом та переходимо у нього



1. Створюємо файл .gitignore з описом ігнорованих файлів





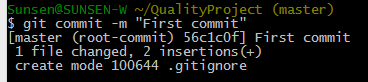
1. Створюємо репозиторій для проекта



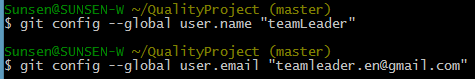
1. Додаємо у нього файли



1. Робимо коміт

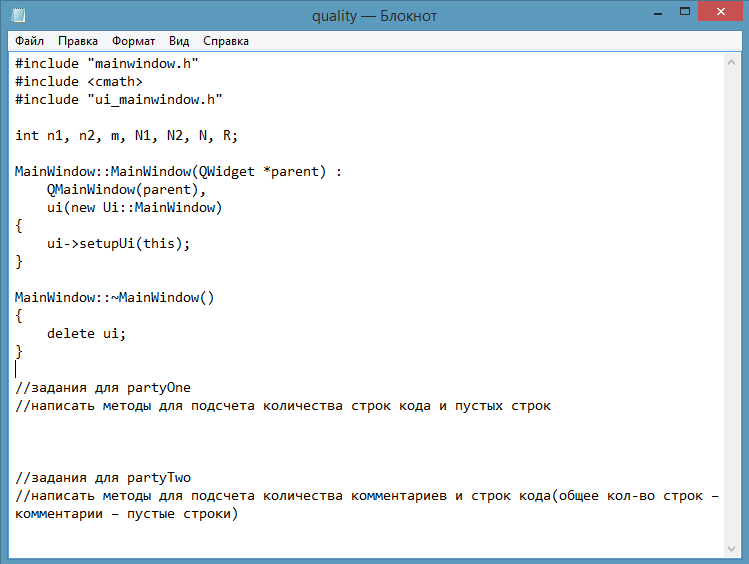


1. Перший користувач(керівник проекту) виконує наступні дії:
   1. Вносить свої дані (ім’я и emil)



* 1. Створює файл з описом структури проекту

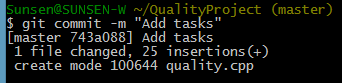




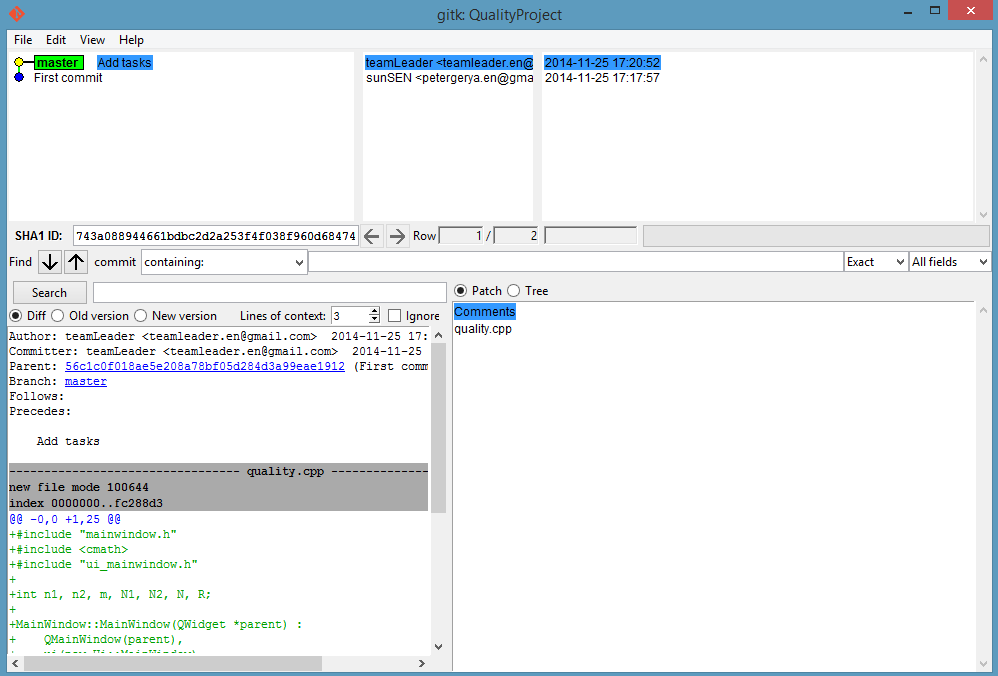
* 1. Додає усі файли до репозиторію



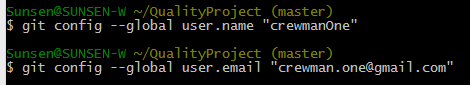
* 1. Виконує коміт



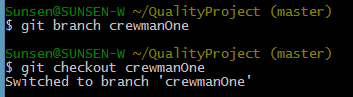
При виконанні команди gitk отримуємо:



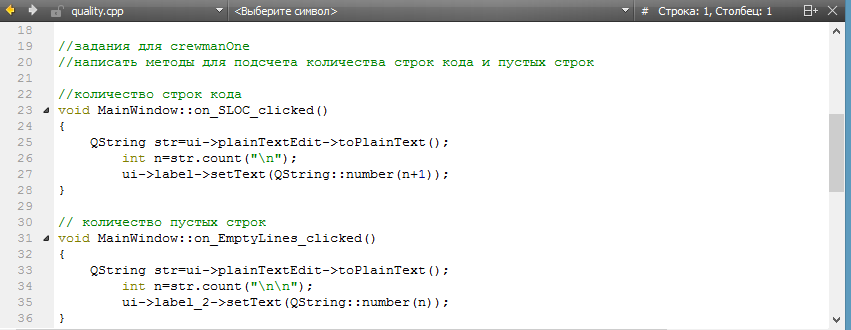
1. Перший член проекту виконує наступні дії:
   1. Вносить свої дані (ім’я та emil)



* 1. Створює свою гілку та переходить у неї



* 1. Виконує своє завдання, яке прописано керівником команди



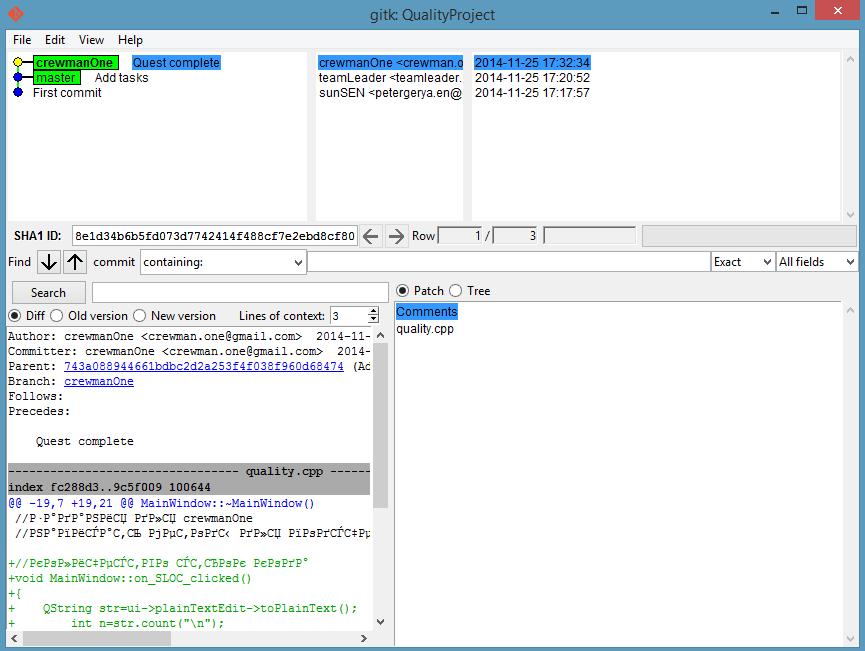
* 1. Додає усі файли до репозиторію



* 1. Робить коміт



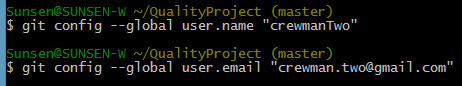
При виконанні команди gitk отримуємо:



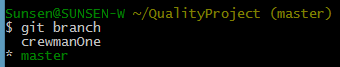
* 1. Повертається у гілку майстра



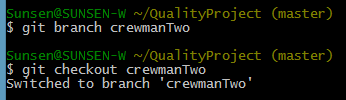
1. Другий член проекту виконує наступні дії:
   1. Вносить свої дані (ім’я та emil)



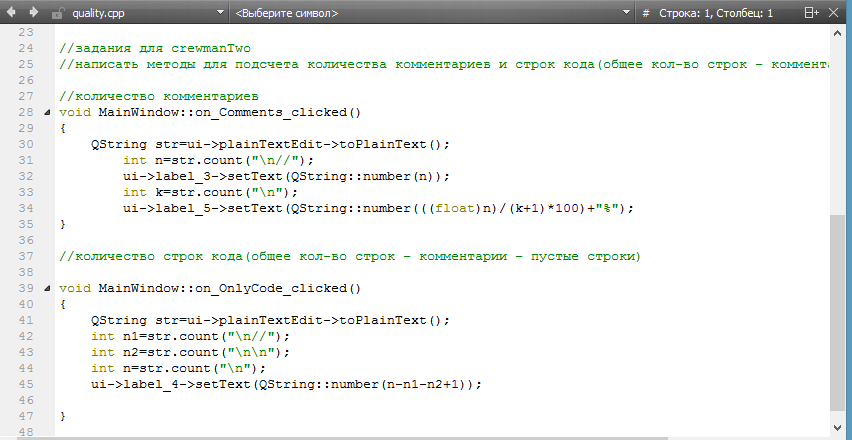
* 1. Дивиться назви усіх існуючих гілок, щоб визначитись з назвою своєї гілки



* 1. Створює свою гілку та переходить у неї



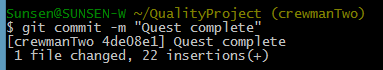
* 1. Виконує своє завдання прописане керівником проекту



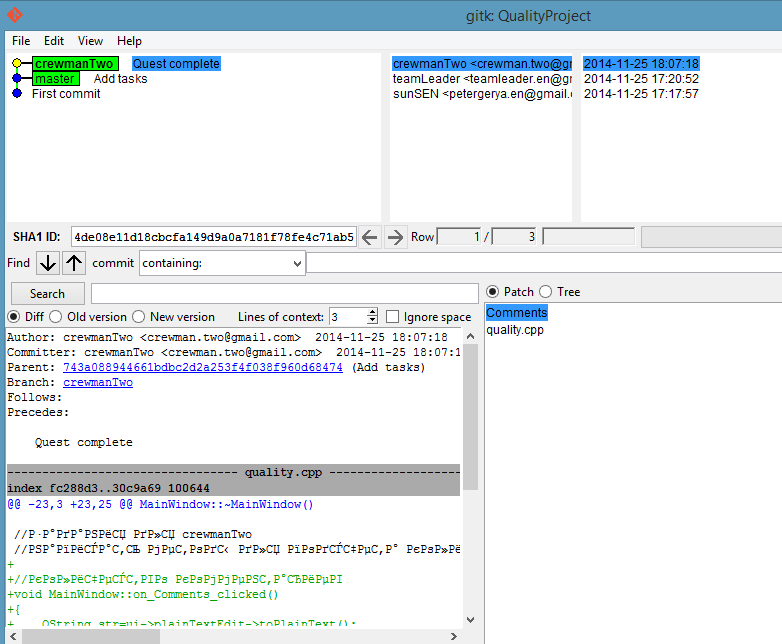
* 1. Додає усі файли до репозиторію



* 1. Виконує коміт



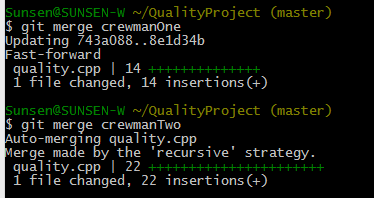
При виконанні команди gitk отримуємо:



* 1. Повертається у гілку майстра

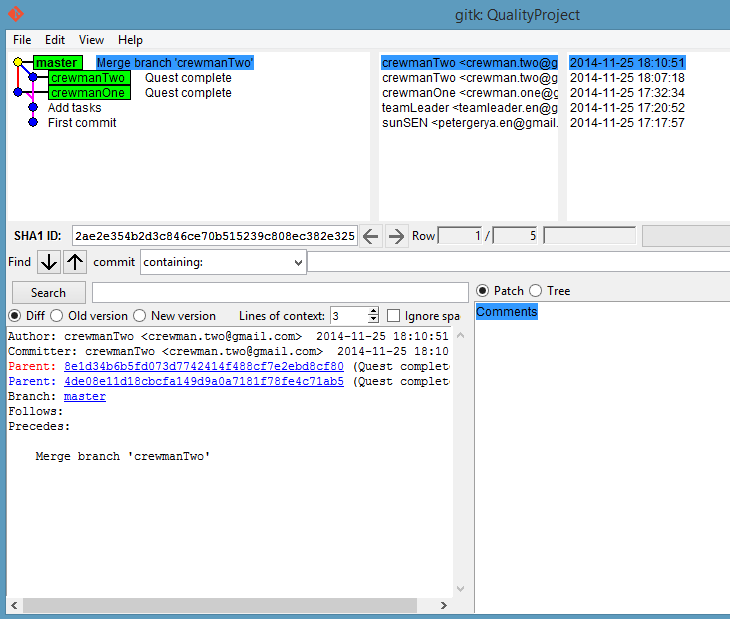


1. Керівник проекту зливає усі гілки разом

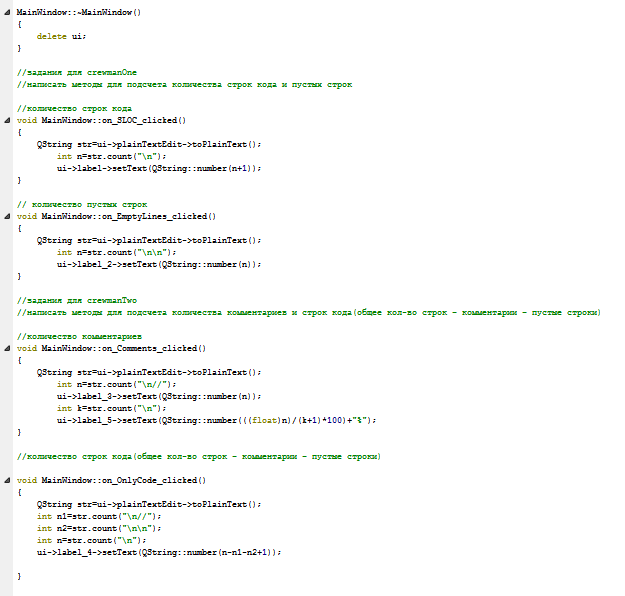


1. Перевіряє результат

При виконанні команди gitk отримуємо:



Файл quality.cpp має вигляд:



**Теорія по Git**

**ОСНОВНІ ХАРАКТЕРИСТИКИ GIT**

Переваги:

1. Надійна система порівняння ревізій та перевірки коректності даних, заснована на алгоритмі хеширувания 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

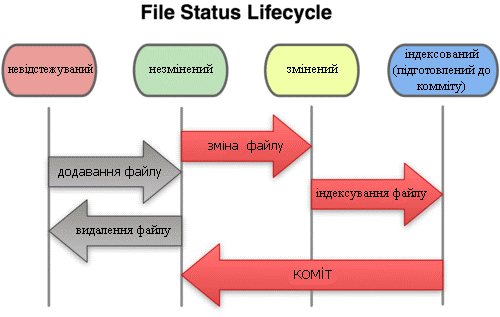
**ТРИ СТАНИ ФАЙЛУ**

• зафиксований

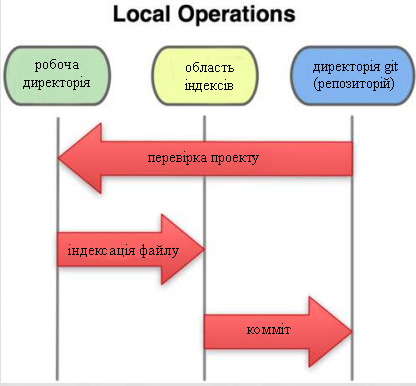
• змінений

• 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>

**КОРИСНІ ПОСИЛАННЯ**

http://m-ivanov.livejournal.com/7988.html - командная работа с Git

https://www.ibm.com/developerworks/ru/library/l-git\_1/ Распределенная система управления версиями Git. Часть 1: Введение

https://www.ibm.com/developerworks/ru/library/l-git\_2/ - Распределенная система управления версиями Git: Часть 2. Структура и философия

http://git-scm.com/book/ru Pro Git book, written by Scott Chacon