**Лабораторная работа №5. Использование системы контроля версий в процессе разработки ПО**

**Система управления версиями** (от англ. *Version Control System, VCS* или *Revision Control System*) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы.  В данной лабораторной работе будет рассмотрена система управления версиями GIT.

**Установка.**

Для установки необходимо скачать файл устанавливающий GIT систему на ваш компьютер.

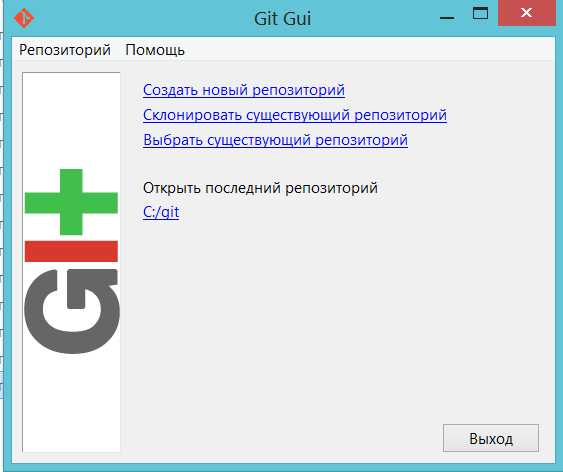
Ссылка для скачивания:

<http://git-scm.com/downloads>

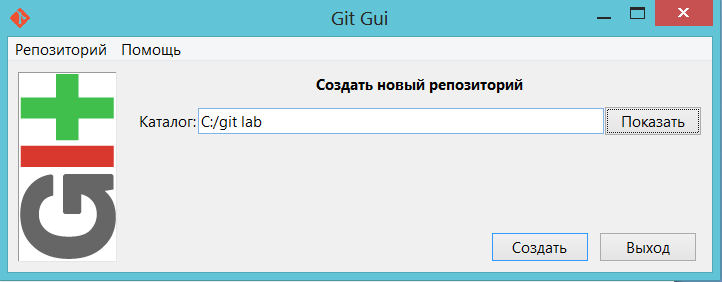
Рассмотрим базовые возможности GIT.

**Создание хранилища**

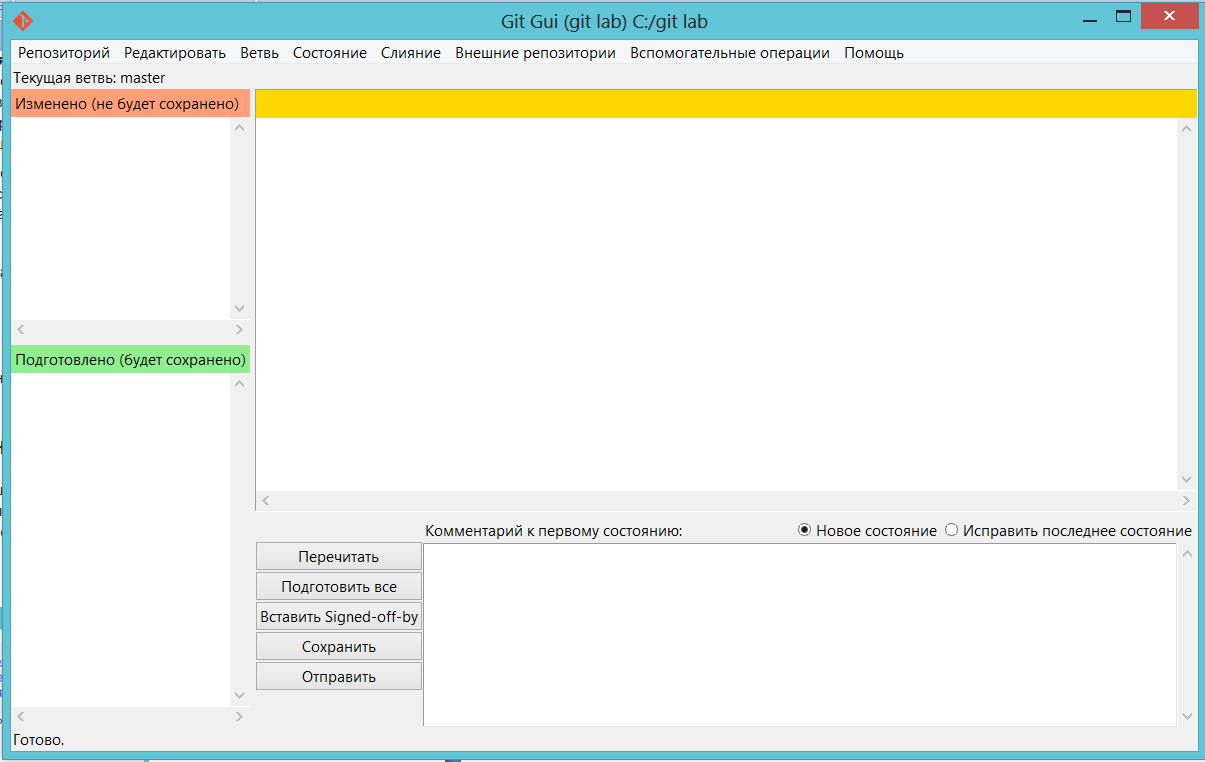
Чтобы создать хранилище, сначала создайте папку для вашего проекта. Далее, правый щелчок мышки на этой папке и выберете Git GUI . Так как в папке пока что не содержится git хранилища вам будет показано окно создания.



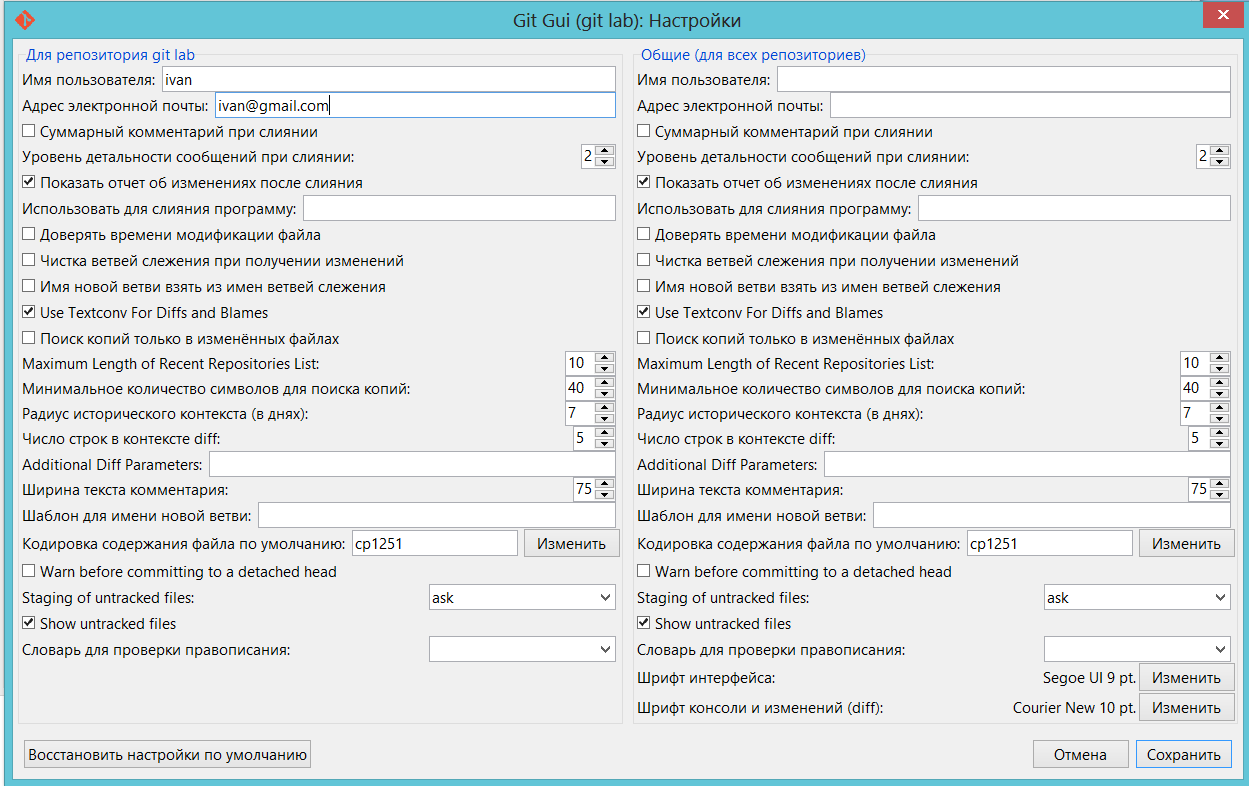
Выбор *Создать новый репозиторий*(Create New Repository) приводит нас к следующему диалогу.



Заполните путь к вашей новой директории и щёлкните *Создать* ( 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, выберите *Редактировать→Настройки* (*Edit → Options)*.



В окне 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 e-mail.

**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:

#include <stdio.h>

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

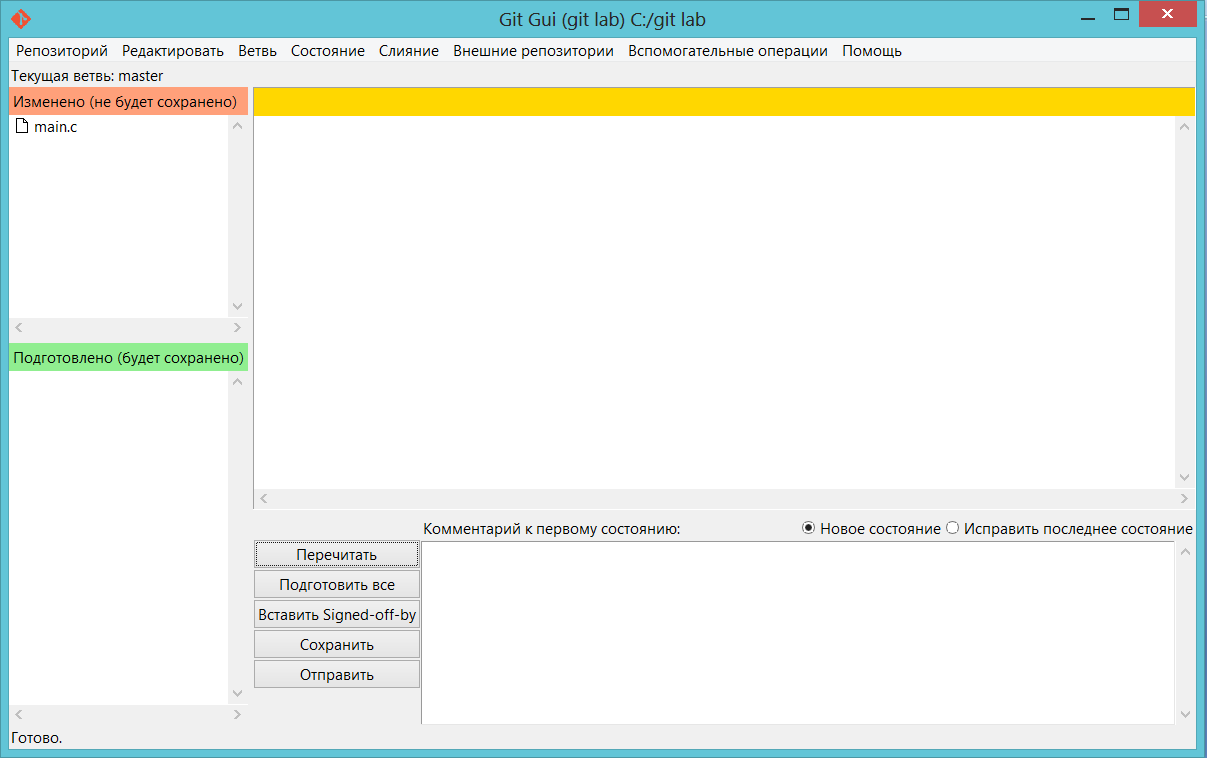
{

printf("Hello world!\n");

return 0;

}

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



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

Говорить 'hello world' это конечно хорошо, но мы хотим, чтобы наша программа была более персонализирована. Давайте скажем 'hello' пользователю. Вот как будет выглядеть код:

#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 \*/

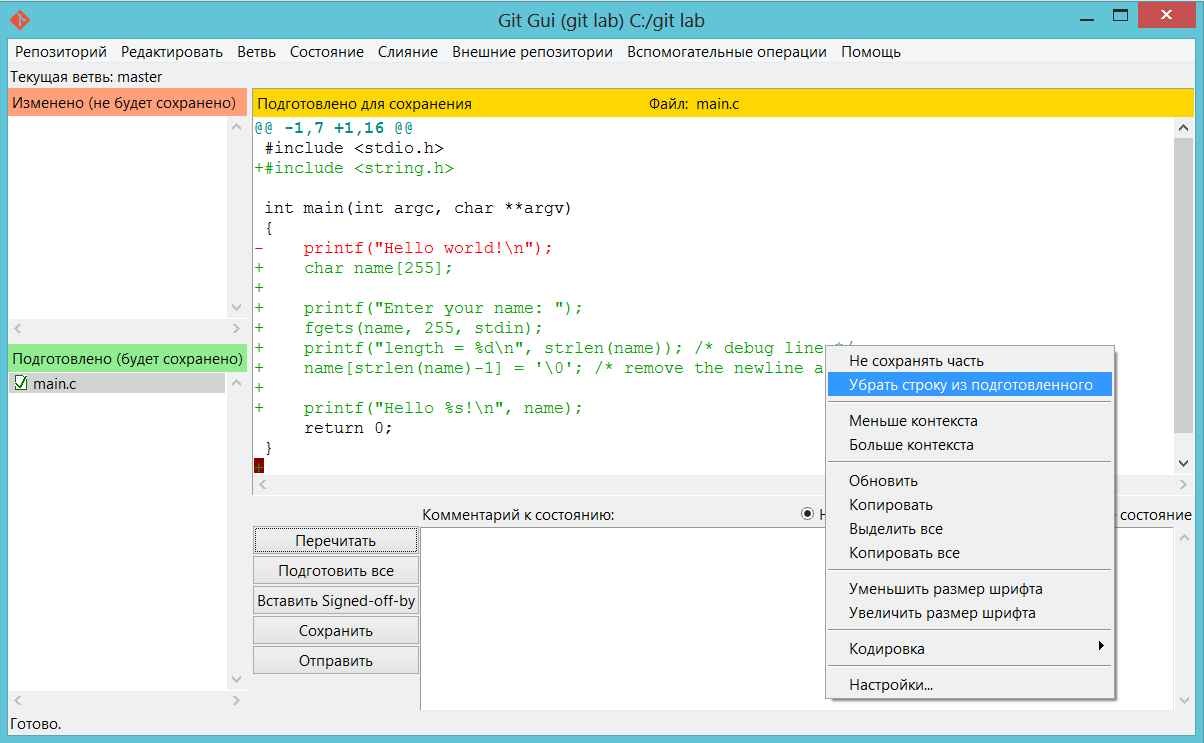
name[strlen(name)-1] = '\0'; /\* remove the newline at the end \*/

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

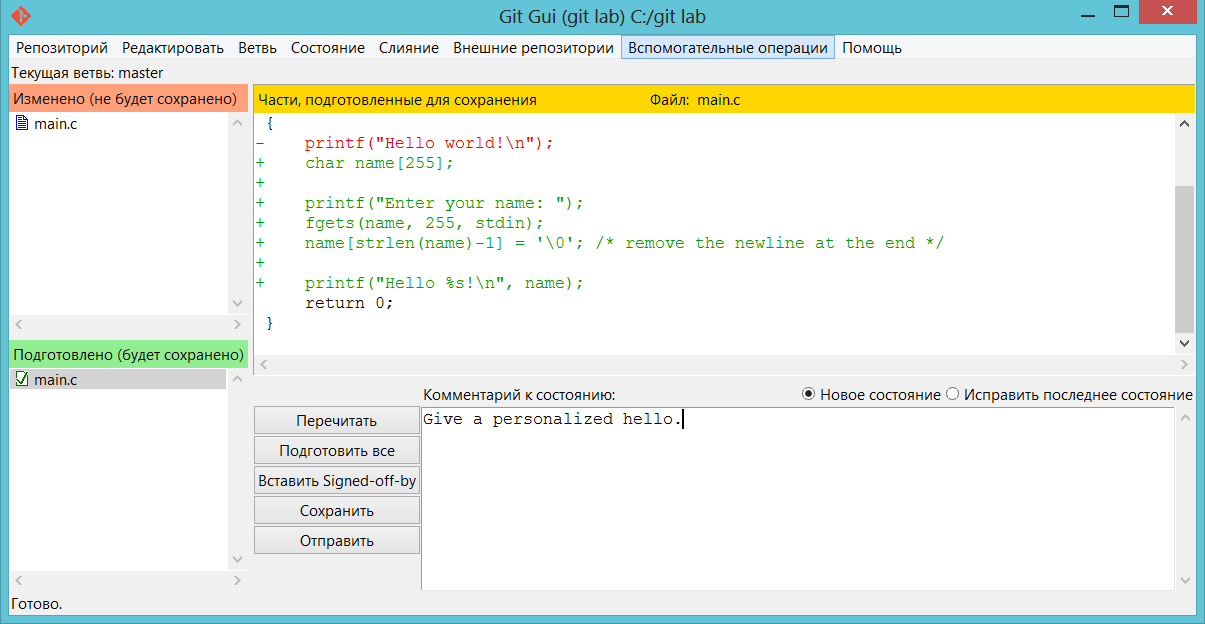
return 0;

}

Были проблемы с тем, что новая линия печаталась после имени пользователя, поэтому добавили отладочную линию кода, что бы помочь себе найти причину. Мы бы хотели зафиксировать изменение без этой отладочной линии, но хотим сохранить эту линию в моей рабочей копии что бы продолжить отладку. С git gui это не проблема. Сначала щёлкните *Перечитать*(Rescan) для поиска изменений. Далее щёлкните на иконке слева от файла, чтобы подготовить (stage) изменения к фиксации. Затем правый щелчок на отладочной линии и выберите *Убрать строку из подготовленного(*Unstage Line From Commit) .

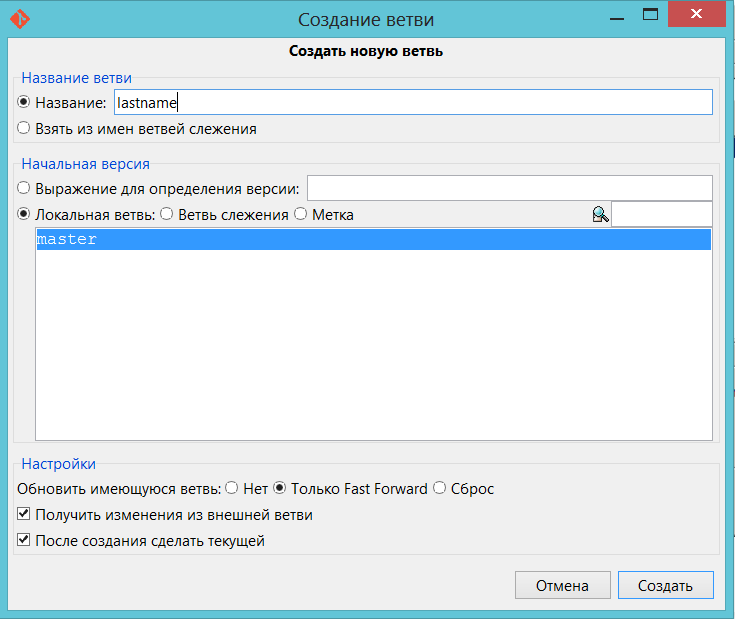


Теперь отладочная линия не была подготовлена (unstaged) к фиксации, в то время как остальные изменения были. Осталось только заполнить сообщение фиксации и зафиксировать изменения щёлкнув по *Сохранить(*Commit).



## 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. Чтобы сделать это мы создадим ветку (branch) для наших новый разработок. Чтобы создать новую ветку в git gui выберете *Ветвь→Создать* (*Branch → Create)*. Новая возможность, которую мы хотим добавить - это возможность спросить пользователя его фамилия, поэтому мы назовем ветку lastname. Опции по умолчанию подходят без изменений, так что просто введите имя и щёлкните *Создать(*Create).



Теперь, когда мы в lastname ветке, мы можем делать наши 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: ");

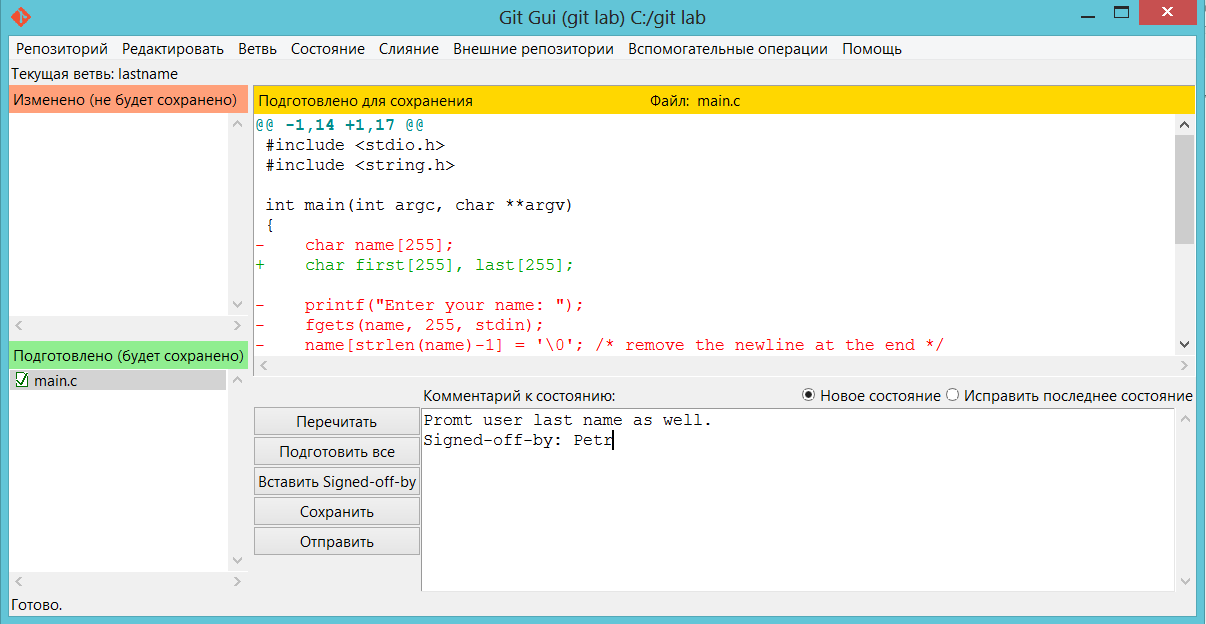
gets(last); /\* buffer overflow? what's that? \*/

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

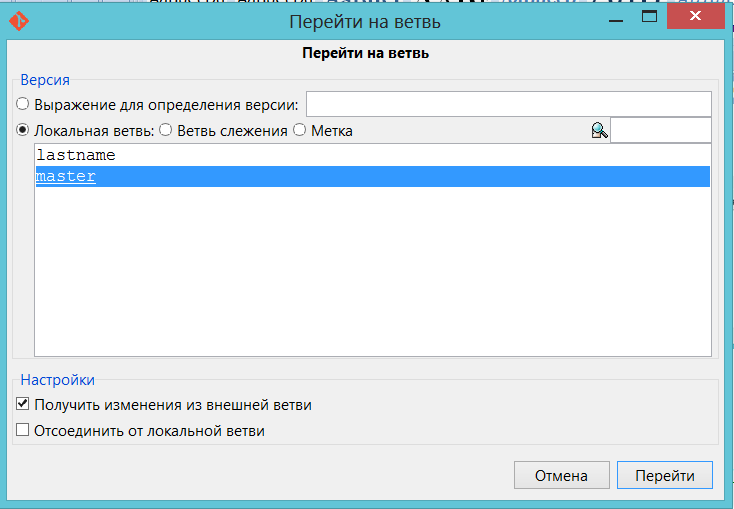
return 0;

}

Теперь мы можем 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.

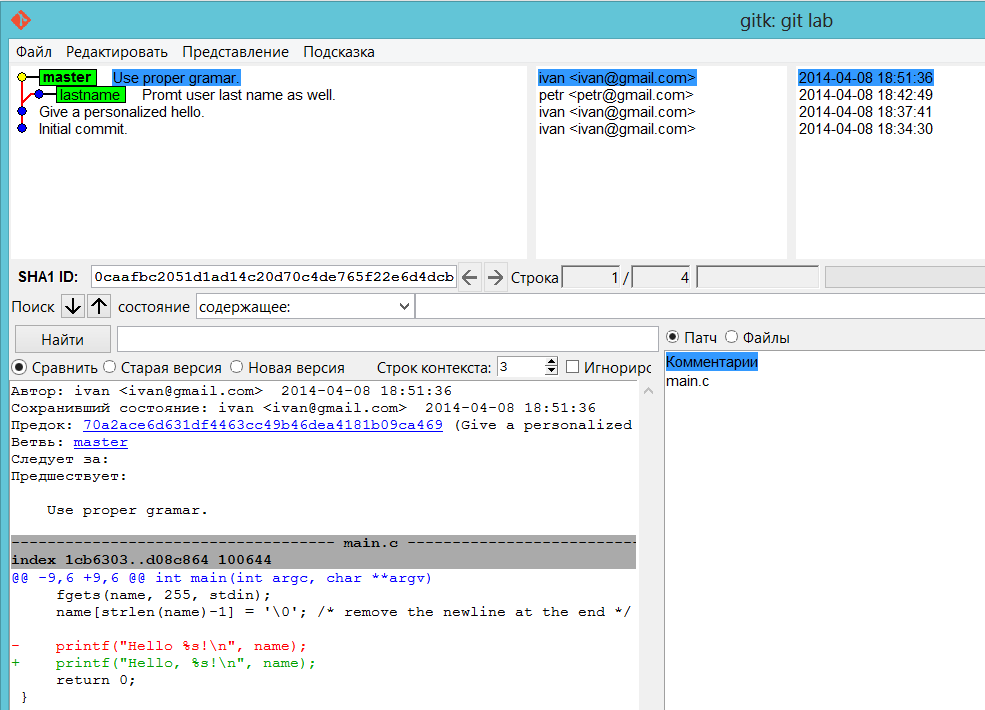


Между тем, пользователь проинформировал нас, что непоказ запятой после прямого обращения к кому-то это серьёзная ошибка. Что бы исправить её в нашей стабильной ветке, мы сначала должны переключится назад на неё. Это достигается используя *Ветвь→Перейти(Branch → Checkout)*.



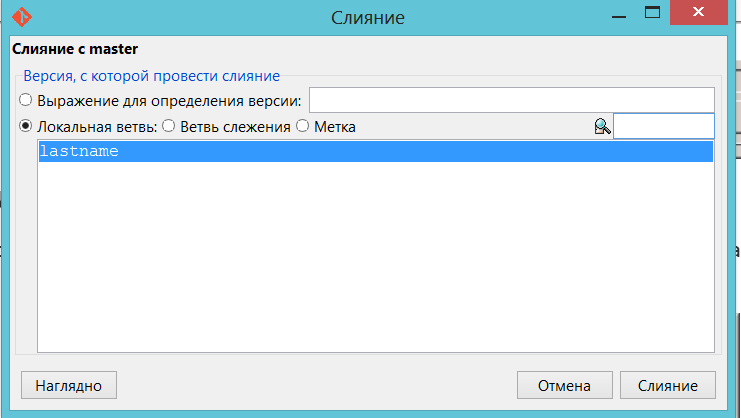
Теперь мы можем исправить нашу большую ошибку.

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

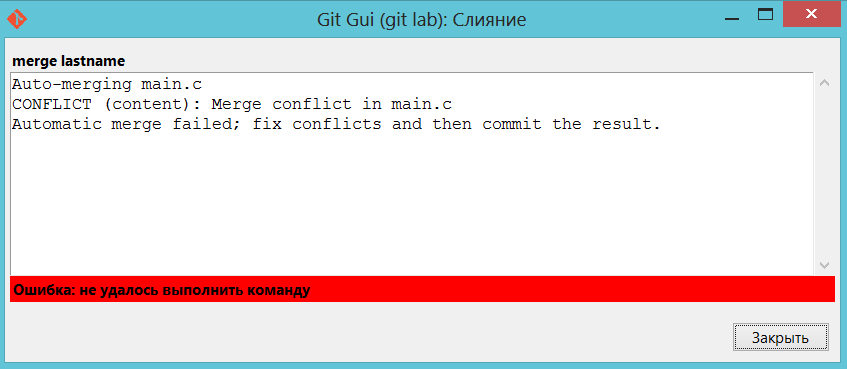


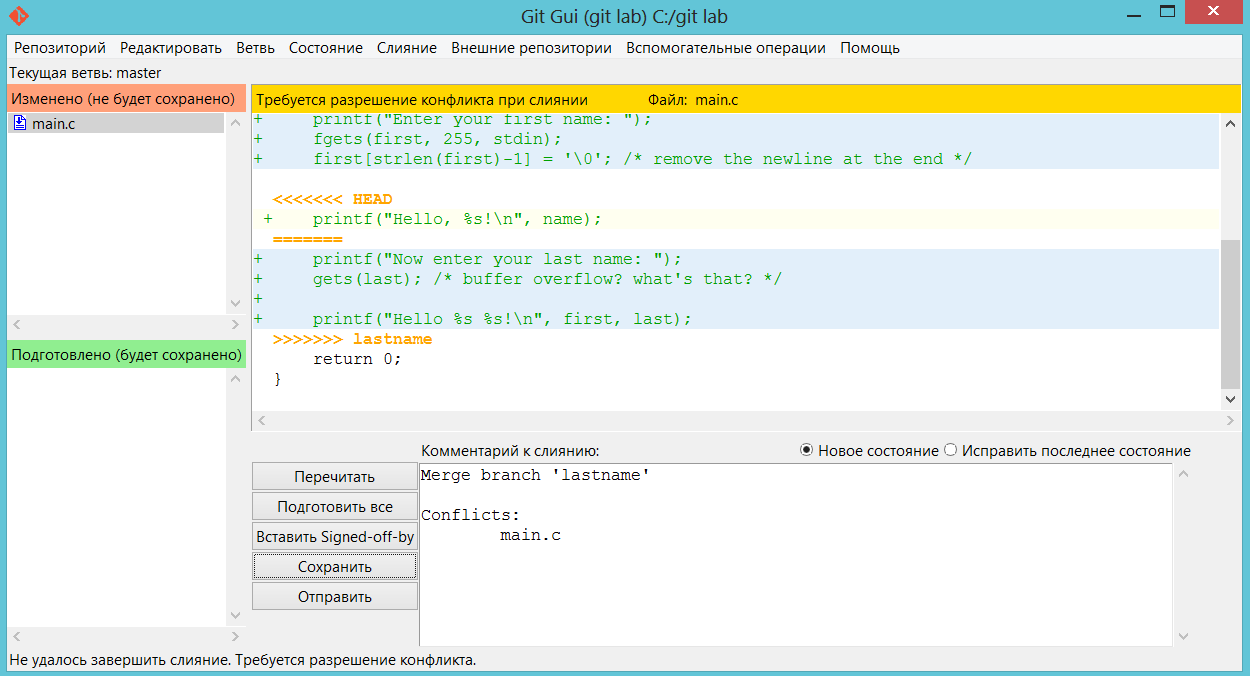
## Слияние (merging)

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

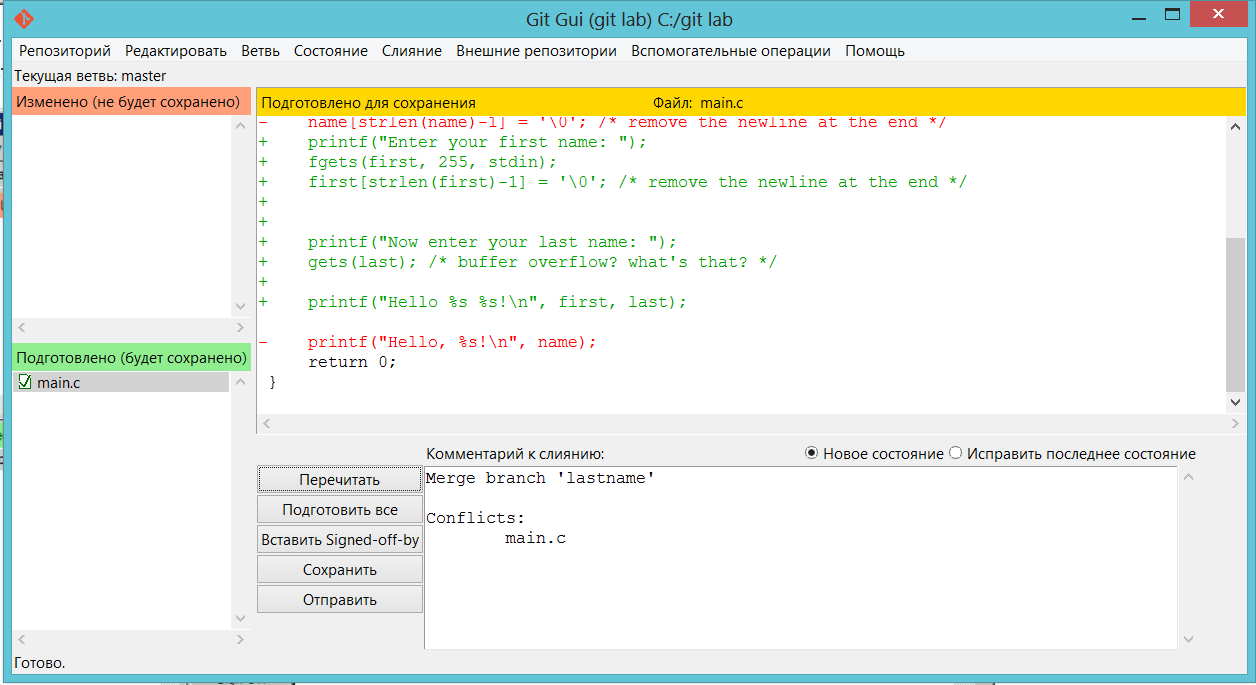


Так как две разных фиксации делали два разных изменения на одной и той же линии, происходит конфликт (conflict).





Конфликт может быть разрешён с помощью любого текстового редактора. После разрешения конфликта, подготовьте изменения щёлкнув на иконке файла и зафиксируйте слияние щёлкнув по кнопке *Сохранить(*Commit*)*.



## Просмотр истории

Файл main.c становится большим, поэтому мы решили вынести код, спрашивающий имя пользователя в отдельную функцию. Пока мы это делали, решили 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: ");

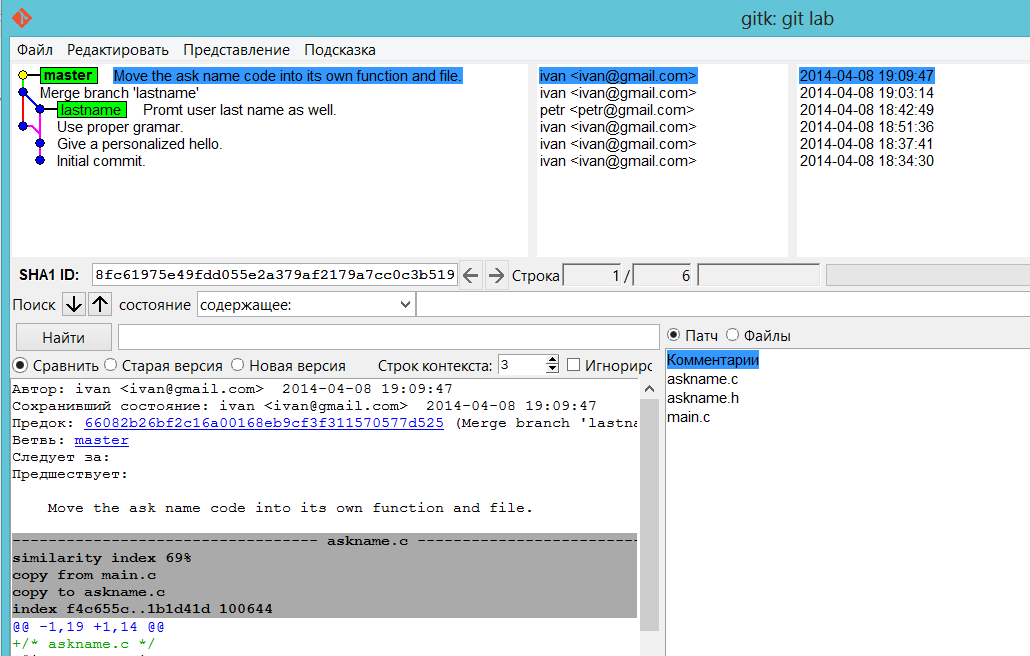
gets(last); /\* buffer overflow? what's that? \*/

}

/\* askname.h \*/

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

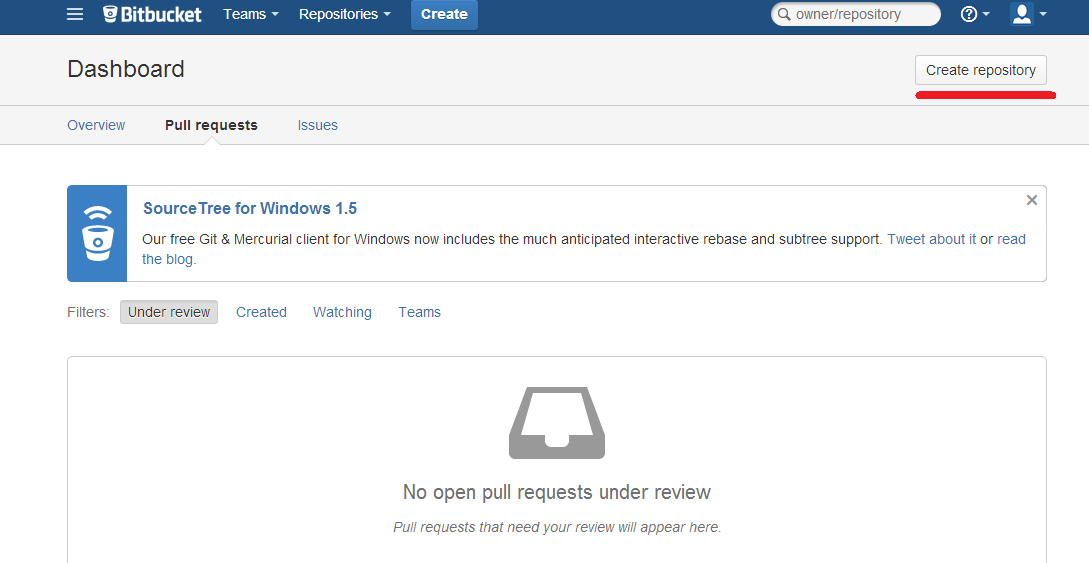




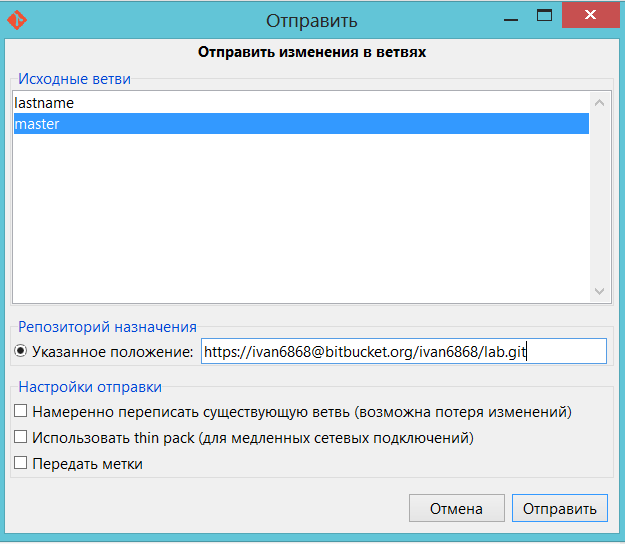
**Публикация изменений (pushing) на удалённом сервере**

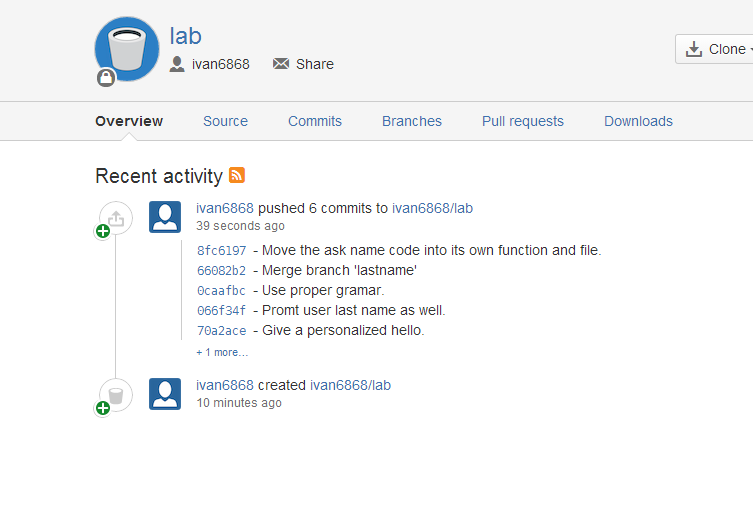
Имеется множество различных хостингов GIT. Мы рассмотрим BitBucket(<https://bitbucket.org/>) позволяет бесплатно создавать как открытые, так и закрытые репозитории.

После регистрации необходимо создать репозиторий:

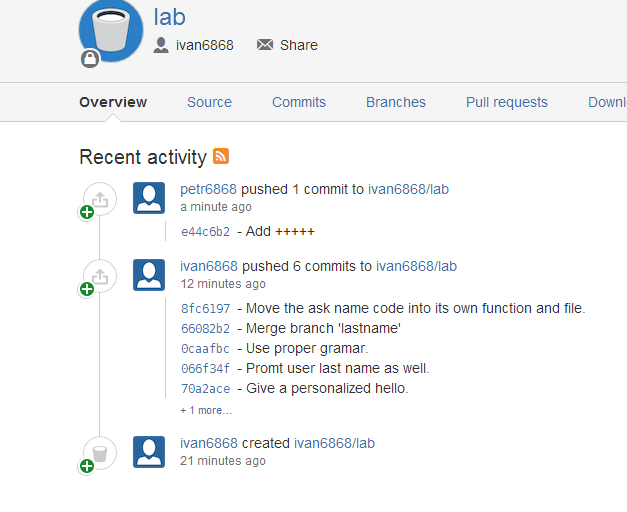


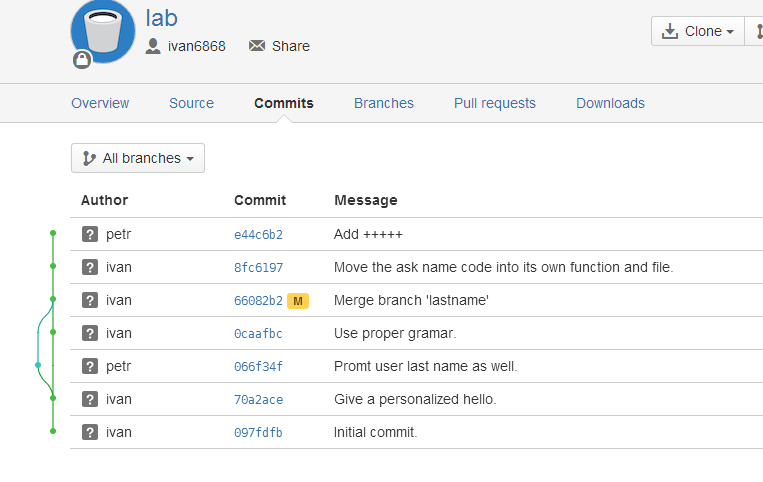
После создания репозитория необходимо отправить свои данные, для этого в git gui выбираем *Внешние репозитории→Отправить(Remote → Push).* Наберите адрес публикации проекта и щёлкните *Отправить* (Push), это вышлет изменения.



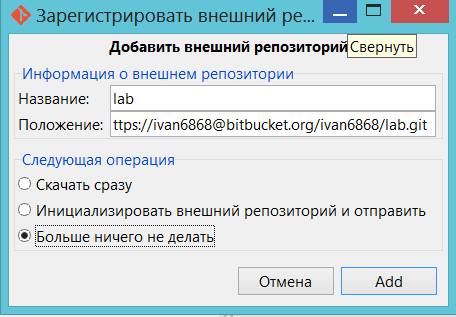
**

Теперь мы хотим обеспечить возможность работы с нашим репозиторием другому пользователю, для этого выбираем *Send Invitation. После внесения изменений вторым пользователем:*

**

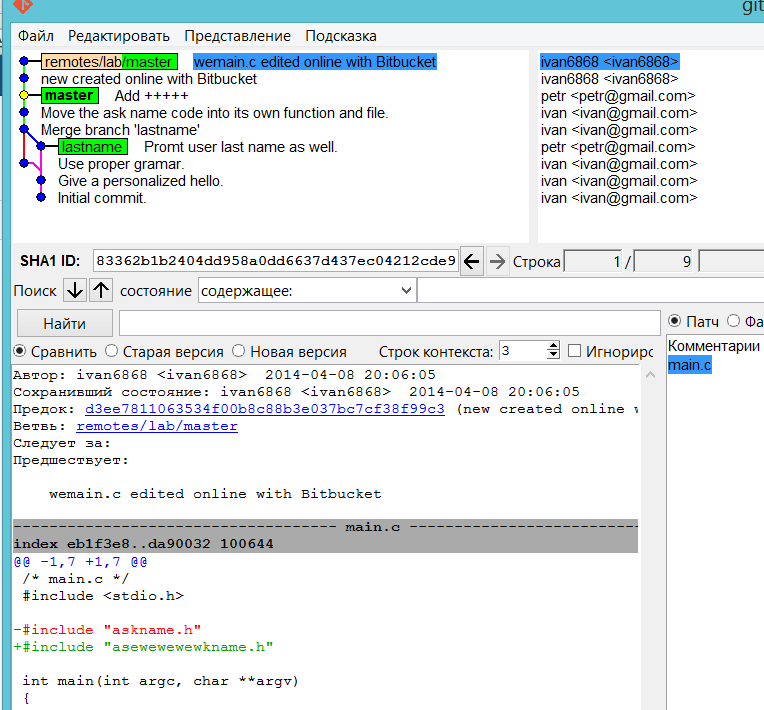
**

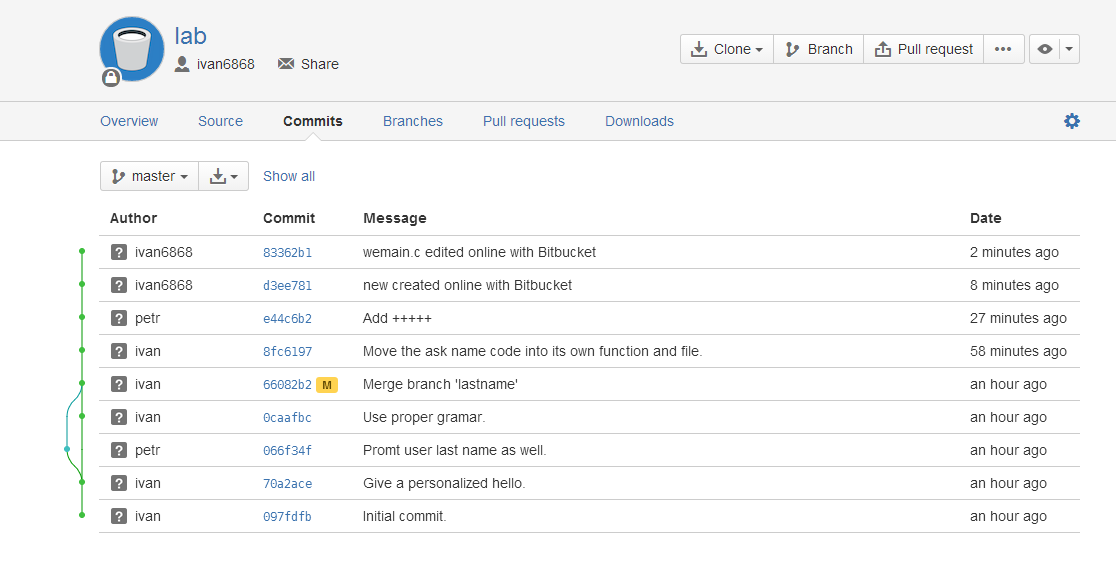
Для того, чтобы каждый раз не прописывать адрес, можно выбрать *Внешние репозитории→Добавить*

**

**Получение изменений (pulling) с удалённого сервера**

Для получения изменений с удаленного сервера нажмем *Внешние репозитории → Получить из(Remote → Fetch from)*

**

**

**Лабораторное задание:**

1. Если система контоля версий Git не установлена, то установить ее (параметры оставить по умолчанию).
2. Запустить Git GUI (или консоль). Создать новый репозиторий. Добавить в папку репозитория файлы. Зафиксировать состояние репозитория (выполнить commit).
3. Внести изменения в файлы. Зафиксировать новое состояние репозитория.
4. Создать новую ветку 1. Внести в нее изменения и зафиксировать их.
5. Переключиться на ветку мастера. Внести в нее изменения и зафиксировать их.
6. Продемонстрировать слияние веток.
7. Просмотреть дерево изменений веток (историю).
8. Создать удаленный репозиторий.
9. Загрузить на него свой проект.
10. Обеспечить доступ к нему все членам команды.
11. Скачать каждым членом команды файлы с удаленного репозитория, произвести изменения и совершить слияние.