Инструментальные средства управления версиями

Учебно-методические материалы «Инструментальные средства управления версиями» представляют собой методические указания к лабораторной работе по дисциплинам «Технологии разработки программного обеспечения» (по направлению магистерской подготовки).

Цель работы:

Изучить базовые возможности систем управления версиями с применением CASE средств.

В материалах рассмотрены принципы управления версиями. Приведены инструкции по работе с версиями программ в среде управления версиями Git. Рассмотрены примеры использования указанных средств. В заключительной части методических указаний приведены контрольные вопросы, список рекомендуемой литературы и пример задания.

Ознакомившись с методическими указаниями и разобрав приведенные в нем примеры, студент может получить у преподавателя свой вариант задания и приступить к его выполнению.

Оглавление

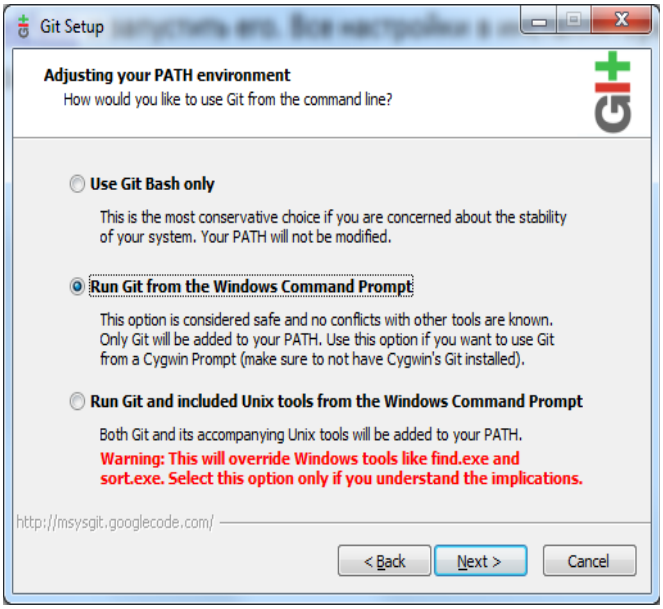
Теоретическая часть

Руководство по системе управления версиями - Git

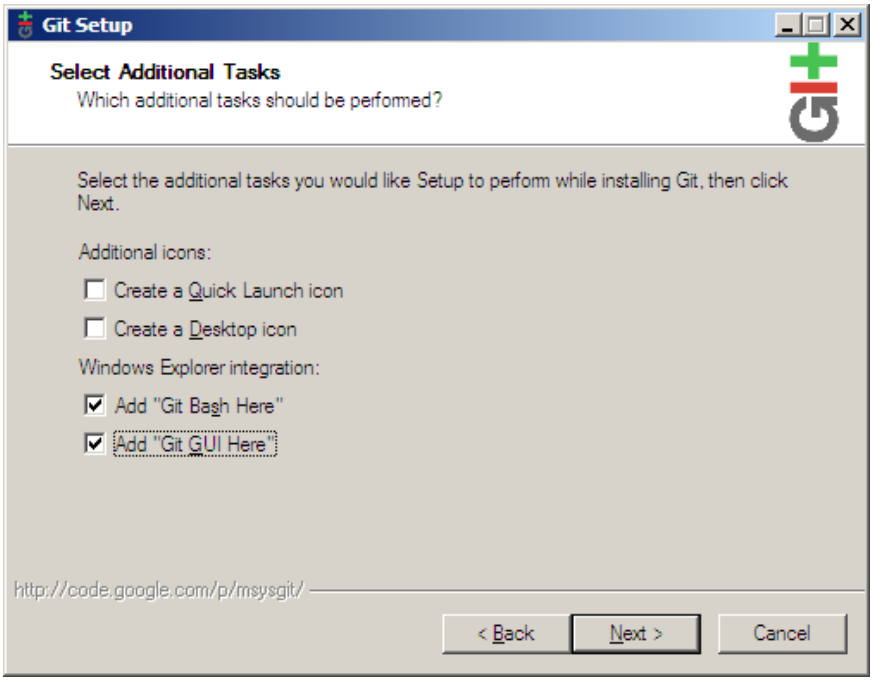
Если необходимо устанивить GIT:

Для начала, скачайте msysgit (инсталляционный пакет http://git-scm.com/download/win) и запустить его. Все настройки в инсталляторе оставляем по умолчанию, кроме представленной ниже.

Не выставляйте её в самое нижнее положение!



При прохождении шагов установочной программы, вы можете захотеть отметить опцию интеграции с Windows Explorer, когда вы совершаете правый щелчок мыши на папке.



Продолжите нажатием *Next* пока установка не завершится.

Последовательность работ с локальным репозиторием

Git обладает необычайной легкостью в использовании не только как распределенная система контроля версий, но и в работе с локальными проектами. Давайте разберем обычный цикл — начиная с создания репозитария — работы разработчика git над собственным персональным проектом:

1. создаем рабочую директорию проекта.
2. создаем репозитарий в директории.
3. индексируем все существующие файлы проекта (добавляем в репозиторий) и создаем инициализирующий коммит.
4. Создаем новую ветку,
5. переключение в новую ветку (можно сделать в один шаг).
6. Далее, после непосредственной работы с кодом,
7. индексируем внесенные изменения и совершаем коммит.
8. Переключаемся в основную ветку,
9. смотрим отличия между последним коммитом активной ветки и последним коммитом экспериментальной.
10. Проводим слияние,
11. если был конфликт, то разрешаем его и повторяем слияние.
12. Ну и на всякий случай оценим проведенную за последний день работу.

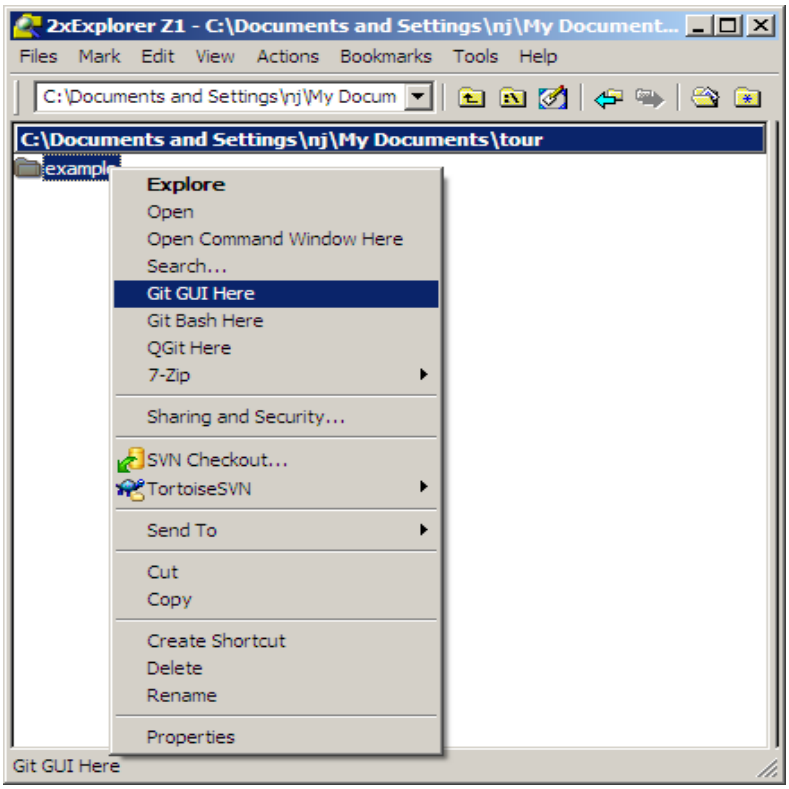
Почему именно так? Зачем отказываться от линейной модели? Хотя бы даже потому, что у программиста появляется дополнительная гибкость: он может переключаться между задачами (ветками); под рукой всегда остается «чистовик» — ветка master; коммиты становятся мельче и точнее.

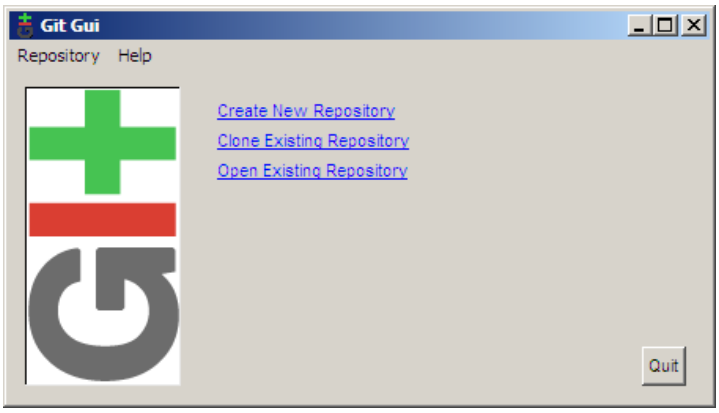
Создание хранилища (GIT GUI)

Работать можно в командной строке (инструкции см. в соответствующем файле), или в интерфейсе Git Gui, или в интерфейсе TortoiseGit (на усмотрение студента).

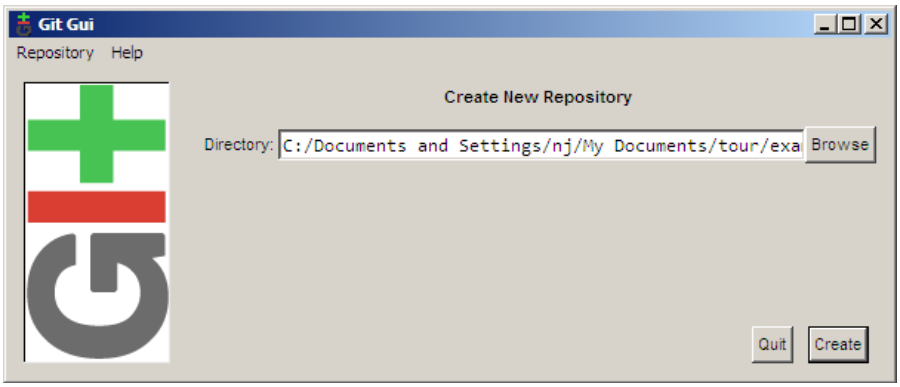
Далее подробно рассматривается работа с Git Gui.

Что бы создать хранилище, сначала создайте папку в которой ваш проект будет жить. Далее, правый щелчок мышки на этой папке и выберете *Git GUI*. Так-как в папке пока что не содержится git хранилища вам будет показан диалог создания.

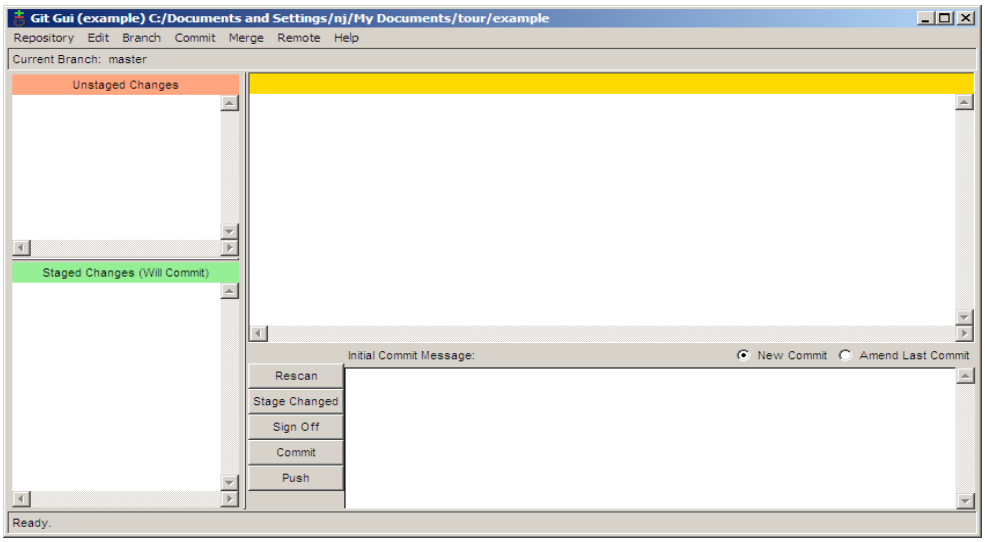




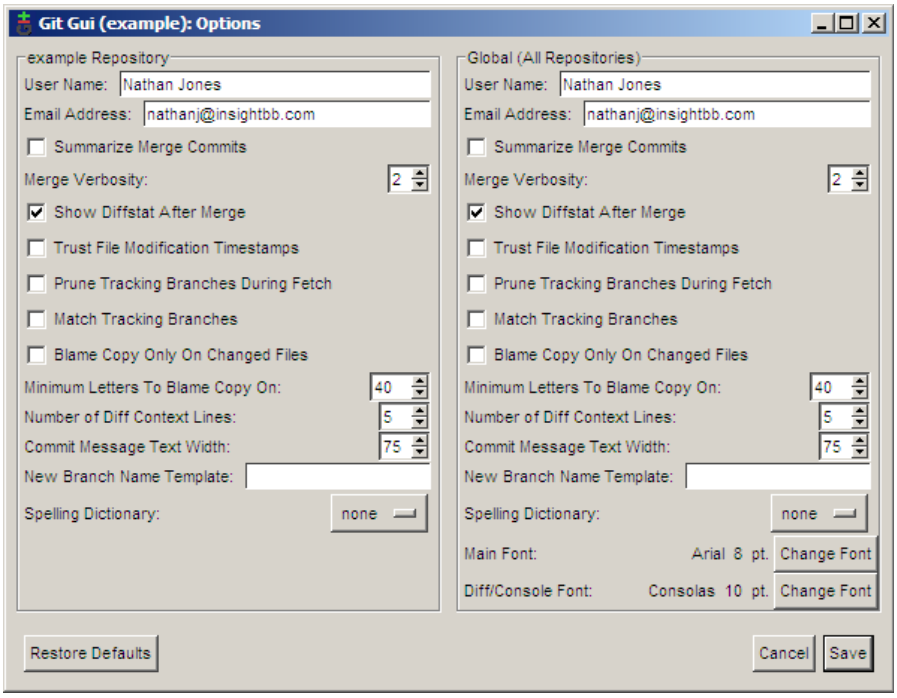
Выбор *Create New Repository* приводит нас к следующему диалогу.



Заполните путь к вашей новой директории и щёлкните *Create*. Далее вы увидите главный интерфейс git gui, который в дальнейшем будет показываться когда вы будете делать правый щелчок на вашей папке и выбирать *Git GUI*.



Теперь когда хранилище создано, вам надо сообщить git-у кто вы такой, что бы сообщении фиксации (commit message) был отмечен правильный автор. Что бы сделать это, выберете *Edit → Options (Редактировать → Настойки)*.



В диалоге опций расположены 2 варианта на выбор. С левой стороны диалога опции которые влияют только на это хранилище, в то время как правая сторона содержит глобальные опции применяемые ко всем хранилищам. Значения по умолчанию приемлемы, так что заполните только имя пользователя и email, пока что. Если у вас есть любимый шрифт, вы можете выставить его сейчас, так же.

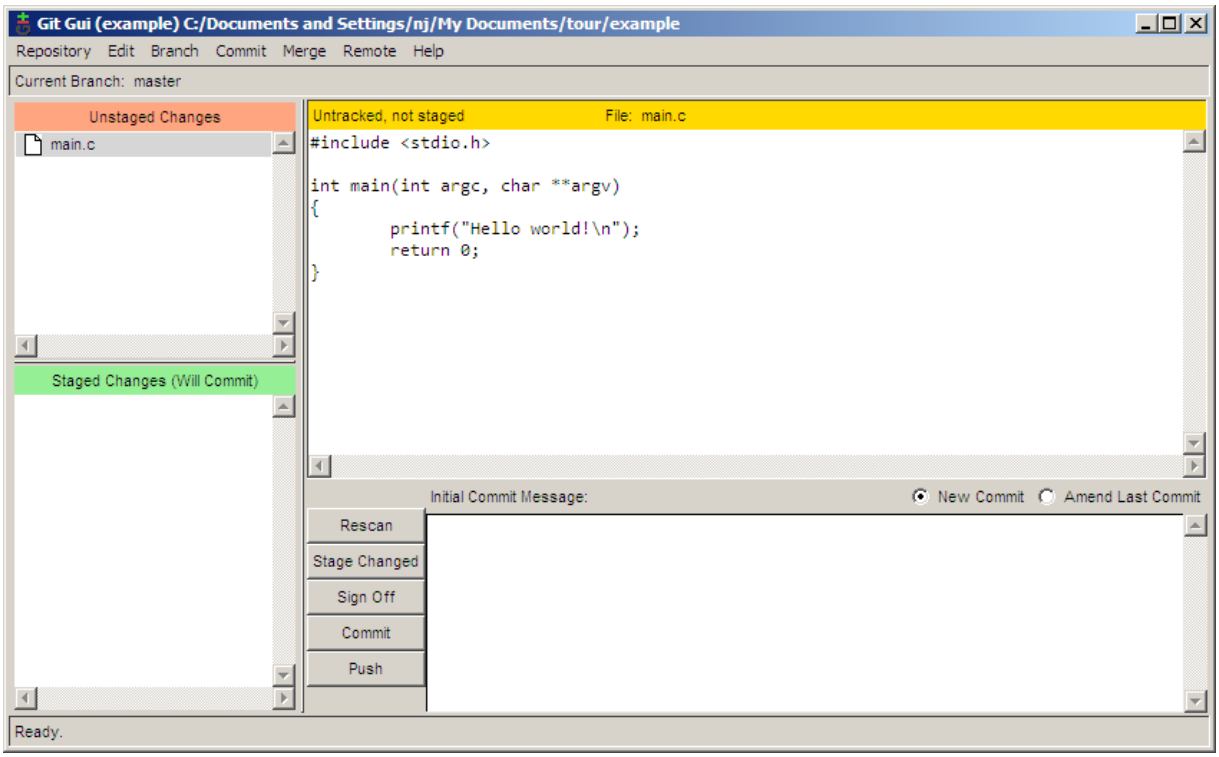
Фиксация изменений (commiting)

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):

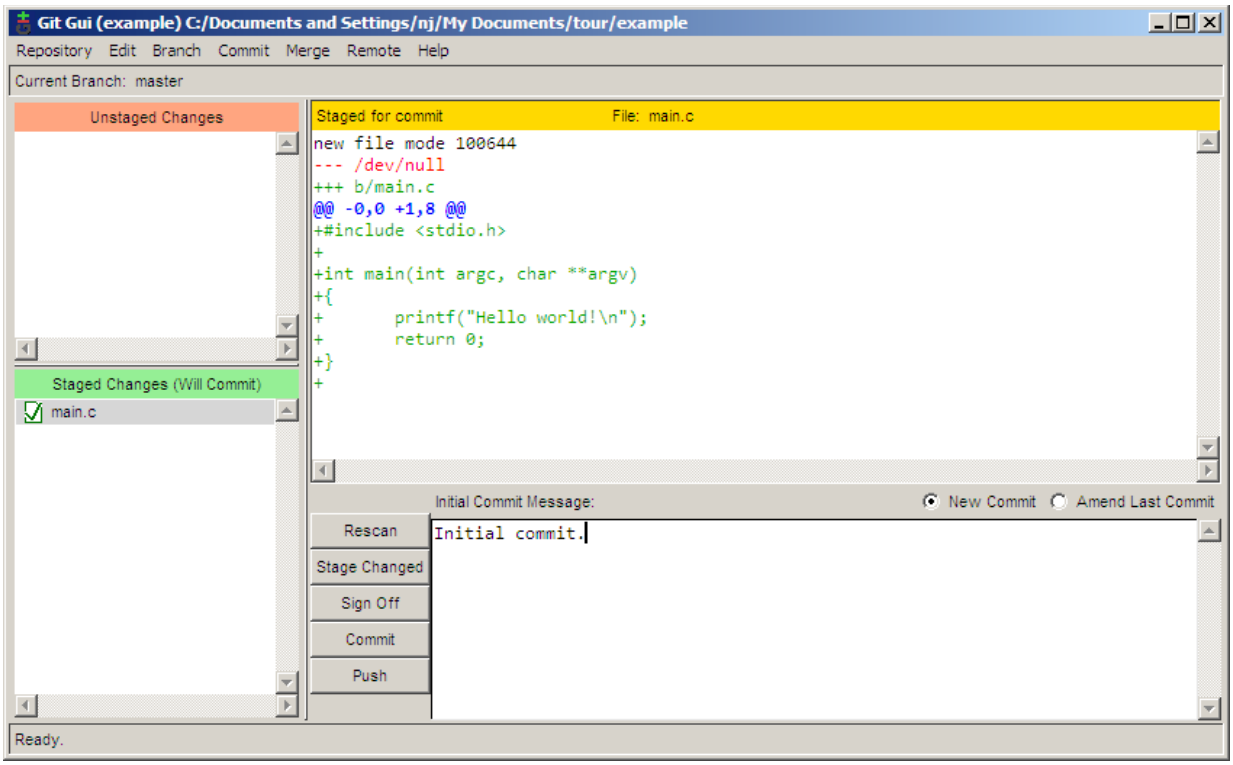
```
#include <stdio.h>
```

```
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 удалённые файлы в директории. На следующем скриншоте git gui нашёл новый файл.



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



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

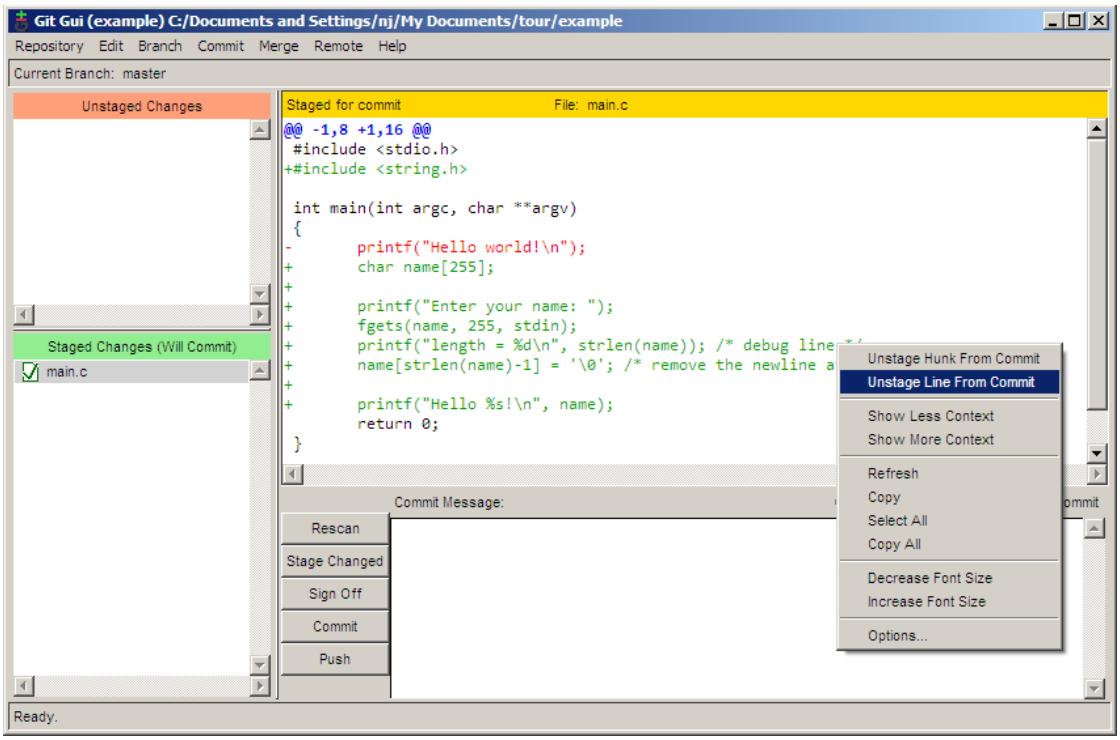
```
#include <stdio.h>
#include <string.h>

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;
}
```

У меня были проблемы с тем что новая линия печаталась после имени пользователя, поэтому я добавил отладочную линию кода, что бы помочь себе найти причину. Я бы хотел зафиксировать изменение без этой отладочной линии, но я хочу сохранить эту линию в моей рабочей копии что бы продолжить отладку. С git gui это не проблема. Сначала щёлкните *Rescan* (перечитать) для поиска изменений.

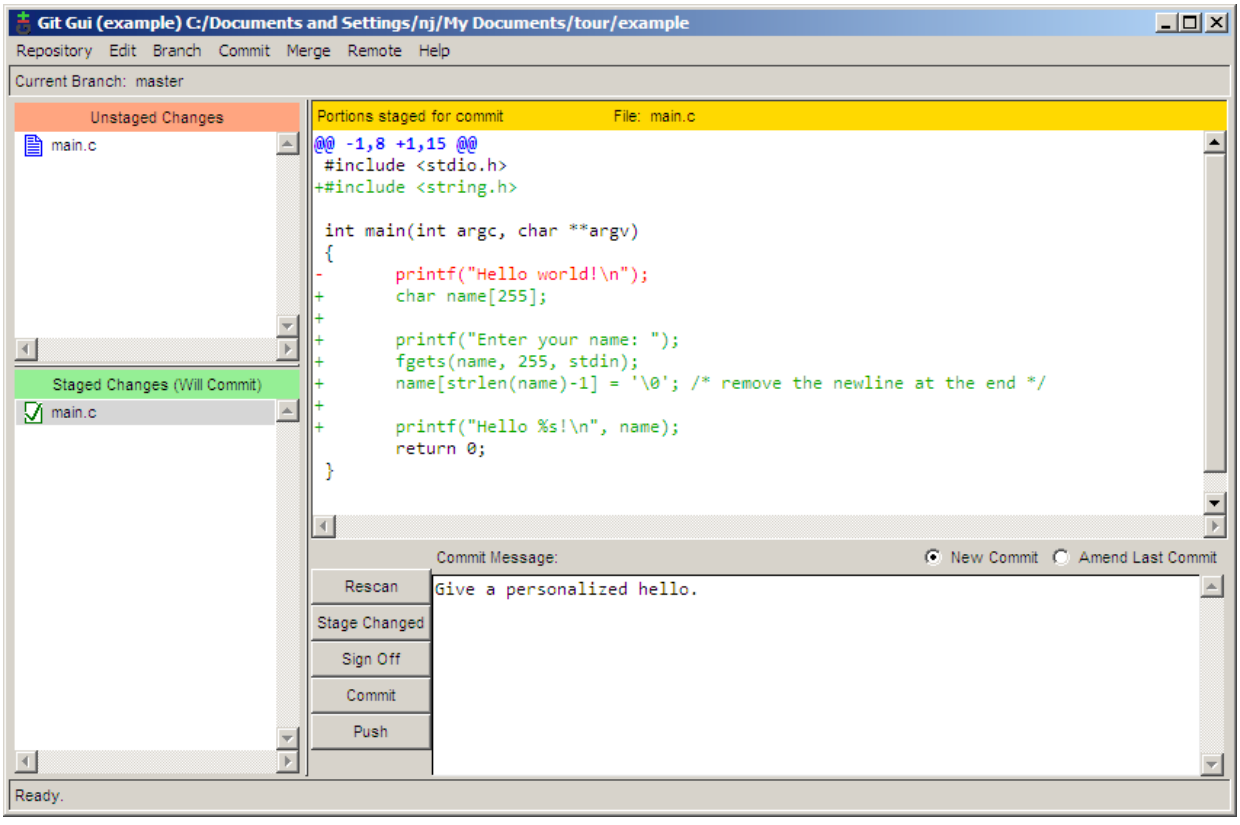
Изменения выделены красным (удаленные линии) или зеленым (добавленные линии).

Далее щёлкните на иконке слева от файла что бы подготовить (stage) изменения к фиксации. Файл будет перенесен в нижнее окно.

Затем правый щелчок на отладочной линии и выберите *Unstage Line From Commit (Убрать строку из подготовленного)*.



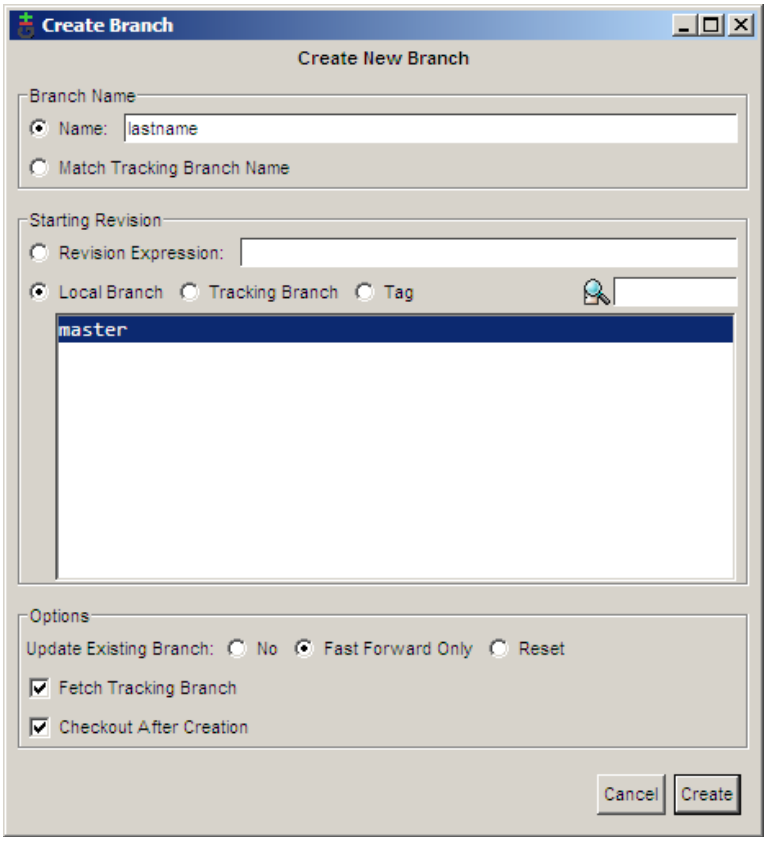
Теперь отладочная линия не была подготовлена (unstaged) к фиксации, в то время как остальные изменения были. Осталось только заполнить сообщение фиксации и зафиксировать изменения щёлкнув по *Commit (Сохранить)*.



Ветвление (branching)

Теперь, давайте предположим что мы хотим начать добавлять новые возможности в нашу следующую большую версию программы. Но мы так же хотим сохранить стабильную версию в которой исправлять ошибки. Что бы сделать это мы создадим ветку (branch) для наших новый

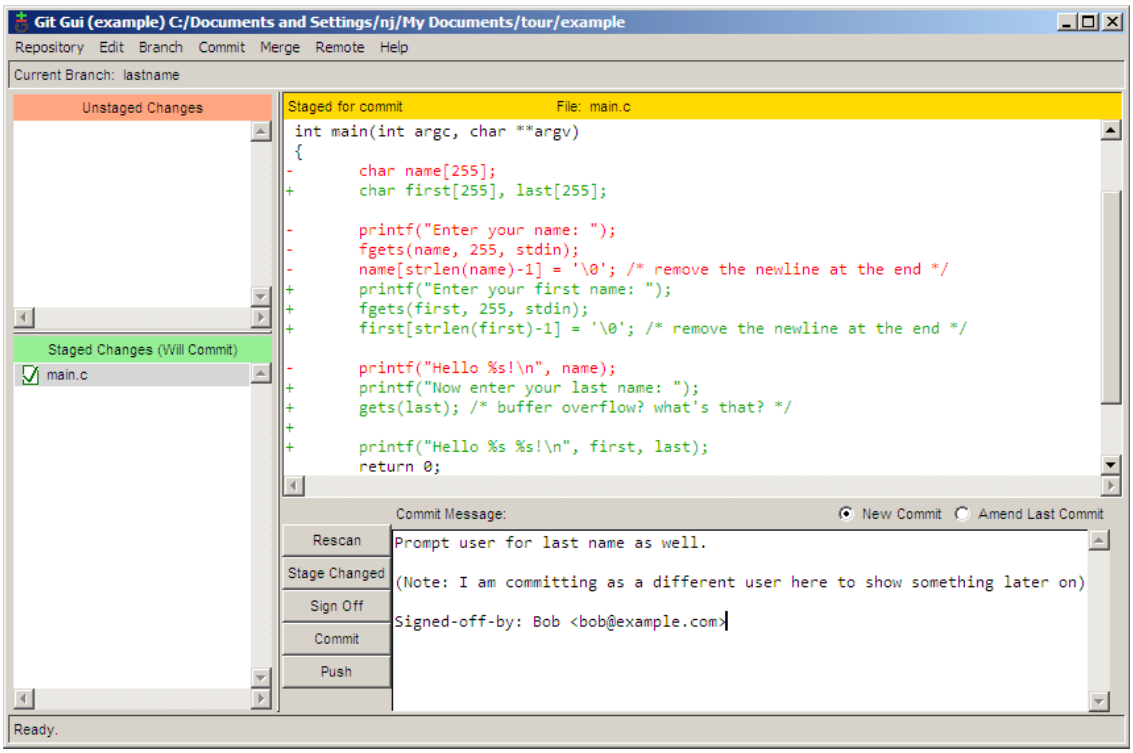
разработок. Что бы создать новую ветку в git gui выберете *Branch → Create (Ветвь → Создать)*. Большая возможность какую я хочу добавить это возможность спросить пользователя его фамилию, поэтому я назову ветку lastname. Опции по умолчанию подходят без изменений, так что просто введите имя и щёлкните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



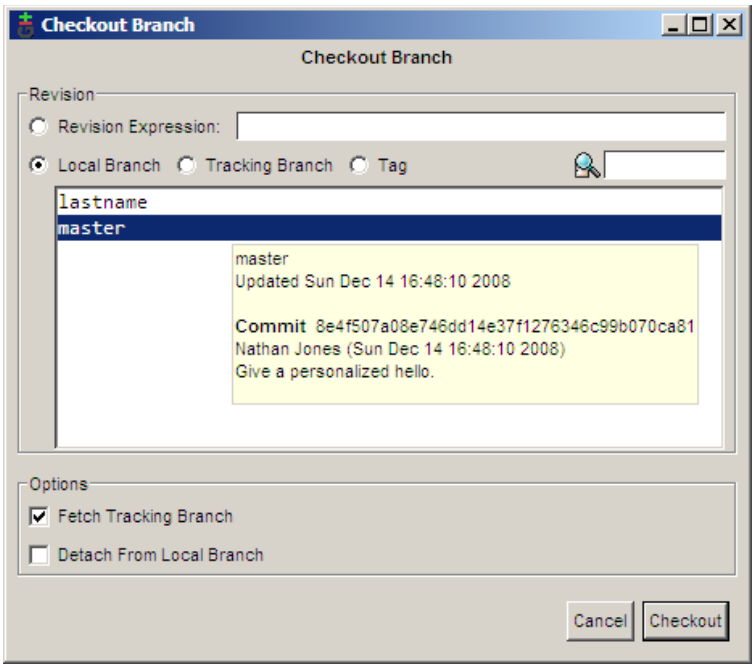
Теперь когда я в lastname ветке, я могу делать мои новые модификации:

```
#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;
}
```

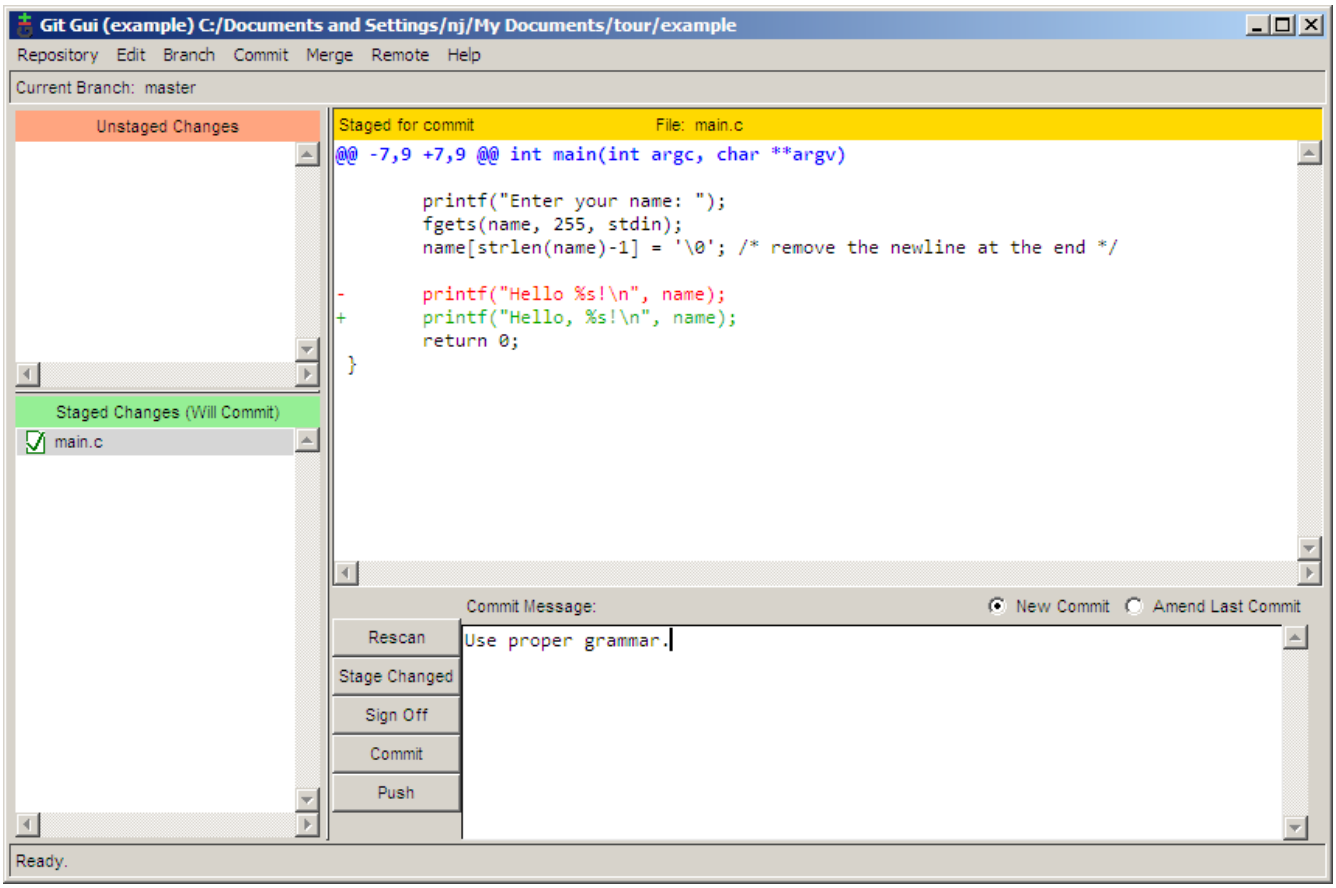
Теперь я могу зафиксировать изменения. Замете что я фиксирую изменения используя другое имя. Мы рассмотрим это позже. Обычно вы всегда будете использовать одно и тоже имя для фиксаций.



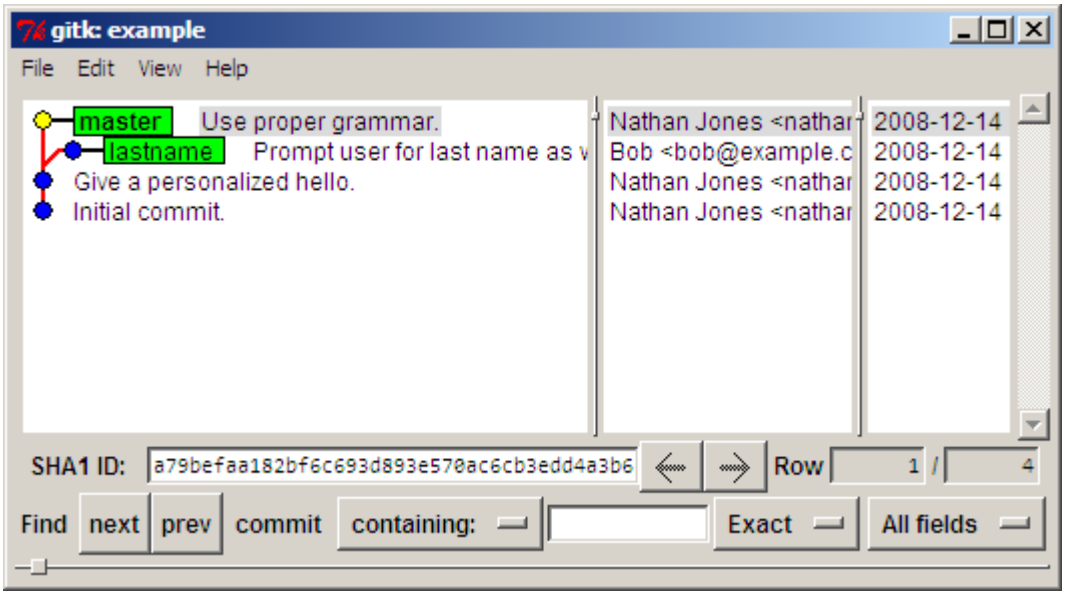
Между тем, пользователь проинформировал нас что не показ запятой после прямого обращения к кому-то это серьёзная ошибка. Что бы исправить её в нашей стабильной ветке, вы сначала должны переключится назад на неё. Это достигается используя *Branch → Checkout (Ветвь → Перейти)*.



Теперь мы можем исправить нашу большую ошибку.

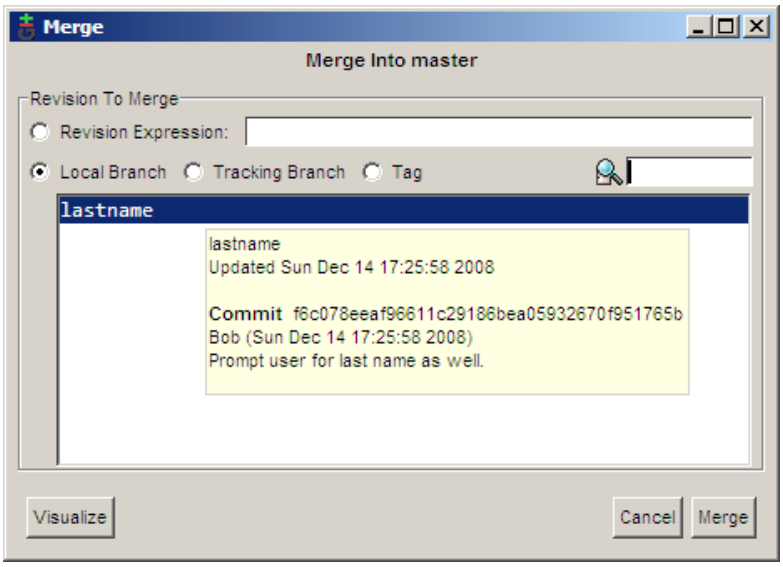


Если мы выберем *Repository → Visualize All Branch History (Репозиторий → Показать историю всех ветвей)*, мы увидим как складывается наша история.

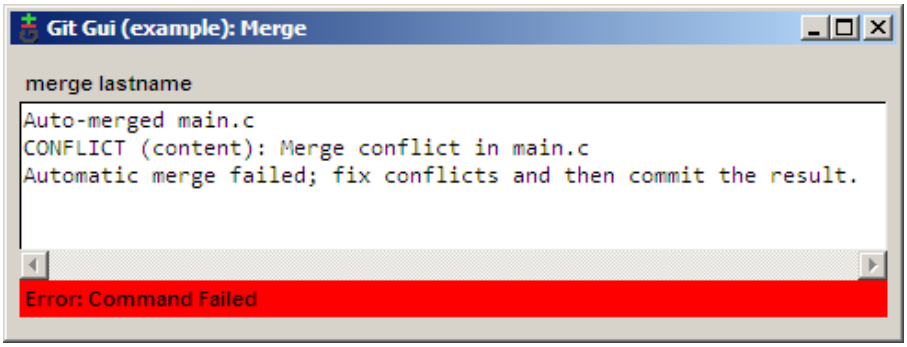


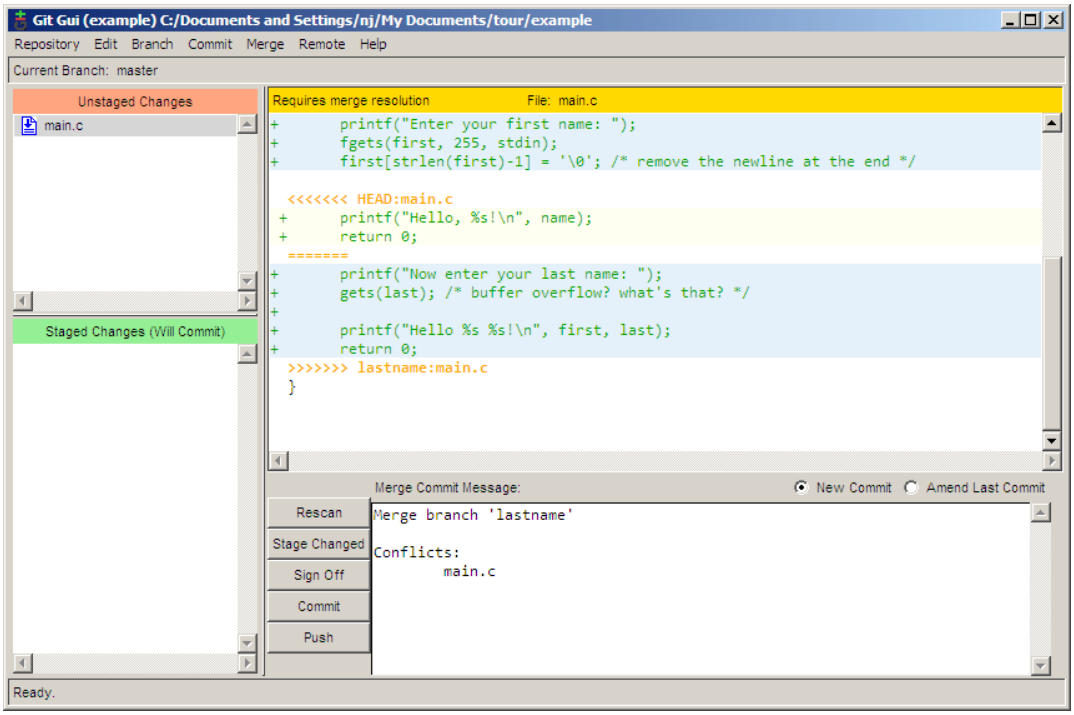
Слияние (merging)

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



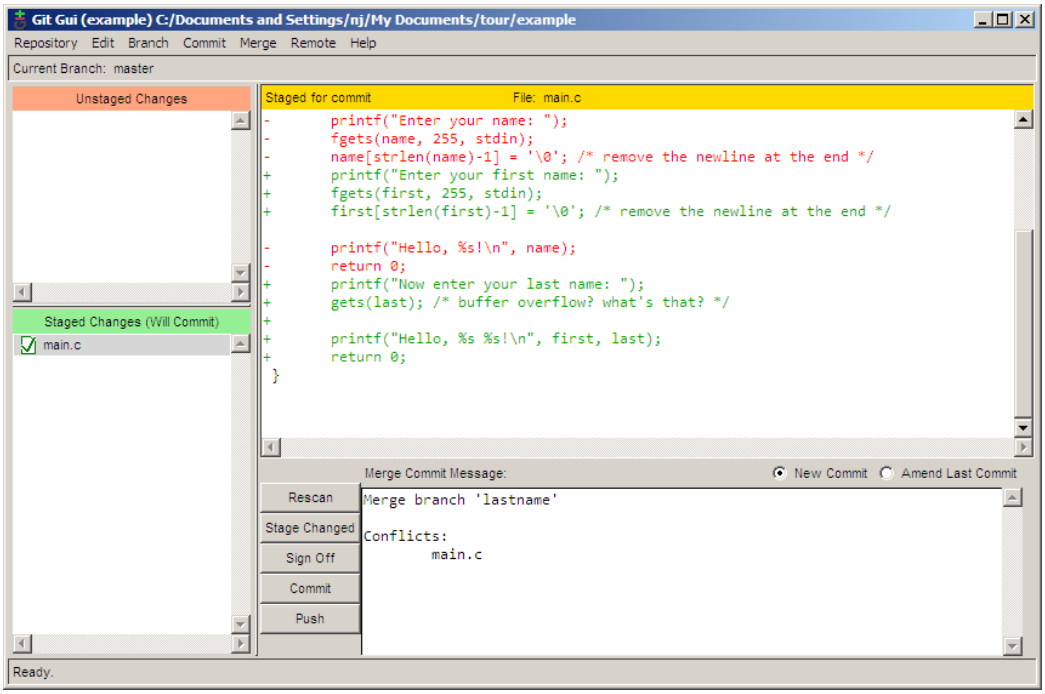
Так как две разных фиксации делали два разных изменения на одной и той же линии, происходит конфликт (conflict).





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

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



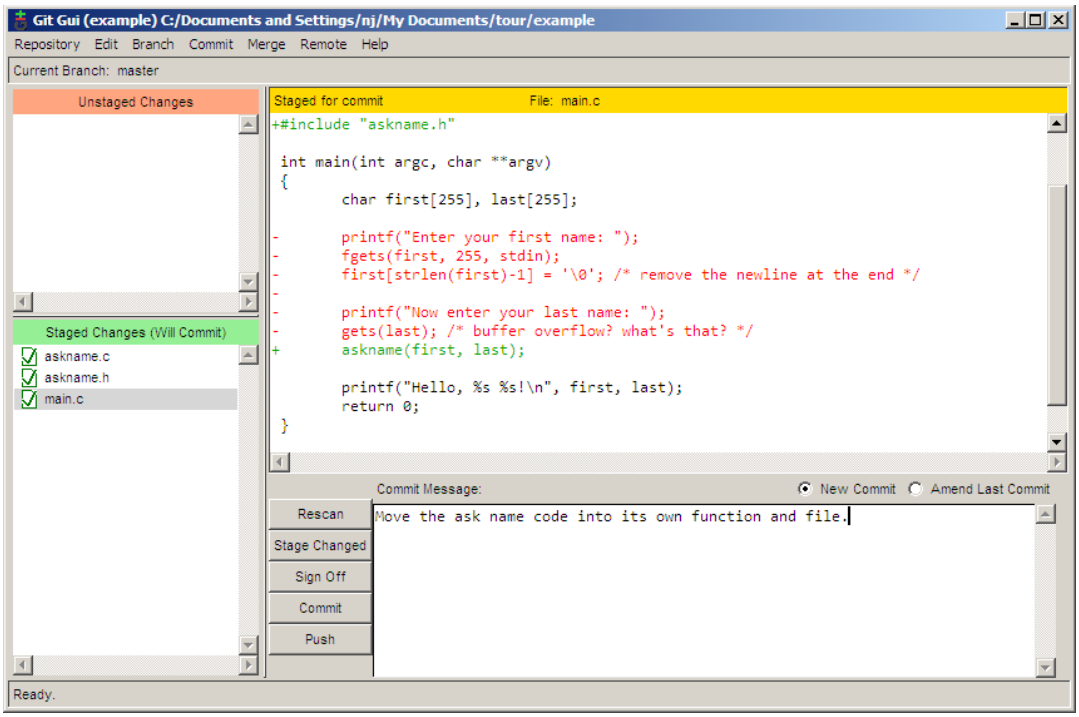
Просмотр истории

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? */
}
```

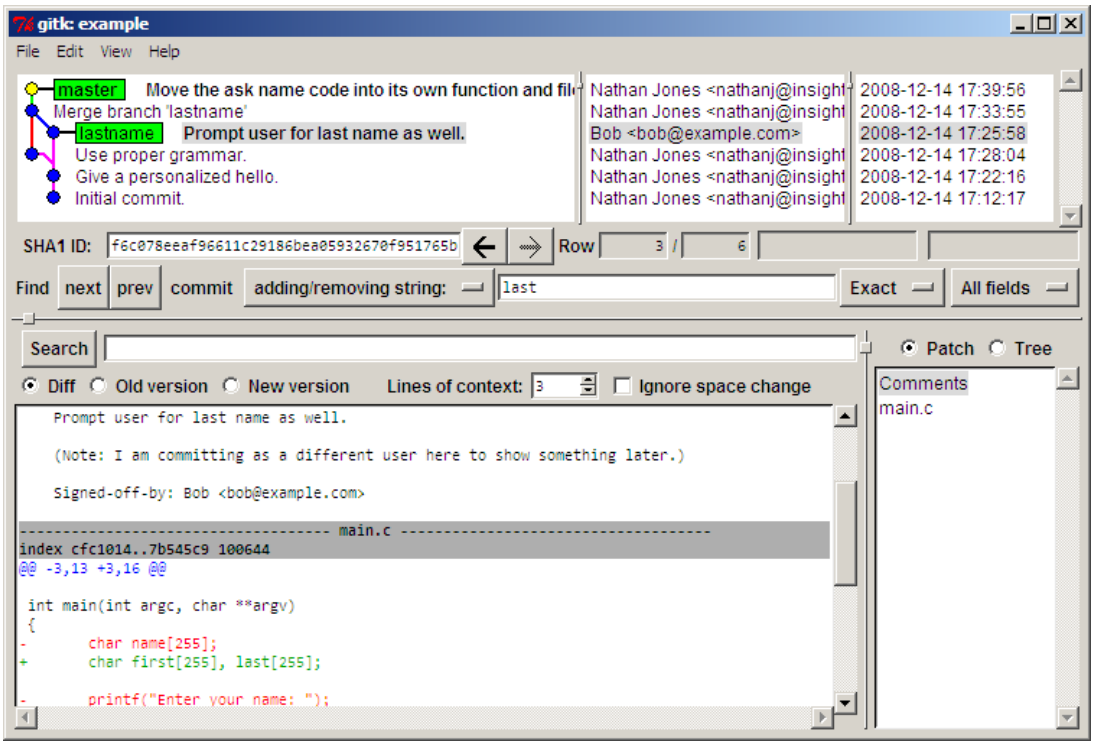
```
/* askname.h */
void askname(char *first, char *last);
```

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



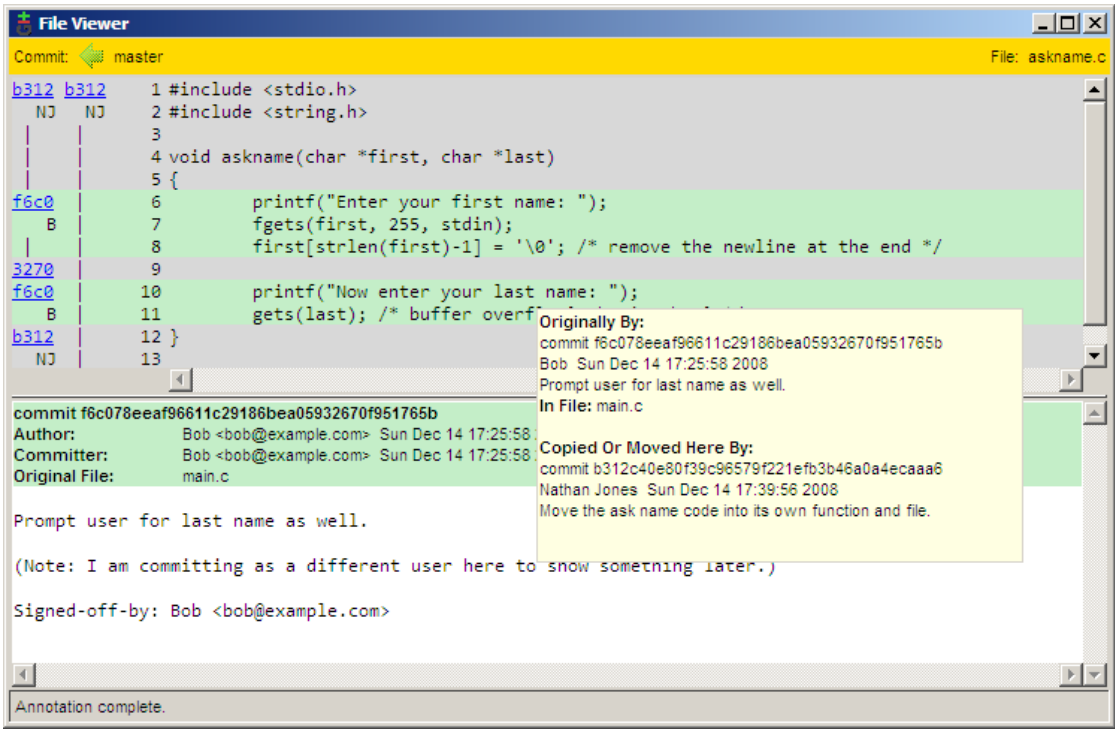
История хранилища может быть просмотрена и изучена выбрав *Repository → Visualize All Branch History*. На следующем скриншоте я пытаюсь найти в какой фиксации была добавлена last переменная, ища все фиксации в которых было добавлено или убрано слово *last*.

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 (revert или reset)

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).

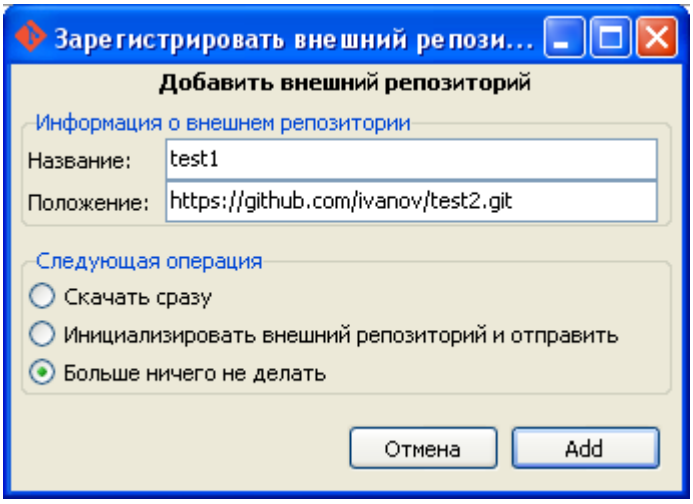
Публикация изменений (pushing) на удалённом сервере

Зарегистрируйтесь на сайте https://github.com/ (укажите логин, почту и пароль, дальше выберите бесплатный доступ).

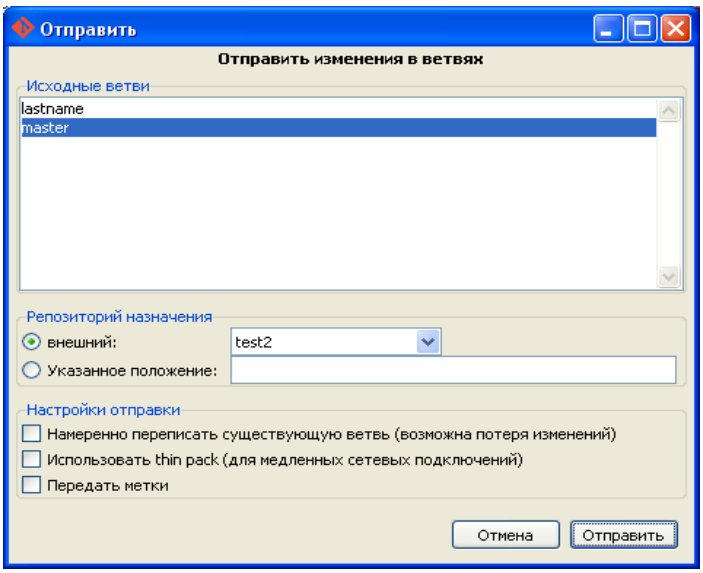
Создайте новый репозиторий: После входа на сайт выберите Create new repository, укажите его название, тип (публичный) и нажмите Create repository.

После создания репозитория будет отображено его имя - адрес (в формате HTTPS) и команды для работы с ним в режиме командной строки (для желающих).

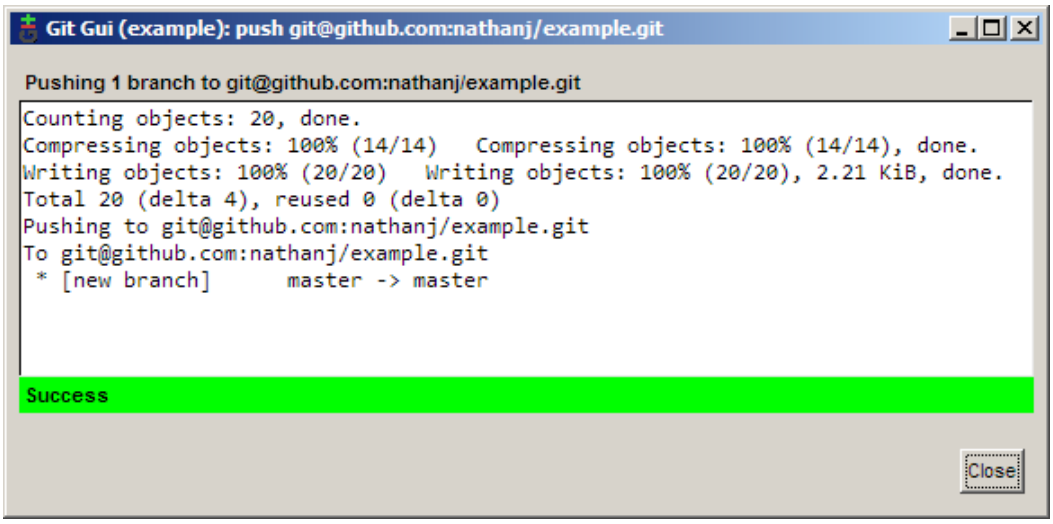
Далее из каталога репозитория на локальном ПК вызовите Git Gui, выберите меню *Внешние репозитории → Добавить,* в новом окне укажите псевдоним удаленного репозитория (любой, на рис. это Test1) и его адрес (https://github.com/ivanov/test2.git, где ivanov – логин пользователя сайта, test2 – название репозитория на сайте). Адрес рекомендую копировать с сайта.



Затем выбрать в меню *Внешние репозитории → Отправить*, указать псевдоним удаленного репозитория и отправляемую ветку, нажать *Отправить*.



При запросе ввести логин и пароль, с которыми регистрировались на сайте.

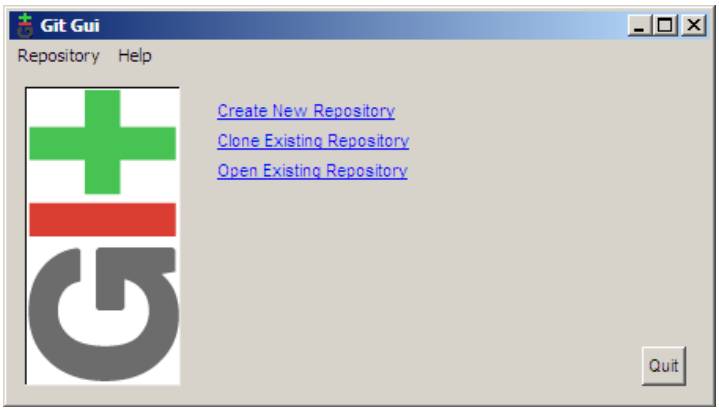


Если все успешно, то на сайте перейти в репозиторий и просмотреть его содержимое.

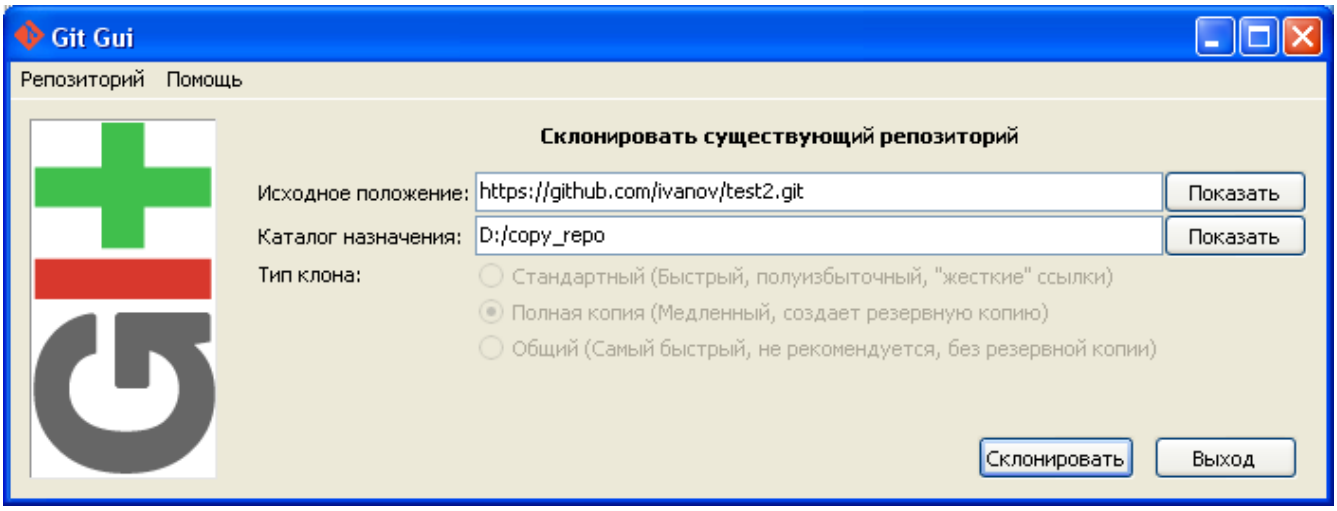
Можно просматривать историю изменений репозитория, содержимое фиксаций, изменения.

Получение изменений (pulling) с удалённого сервера

Для получения копии удаленного репозитория открыть проводник, щелкнуть правой мышью и из контекстного меню выбрать *Git GUI*. Вам будет показан диалог создания.



Выбрать Clone (Клонировать существующий репозиторий). Будет открыт диалог клонирования. В качестве источника укажите удаленный репозиторий (его адрес), в качестве приемника — новый каталог.



Нажать склонировать. Если все успешно, то откроется среда Git Gui, в которой возможно посмотреть содержимое файлов и историю изменений.

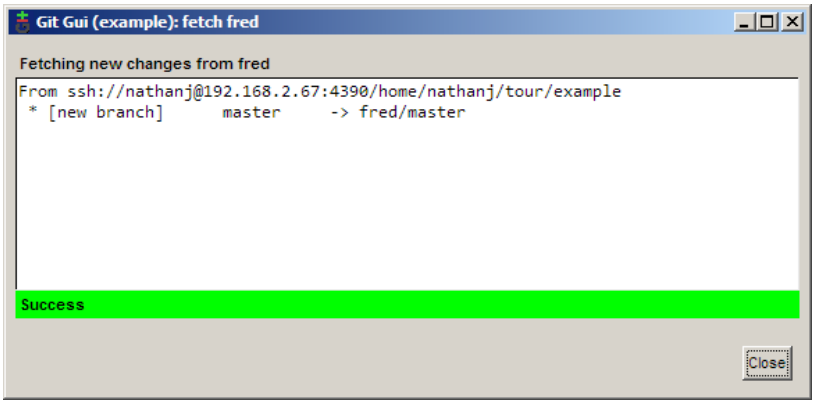
Дальше можно работать над проектами независимо. После внесения изменений и их фиксации отправим изменения обратно на сервер (меню *Внешние репозитории → Отправить*, указать псевдоним удаленного репозитория и отправляемую ветку, нажать *Отправить)*.

Предварительно необходимо разрешение владельца репозитория на внесение изменений. Для этого владелец проекта на сайте выбирает (сверху вверху) New *Collaborator,* откроется страница участников проектов. На ней указать имя добавляемого участника (он должен быть найден по имени или его части) и нажать *Add Collaborator*. Новый участник будет добавлен в список участников.

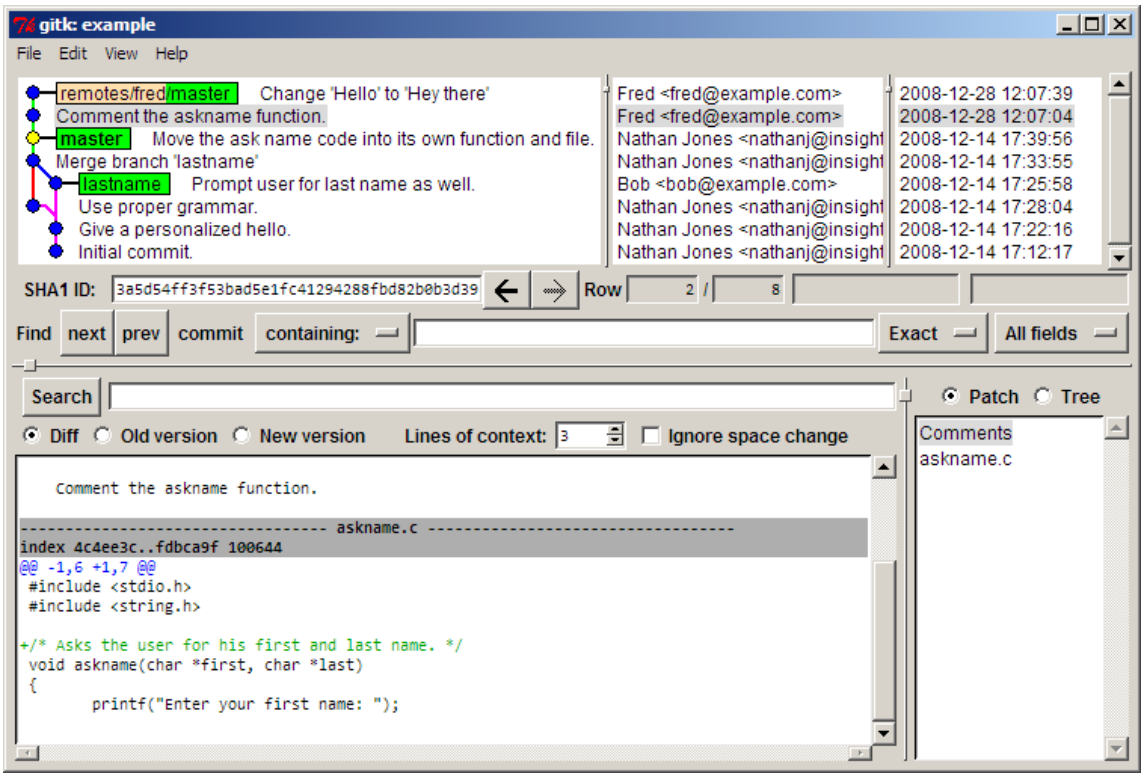
Все участники проекта могут вносить свои изменения, используя адрес репозитория, свои логин и пароль.

Из-за того что наш код такой полезный, несколько человек скачали его и теперь используют нашу программу. А один человек, Фред, даже решил форкунуть его и добавить собственные изменения. Теперь когда он добавил свой код, он хотел бы что бы мы перетянули его изменения в своё хранилище.

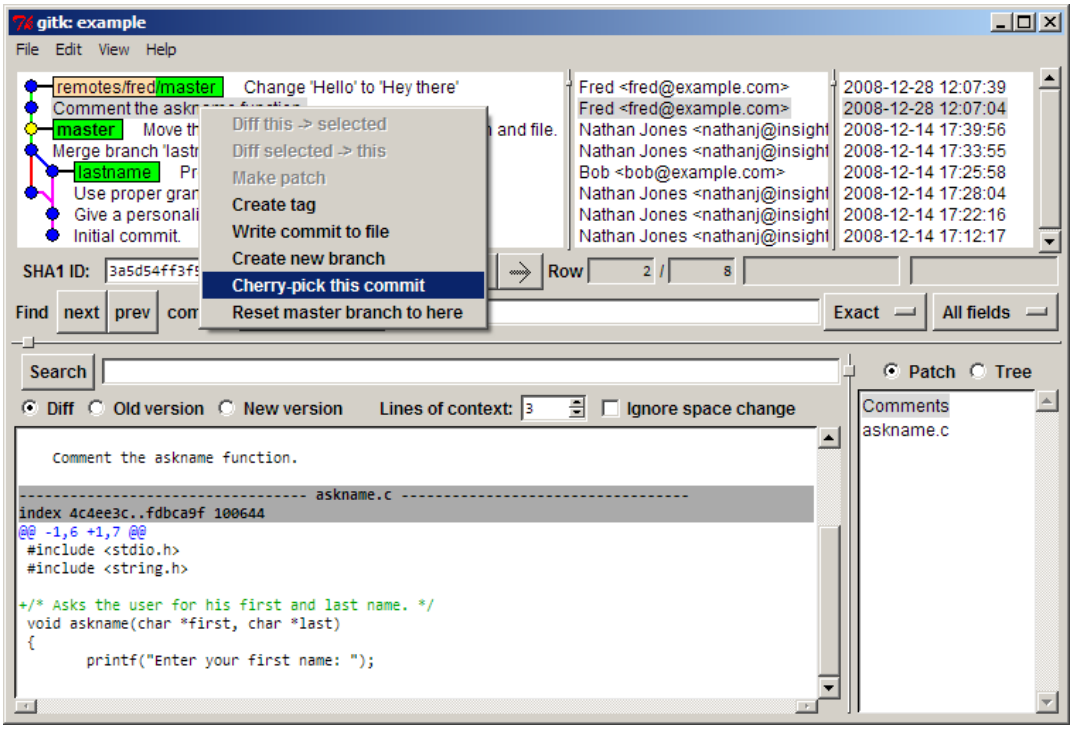
Мы можем получить изменения Фреда, используя *Remote → Fetch from → fred (Внешние репозитории → Получить)*.



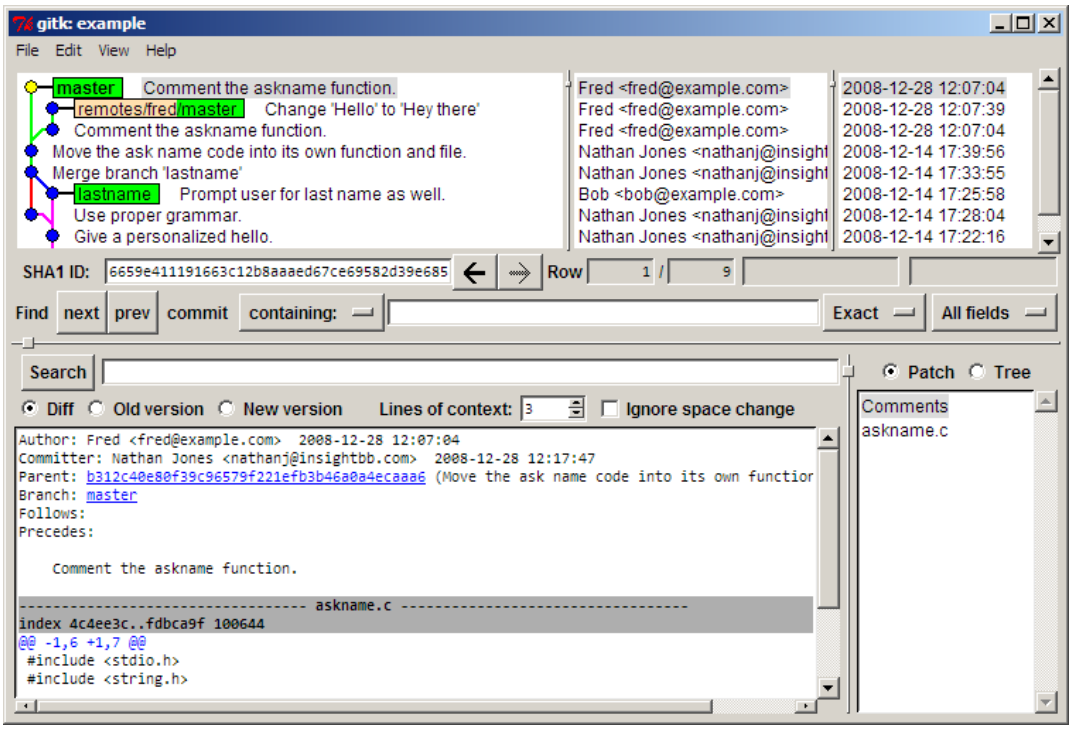
После скачивания, изменения Фреда были добавлены в наше локальное хранилище в remotes/fred/master ветку. Мы можем использовать gitk что бы визуализировать изменения которые сделал Фред.



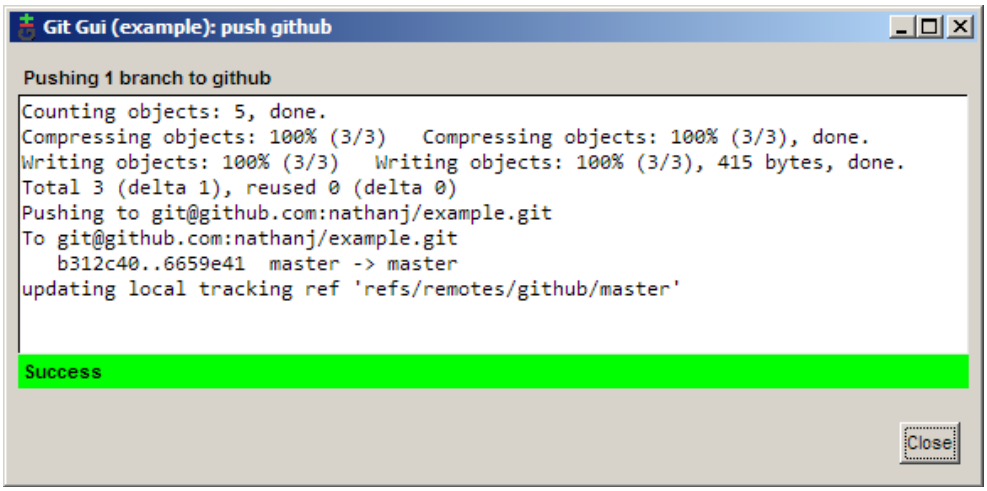
Если нам нравятся все изменения Фреда, мы можем сделать обычное слияние как было показано выше. Но всё же в этом случае, мне нравится только одно изменение Фреда. Что бы влить только одно из изменений Фреда, щёлкните правой кнопкой мыши на выбранной фиксации и выберите *Cherry-pick this commit (Скопировать это состояние)*. Фиксация будет влита в текущую ветку.



Результат:



Теперь мы можем опубликовать изменение Фреда в нашем хранилище на github-е, что бы все могли видеть и использовать его. При отправке надо проставить флаг *Перезаписать ветвь*.



Последовательность работ с удаленным репозиторием

Предположим, что вы и несколько ваших напарников создали общественный репозитарий, чтобы заняться неким общим проектом. Как выглядит самая распространенная для git модель общей работы?

Первым делом создаем создаем копию удаленного репозитария. Далее выполняем итерационно:

1. «Вытягиваем» последние обновления с удаленного репозитория;
2. смотрим, что же изменилось;
3. создаем новую ветвь и переключаемся в нее;
4. работаем в новой ветке, индексируем все изменения и создаем из них коммит;

5. переключаемся в главную ветвь,
6. обновляем ее из удаленного репозитория;
7. проводим слияние с веткой,
8. если есть конфликт, то разрешаем его и делаем коммит слияния.
9. закидываем изменения в удаленный репозитарий