Институт среднего профессионального образования

Санкт-Петербургского политехнического университета Петра Великого

**Git - GUI**

2022

Цель работы: Изучить программу Git Gui и научиться ею пользоваться

# Задание к выполнению

**Задание по управлению версиями:**

1. Если система контоля версий Git не установлена, то установить ее (параметры оставить по умолчанию).
2. Запустить Git  GUI или TortoiseGit. Создать новый репозиторий (в папке по фамилии студента).
3. Добавить в папку репозитория файлы. Зафиксировать состояние репозитория (выполнить commit).
4. Внести изменения в файлы.  Зафиксировать новое состояние репозитория.
5. Создать новую ветку 1. Внести в нее изменения (добавить новый файл и изменить существующий файл: добавить, удалить и изменить строки) и зафиксировать их.
6. Переключиться на ветку мастера. Внести в нее изменения (добавить новый файл;  изменить существующие файлы: добавить, удалить и изменить строки первоначального файла) и зафиксировать их.
7. Продемонстрировать слияние веток. Разрешить возникший конфликт.
8. Просмотреть дерево изменений веток (историю).
9. Продемонстрировать откат изменений в ветке 1.
10. Создать удаленный репозиторий (на github.com или в сети).
11. Отправить данные на удаленный репозиторий (выполняется одним из студентов подгруппы).Добавить к удаленному репозиторию участников проекта.
12. Получить данные из удаленного репозитория (выполняется прочими студентами).
13. Изменить полученные данные.
14. Зафиксировать изменения и отправить их на удаленный репозиторий (выполняется всеми студентами подгруппы).
15. Получить данные из удаленного репозитория.
16. Просмотреть историю изменений.

# **Ход работы:**

## Создание хранилища (GIT GUI)

Во-первых необходимо открыть программу:



Рисунок 1 – «Ярлык программы Git Gui»

Дважды нажимаем по ярлыку и у нас открывается программа: В данном окне нам будет необходимо выбрать Repository(Создать, скопировать, открыть).

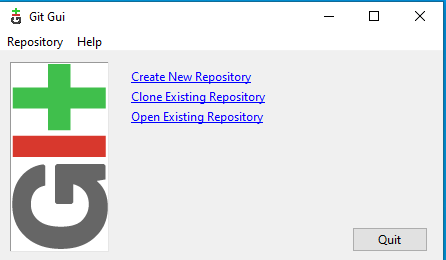


Рисунок 2 – «окно выбора Repository»

Далее выбираем создать новый, откроется следующие окно:

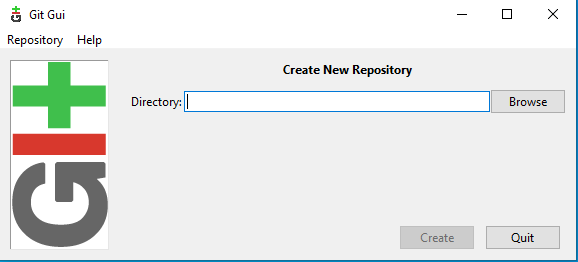


Рисунок 3 – «Окно выбора папки для сохранения»

После выбора папки нажимаем на «Сreate» и переходим на следующие окно:

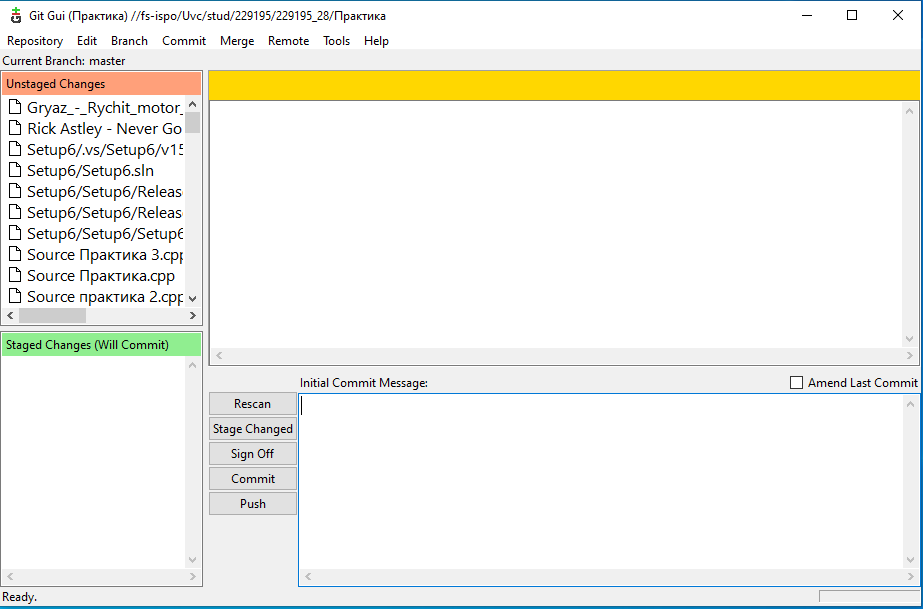


Рисунок 4 – «Основное окно для работы с repository»

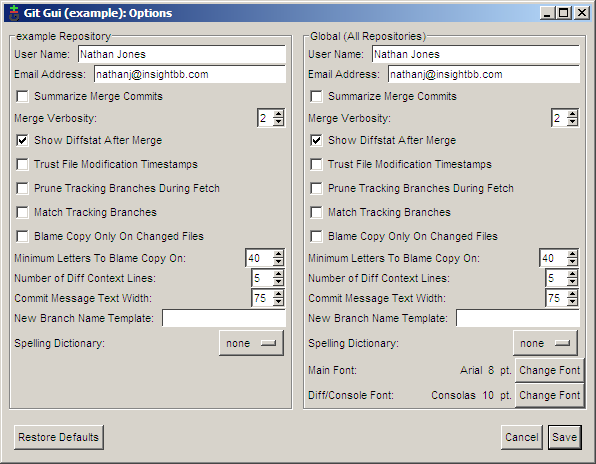


Рисунок 5 – «Настройка repository»

## Фиксация изменений (commiting)

Теперь, когда хранилище было создано, пора создать что-нибудь для фиксации. Для этого примера я создал файл main.c со следующим содержимым (редактирование выполняется в блокноте или другом редакторе):

#include <stdio.h>

int main(int argc, char \*\*argv)

{

    printf("Hello world!\n");

    return 0;

}

Щелчок на кнопку *Rescan (Перечитать)* в git gui заставит его искать новые, измененные и удаленные файлы в директории. На следующем скриншоте git gui нашёл новый файл.

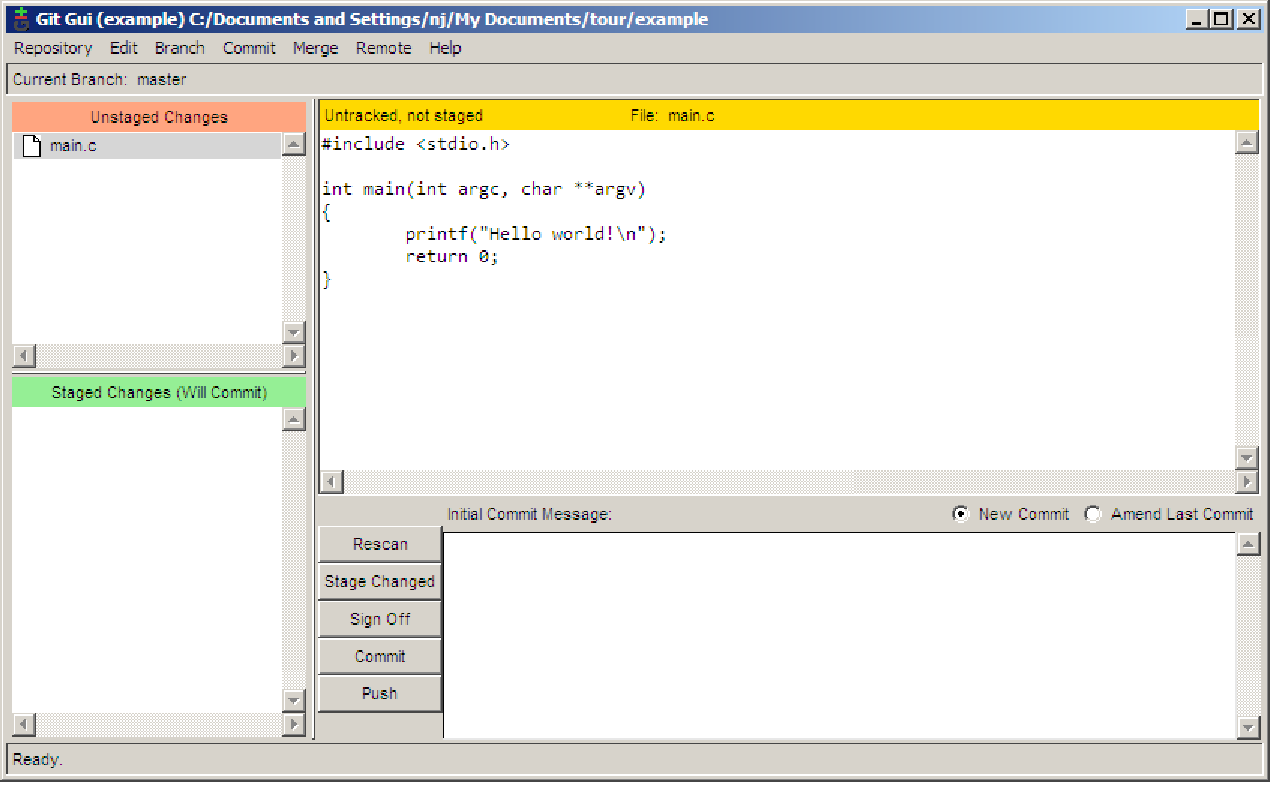


Рисунок 6

Чтобы добавить этот файл в фиксацию, щёлкните на иконке слева от имени файла. Файл будет перемещен с *Unstaged Changes* (Изменено) панели на *Staged Changes* (Подготовлено) панель. Теперь мы можем добавить сообщение фиксации (commit message) и зафиксировать изменения *Commit* (Сохранить) кнопкой.

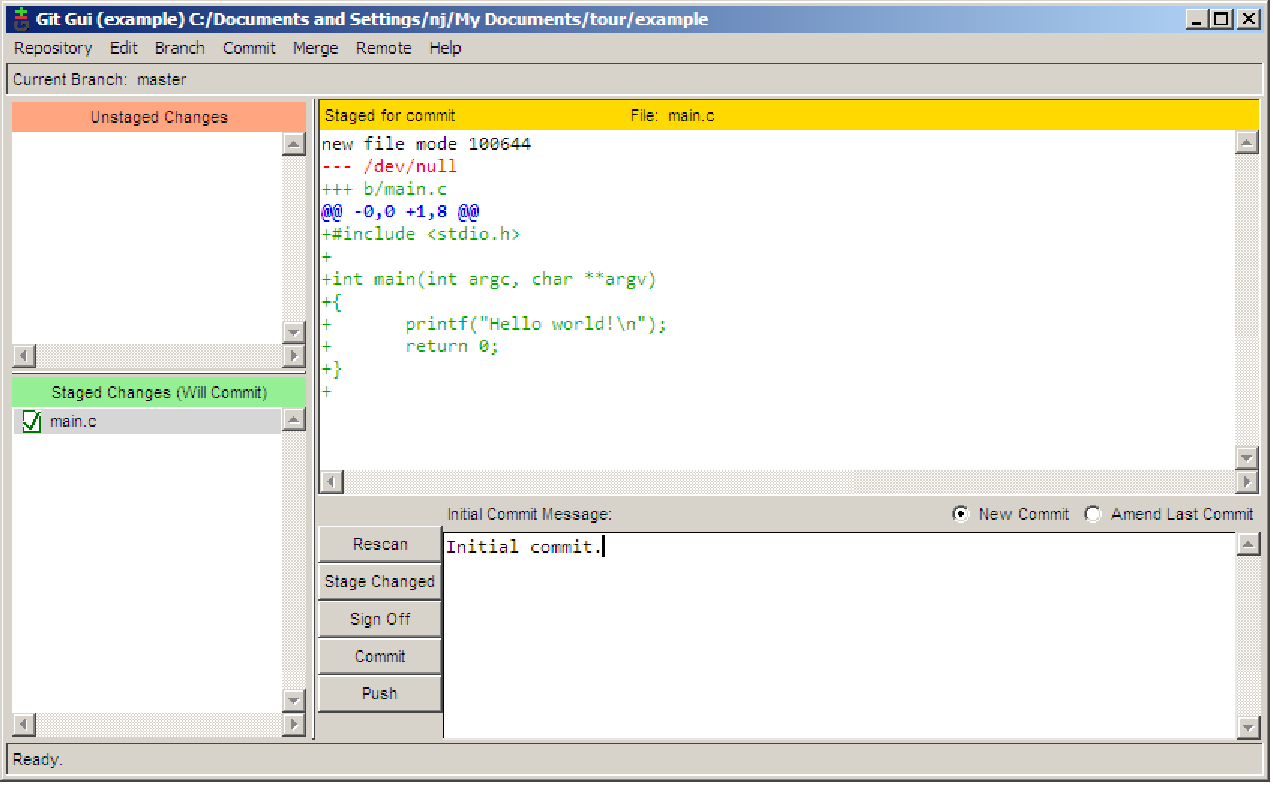


Рисунок 7

Говорить 'hello world' это конечно хорошо, но я хочу чтобы моя программа была более персонализирована. Давайте скажем 'hello' пользователю. Вот как будет выглядеть измененный код (редактирование выполняется в блокноте или другом редакторе):

#include <stdio.h>

#include <string.h>

int main(int argc, char \*\*argv)

{

    char name[255];

    printf("Enter your name: ");

    fgets(name, 255, stdin);

    printf("length = %d\n", strlen(name)); /\* debug line \*/

    name[strlen(name)-1] = '\0'; /\* remove the newline at the end \*/

    printf("Hello %s!\n", name);

    return 0;

}

**Ветвление (branching)**

#include <stdio.h>

int main(int argc, char \*\*argv)

{

    printf("Hello world!\n");

    return 0;

}

 Создадим ветку (branch) для наших новый разработок. Чтобы создать новую ветку в git gui выберете *Branch → Create (Ветвь → Создать)*. Большая возможность какую я хочу добавить это возможность спросить пользователя его фамилию, поэтому я назову ветку lastname. Опции по умолчанию подходят без изменений, так что просто введите имя и щёлкните *Create*.

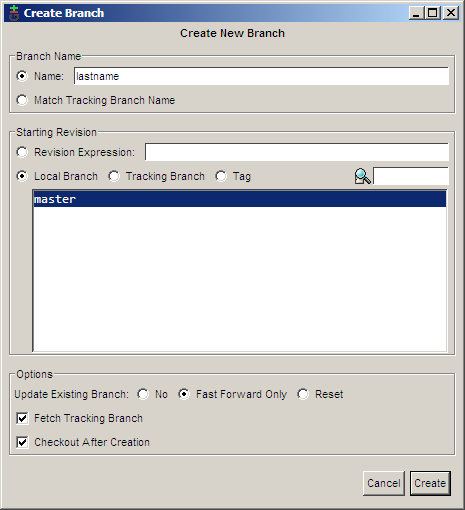


Рисунок 8 – «Окно создание ветки»

Теперь когда я в lastname ветке, я могу делать мои новые модификации:

#include <stdio.h>

#include <string.h>

int main(int argc, char \*\*argv)

{

    char first[255], last[255];

    printf("Enter your first name: ");

    fgets(first, 255, stdin);

    first[strlen(first)-1] = '\0'; /\* remove the newline at the end \*/

    printf("Now enter your last name: ");

    gets(last); /\* buffer overflow? what's that? \*/

    printf("Hello %s %s!\n", first, last);

    return 0;

}

Теперь я могу зафиксировать изменения. Заметьте, что я фиксирую изменения используя другое имя. Мы рассмотрим это позже. Обычно вы всегда будете использовать одно и тоже имя для фиксаций.

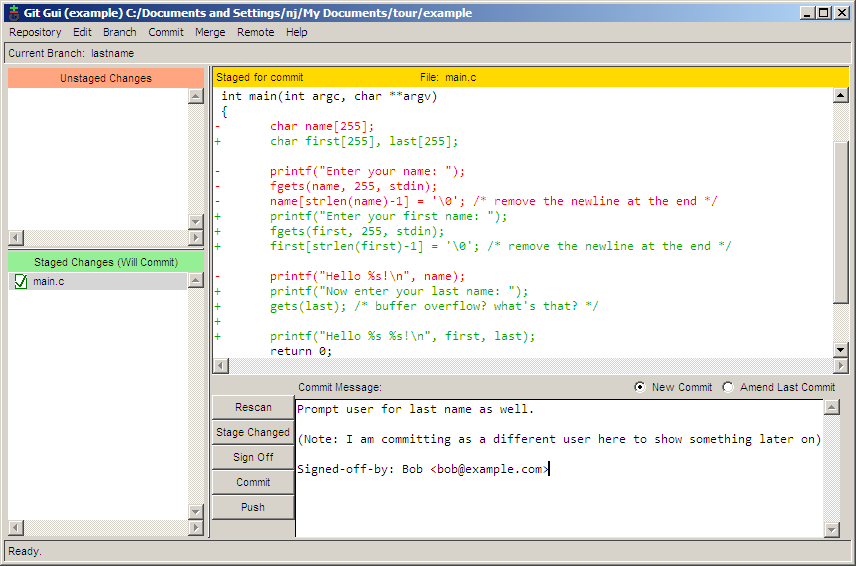


Рисунок 9

Между тем, пользователь проинформировал нас, что не показ запятой после прямого обращения к кому-то - это серьезная ошибка. Чтобы исправить её в нашей стабильной ветке, вы сначала должны переключится назад на неё. Это достигается используя *Branch → Checkout (Ветвь → Перейти)*.

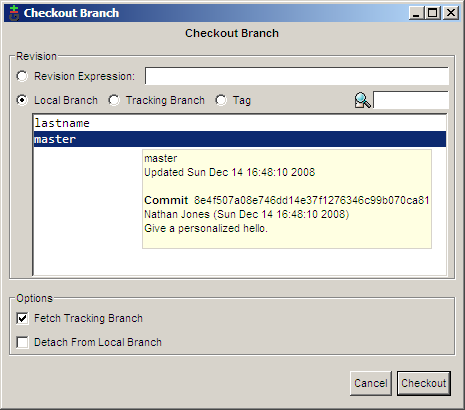


Рисунок 10

Теперь мы можем исправить нашу большую ошибку.

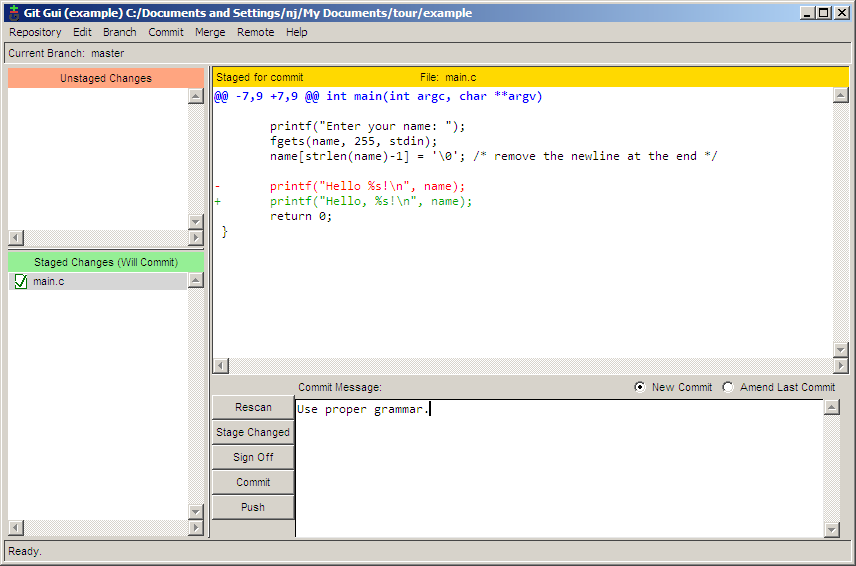


Рисунок 11

Если мы выберем *Repository → Visualize All Branch History (Репозиторий → Показать историю всех ветвей)*, мы увидим как складывается наша история.

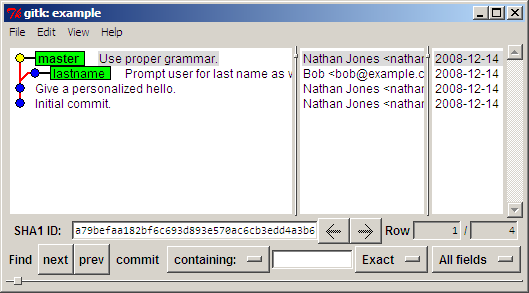
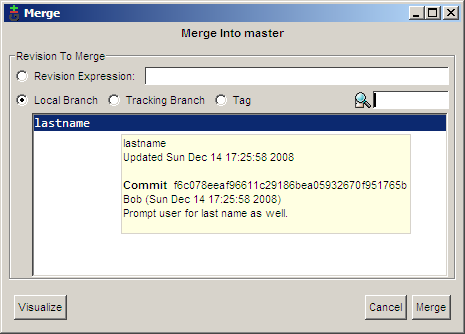


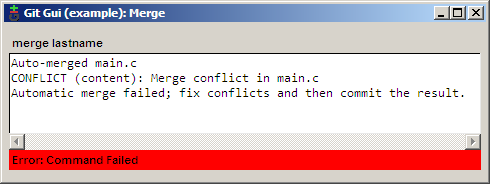
Рисунок 12

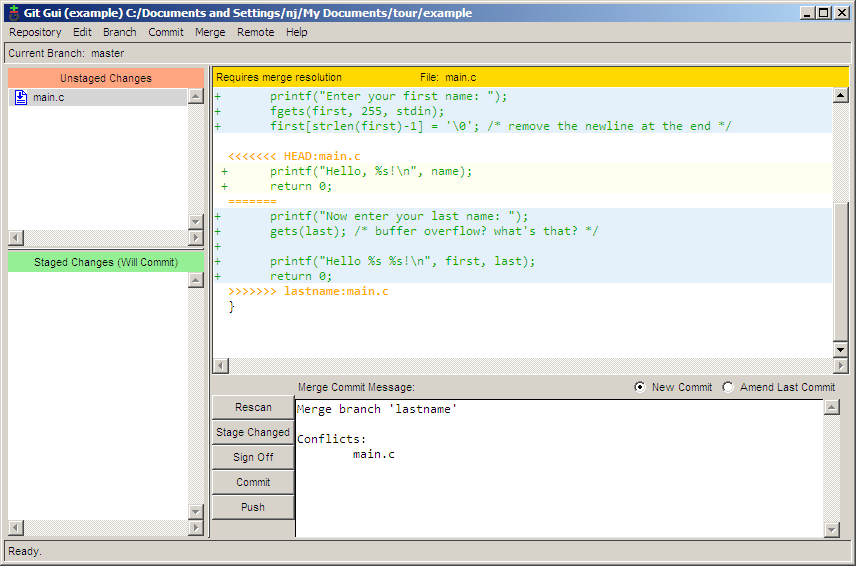
## Слияние (merging)

После напряженной работы мы решили что наша lastname ветка достаточно стабильна, чтобы влить её в master ветку. Чтобы выполнить слияние, используйте *Merge → Local Merge (Слияние → Локальное слияние)*.



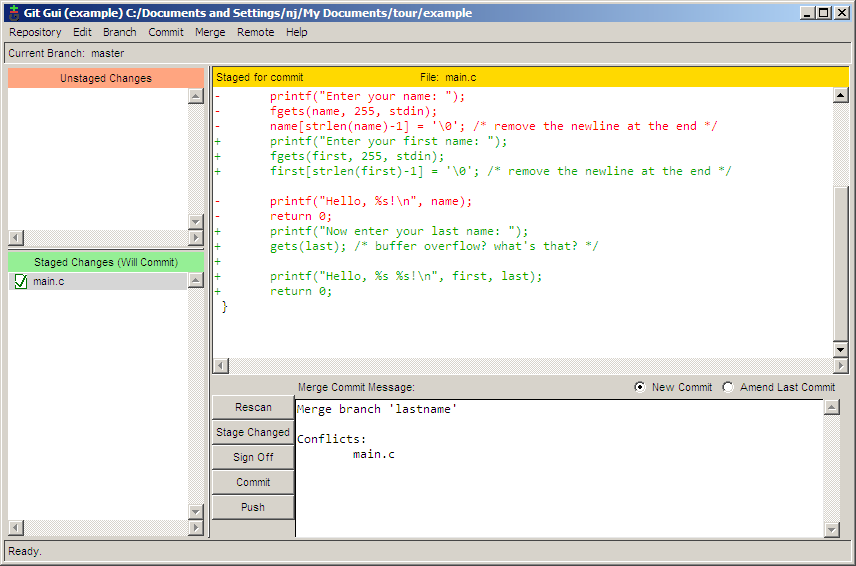
Так как две разных фиксации делали два разных изменения на одной и той же линии, происходит конфликт (conflict).

у



Конфликт может быть разрешен используя любой текстовый редактор (оставляете в тексте только нужный вариант).

После разрешения конфликта, подготовьте изменения, щелкнув на иконке файла, и зафиксируйте слияние щёлкнув по *Commit* кнопке: в редакторе внести изменения в файл и сохранить его - Перечитать — Подготовить - Сохранить.



**Просмотр истории**

Файл main.c становится немного большим, поэтому я решил вынести код, спрашивающий имя пользователя в отдельную функцию. Пока я это делал, я решил вынести функцию в отдельный файл. Хранилище теперь содержит файлы main.c, askname.c, и askname.h.

/\* main.c \*/

#include <stdio.h>

#include "askname.h"

int main(int argc, char \*\*argv)

{

    char first[255], last[255];

    askname(first, last);

    printf("Hello, %s %s!\n", first, last);

    return 0;

}

/\* askname.c \*/

#include <stdio.h>

#include <string.h>

void askname(char \*first, char \*last)

{

    printf("Enter your first name: ");

    fgets(first, 255, stdin);

    first[strlen(first)-1] = '\0'; /\* remove the newline at the end \*/

    printf("Now enter your last name: ");

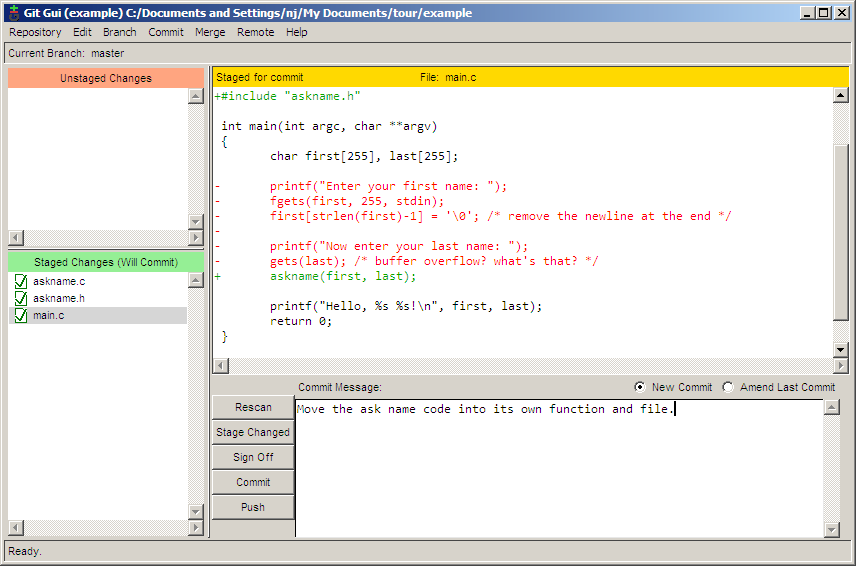
    gets(last); /\* buffer overflow? what's that? \*/

}

/\* askname.h \*/

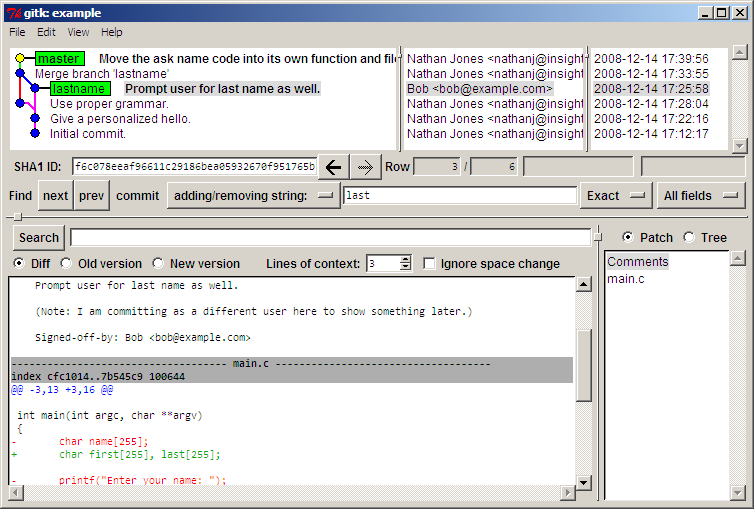
void askname(char \*first, char \*last);

Файлы создают в редакторе. Затем перечитать репозиторий, подготовить все и сохранить.



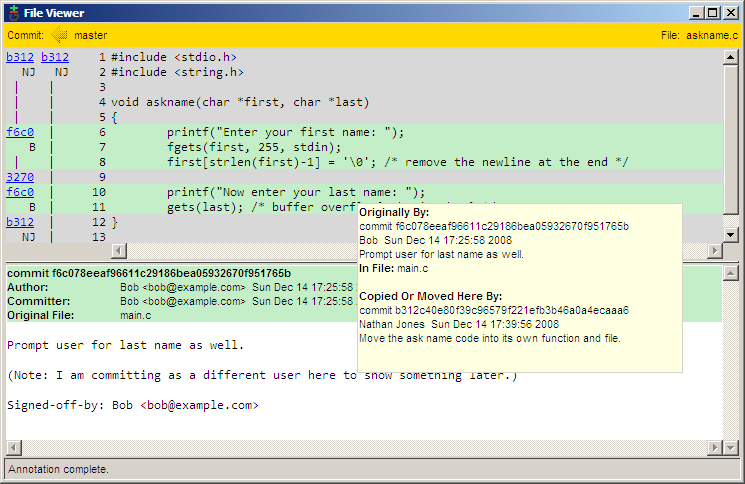
История хранилища может быть просмотрена и изучена выбрав *Repository → Visualize All Branch History*. На следующем скриншоте я пытаюсь найти в какой фиксации была добавлена last переменная, ища все фиксации в которых было добавлено или убрано слово *last*.

Фиксации, которые подходят под условия поиска отмечены жирным шрифтом, что бы быстро и легко обнаружить нужную фиксацию. Можно посмотреть старую и новую версии. Цветом выделены изменения.



Через пару дней, кто-то просматривая наш код увидел что gets функция может вызвать переполнение буфера. Будучи любителем показывать пальцем, этот человек решает запустить git blame что бы увидеть кто последний раз редактировал эту линию кода. Проблема в том что Боб, тот кто зафиксировал эту линию в хранилище, а я последний кто трогал её когда я переместил строку в другой файл. Очевидно, я не виноват (конечно же). Но так ли умен git что бы обнаружить это? Да, это так.

Что бы запустить blame, выберете *Repository → Browse master's Files (Репозиторий → Показать файлы ветви master)*. Из дерева, которое появится, дважды щёлкните на файле с интересующей строкой, который в данном случае askname.c. Наведённая мышка на интересующую линию показывает нам подсказку, которая говорит нам всё что нам надо знать.



Здесь мы можем видеть что эта линия была зафиксирована Бобом в фиксации f6c0, а затем я её переместил в её новое месторасположение в фиксации b312.

## Отмена изменений

Для отмены внесенных изменений до состояния последней фиксации:

Меню Состояния – Отменить изменения.

Для отката к конкретной фиксации (**reset**):

Меню Репозиторий – История ветки – (выбрать нужное состояние и в контекстном меню выбрать Установить для ветви это состояние). Возможны варианты (мягкий — изменение только индекса или жесткий — изменения в индексе и на диске).

Для отката через новую фиксацию — создается новая фиксация, содержащая изменения, обратные зафиксированным (**revert**):

Меню Репозиторий – История ветки – (выбрать нужное состояние и в контекстном меню выбрать revert this commit).

## Работа в репозитории git hub.

## 

На этой картинке я добавил друга в свой репозиторий