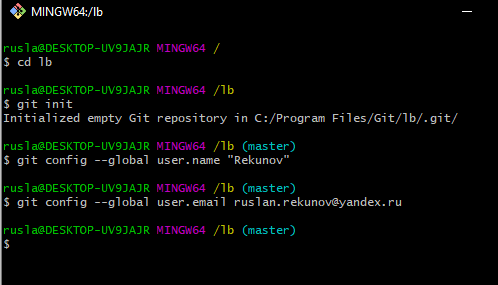
# Лабораторная работа №1 Система контроля версий git

**Цель работы**

1. Изучить на практике понятия и компоненты систем контроля версий (СКВ), приемы работы с ними.
2. Освоить специализированное ПО и распространенный сервис для работы с распределенной СКВ Git — git-bash и GitHub.com.

**Задание**

1. Отработать навыки использования хранилища на локальной машине.
   1. Настроить Git, указав имя и e-mail разработчика для подписи commit-ов.

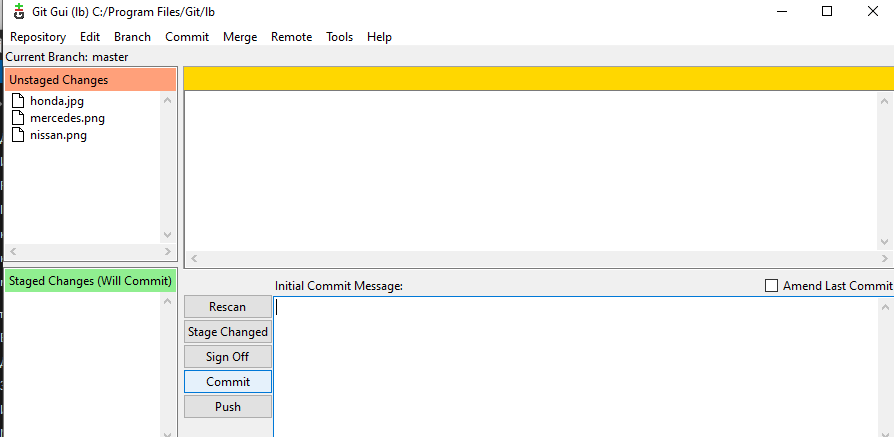


Для перехода в папку репозитория и для работы с ней, необходимо ввести команду cd «Название папки».

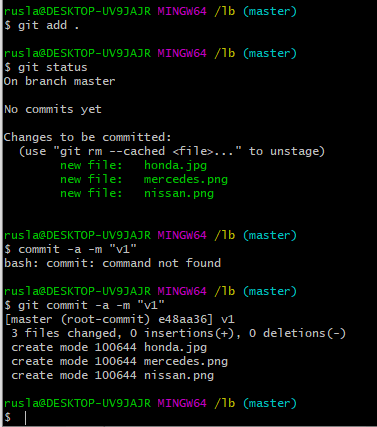
Созданную папку необходимо подключить к системе контроля версий, для этого вводится команда «git init».

Для создания своего локального пользователя, необходимо указать имя и почту. Чтоб указать имя, необходимо ввести команду «git config –global user.name «Name». Для указания почты, «git config –global user.email «email».

* 1. Создать хранилище для учебного проекта.



* 1. Совершить несколько commit-ов.

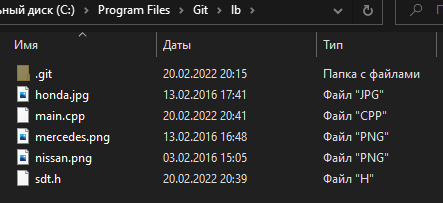
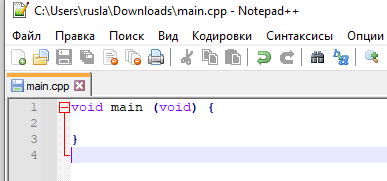


Добавляем в хранилище (папку) несколько файлов. После добавления файлов в папку, вводим команду «git add .», прописывая точку мы берем сразу все файлы. Данная команда позволяет внести изменения в хранилище, которые затем войдут в коммит.

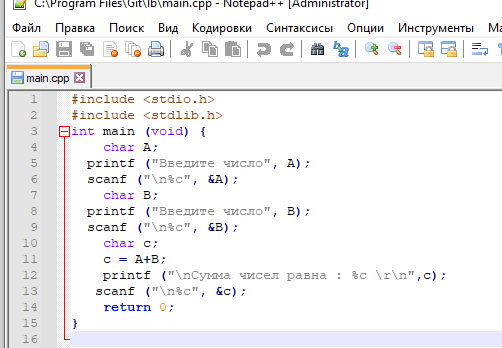
Вводим команду «git status» которая позволяет просмотреть состояние проекта, она выводит информацию обо всех изменениях, внесенных в дерево директорий проекта.

После чего совершаем коммит при помощи команды «commit –a –m «v1». Параметр «-a» совершит коммит, автоматически индексируя изменения в файлах проекта. Новые файлы индексироваться не будут. Параметр –m «v1» комментирует коммит из командной строки.

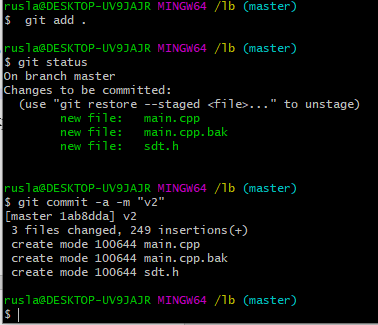
* + 1. Скопировать sdt.h в каталог хранилища и создать файл main.cpp с включением sdt.h и пустой функцией main().



1. Добавить в программу ввод двух целых чисел с приглашением.

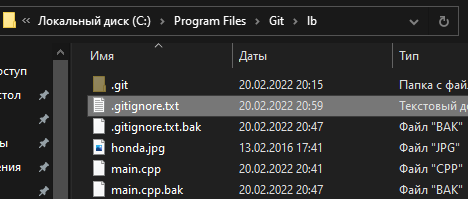


Делаем коммит после изменений.

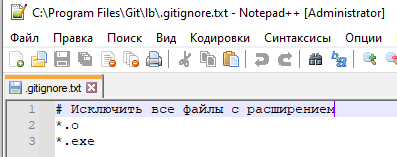


1.4. Предотвратить автоматическое добавление в хранилище файлов, не нуждающихся в контроле версий, — \*.o и \*.exe.

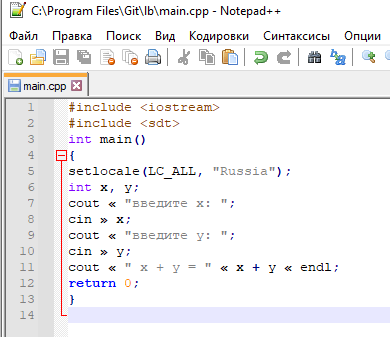
Создаем файл .gitignore.txt.



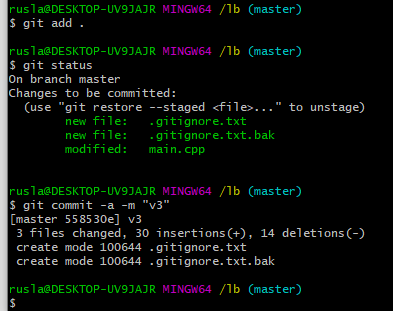
Добавляем в исключение расширения.



* 1. Добавить в программу вывод суммы введенных чисел и совершить commit.

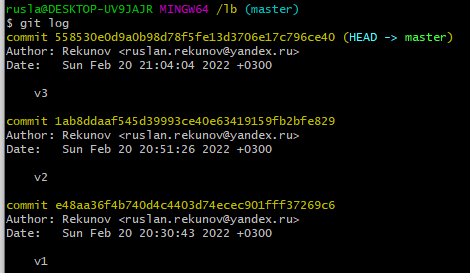


Совершаем коммит после изменений.



* 1. Просмотреть историю (журнал) хранилища.

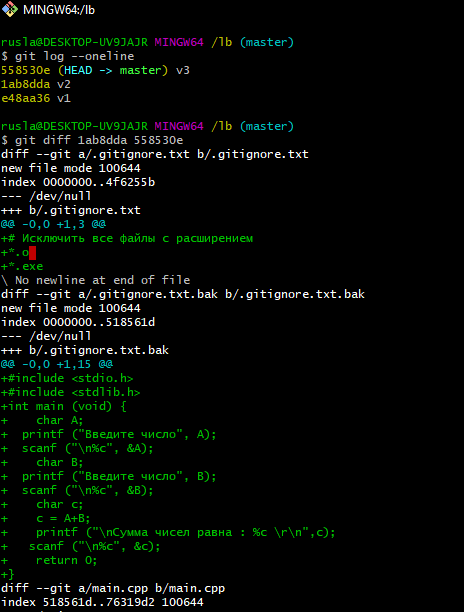
При помощи команды «git log» просматривает историю хранилища.



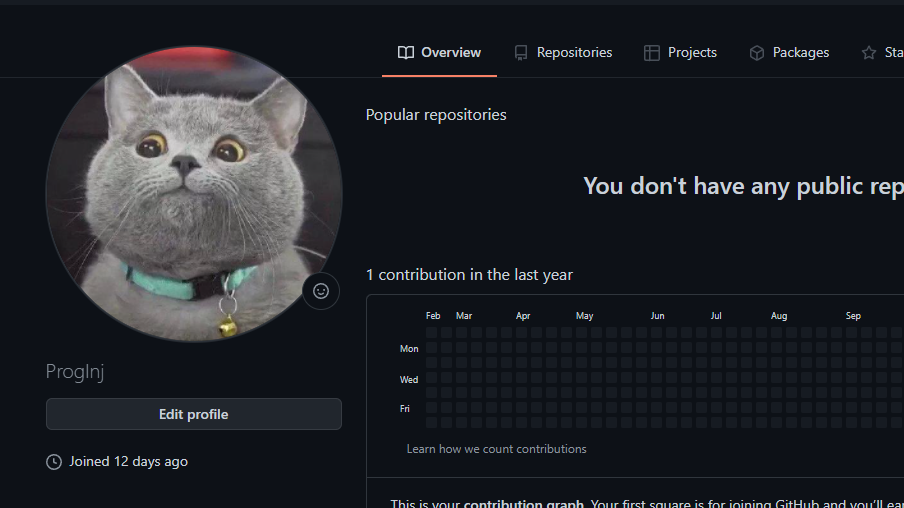
* 1. Просмотреть разность (diff) между пунктами истории [1.3.2](#page2) и [1.5.](#page2)

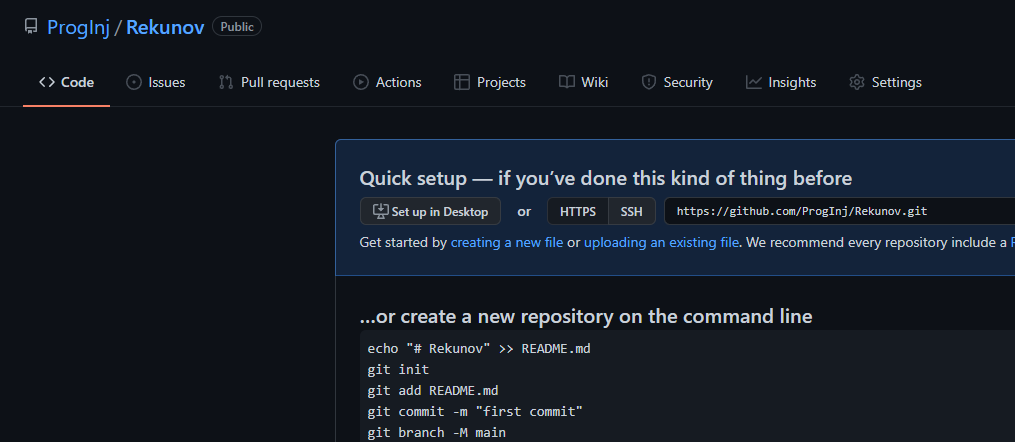
При помощи команды «git log –oneline» определяем хэш (ID) коммитов, которые требуется сравнивать.

Вводим команду «git diff id id» (id хэш коммитов) и выводится сравнение двух веток.



1. Освоить передачу истории хранилища по сети.
   1. Организовать общее хранилище на удаленном сервере.
      1. Зарегистрироваться на [GitHub.](https://github.com/)



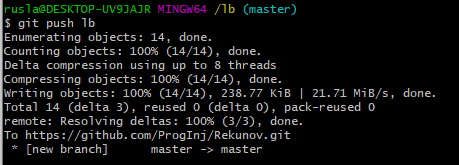
* + 1. Создать пустое удаленное хранилище с любым наименованием.

2.1.3. Настроить локальное хранилище для синхронизации с удаленным.

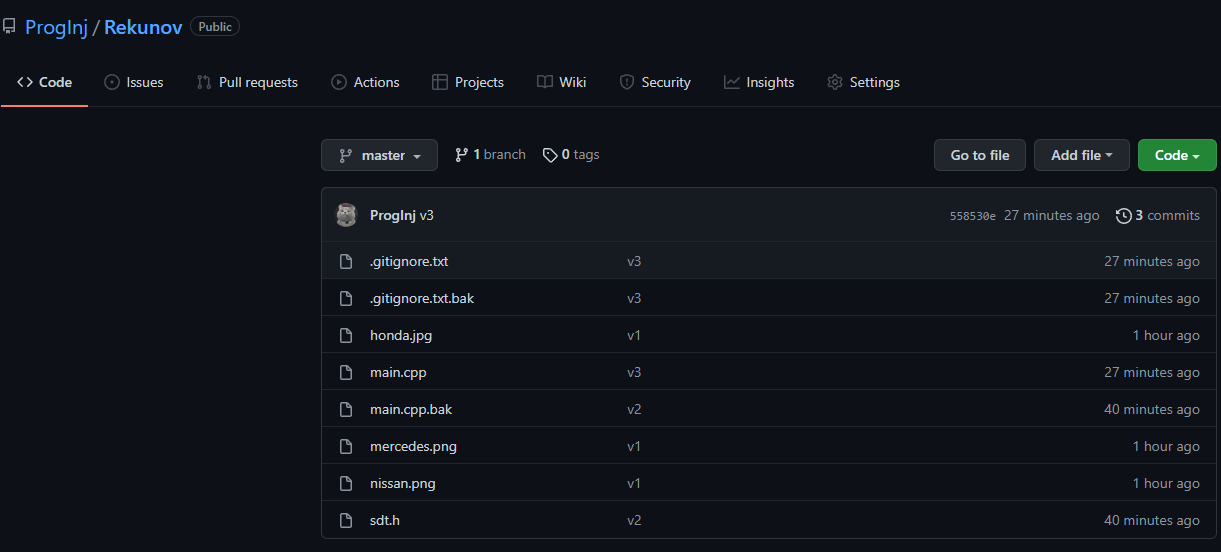
Прописываем команду «git remote add «name» «url»» для создания нового подключения к удаленному репозиторию.



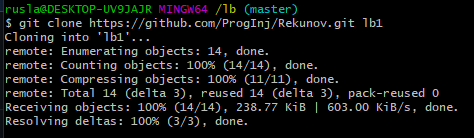
1. Передать локальное хранилище на удаленный сервер (push).



1. Перейти к странице хранилища на GitHub (обновить её) и ознакомиться с возможностями просмотра содержимого через web-интерфейс.

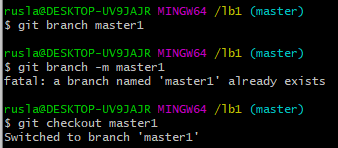


1. Загрузить копию удаленного хранилища на локальную машину (clone).

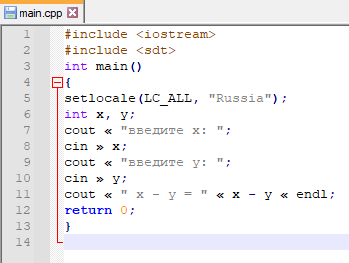


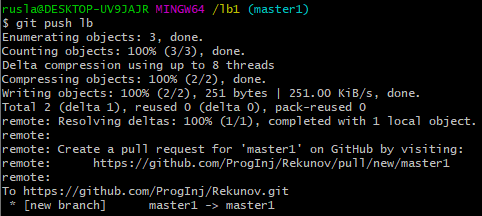
1. Сымитировать параллельную работу над проектом.

После загрузки копии с удаленного хранилища, создаем новую ветку «git branch name» и переключаемся на нее «git checkout name».

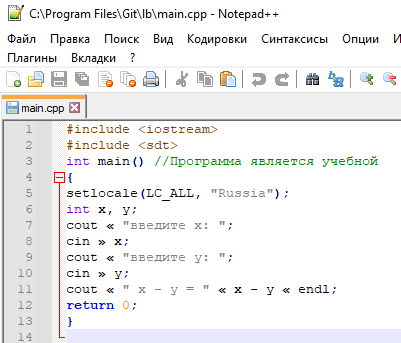


* 1. В локальном хранилище RepoB добавить в программу печать разности введенных чисел, сделать commit и передать изменения на сервер.

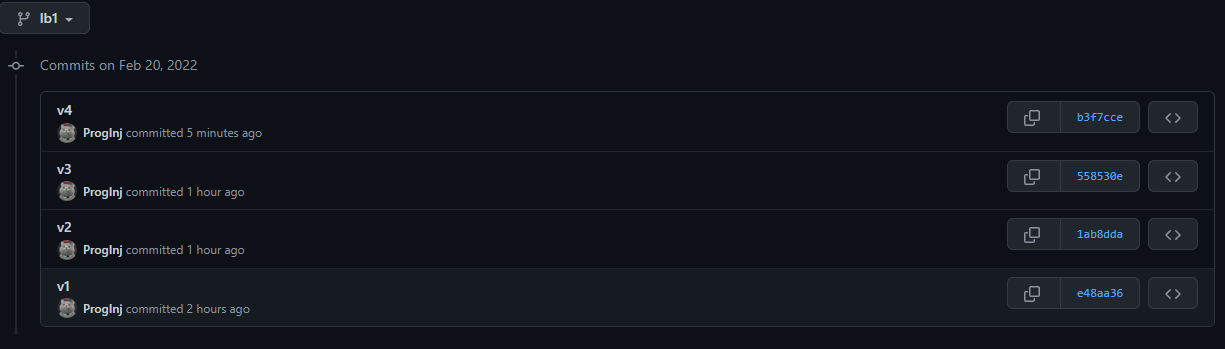
****

****

* 1. В локальном хранилище RepoA добавить над функцией main() комментарий о том, что программа является учебной, сделать commit, **но не отправлять изменений на сервер.**



1. На странице хранилища на GitHub перейти в раздел *Commits* и ознакомиться с возможностью просмотра истории изменений через web-интерфейс.



1. В локальном хранилище RepoA выполнить загрузку с сервера новейших ветвлений и изменений (fetch) и просмотреть журнал хранилища.

*Указание.* По-умолчанию показывается только текущая активная ветвь (по умолчанию — master). Просмотреть все commit-ы во всех ветвях, в том числе в загруженной из удаленного хранилища ветви origin/master, нужно включить флажок «All branches» слева снизу окна журнала.

1. Совместить изменения в локальном хранилище с загруженными.
   1. Использовать действие pull для загрузки изменений с удаленного сервера и автоматического совмещения их с имеющимися локально. Просмотреть журнал изменений.

*Примечание.* Фактически,при обновлении производится слияние ветвейmaster и origin/master — то есть, двух версий истории, существовавших удаленно и локально. При этом история стала нелинейной и появился лишний commit слияния.

Иногда такое усложнение имеет смысл, но в данном случае было бы желательно сохранить историю линейной и просто перенести локальные наработки вслед за новейшими. Добьемся желаемого.

2.8.2. Отменить неудобный результат действия pull.

*Указание.* В журнале изменений в контекстном менюcommit-а,где былдобавлен комментарий (то есть, последнего перед слиянием), выбрать «Reset master to this…» и указать тип отмены «Hard».

1. Выполнить перенос (rebase) локальных изменений на основу новейшего загруженного состояния проекта.
2. Передать итоговое состояние локального хранилища RepoA на удаленный сервер, используя команду push.
3. Действуя аналогично п.п. [2.7](#page3) и [2.8.3,](#page4) синхронизировать с удаленным локальное хранилище RepoB (в нем не хватает commit-а с комментарием).

*Замечание.* На данном этапе во всех трех хранилищах(локальныхRepoAиRepoB и удаленном на GitHub) должна быть одинаковая линейная история из пяти — шести commit-ов.

1. Изучить действия, связанные с ветвлениями и разрешением конфликтов.

*Замечание.* Все действия выполняются в одном локальном хранилище,например,вRepoA.

* 1. Добавить в программу печать произведения чисел и совершите commit.

На данном этапе программа может быть такой:

#include "sdt.h"

// This program is just an example one under VCS.

int main()

{

int a, b;

cout << "Enter A and B: ";

cin >> a >> b;

cout << "A + B = " << a + b << '\n'

cout << "A - B = " << a - b << '\n'

cout << "A \* B = " << a \* b << '\n';

}

1. Создать новую ветвь (branch) под названием division. из пункта истории, в котором был добавлен комментарий над main(). 
2. В новой ветви повторить пункт [3.1,](#page4) заменив умножение делением.

Переключаться между ветвями можно только при чистом (clean) хранилище, то есть, без изменений в рабочей копии.

1. Переключиться обратно на ветвь master.
2. Выполнить слияние ветви division в ветвь master так, чтобы в последней оказался код для печати и произведения, и частного. 
   1. В журнале изменений в контекстном меню пункта-окончания ветви division выбрать пункт «Merge into "master"…» и начать слияние, не меняя настроек.

Действие завершится ошибкой из-за конфликта (conflict): в файле main.cpp строка 10 изменена в обоих commit-ах одинаково, а строка 11 — по-разному, и СКВ не может автоматически выбрать «правильный» вариант. Требуется вручную указать, какие строки должны войти в итоговую версию файла.

3.5.2. Приступить к разрешению конфликта.

*Примечание.* Редактор конфликтов похож на программу для просмотраразностей между файлами: слева показывается файл в ветви, откуда делается слияние (division), справа — ветви, куда делается слияние (master), снизу — результат слияния. Знаками равенства в соседних верхних полях отмечаются не только строки, оставшиеся неизменными, но и строки с одинаковыми изменениями: здесь, в строке 10 убрана точка с запятой в конце в обеих ветвях.

В нижнем поле каждый восклицательный знак обозначает отдельный конфликт.

Примерный вид окна редактирования конфликтов представлен на рисунке ниже.

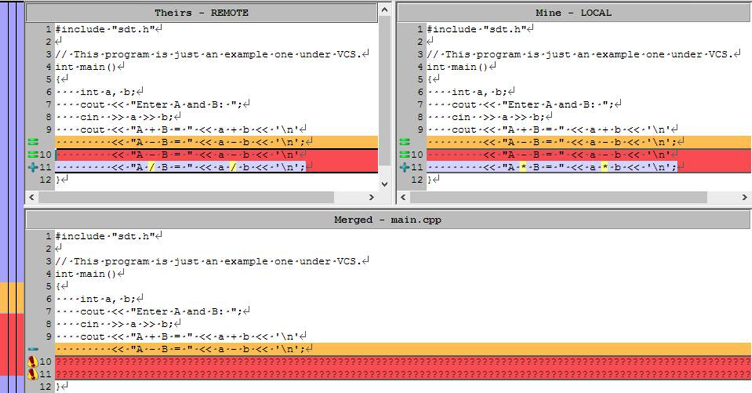


Рис.1 – Вид окна редактирования конфликтов

1. Разрешить конфликты:

— в качестве строки 10 предпочесть строку 10 любой ветви;

— на место строки 11 вставить 2 строки: строку с печатью произведения и строку с печатью частного;

— лишнюю точку с запятой на строке 11 поля-результата удалить.

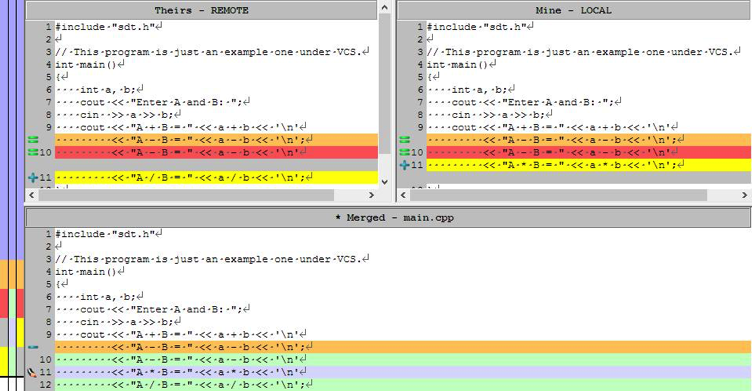


Рис.2 – Вид окна редактирования конфликтов при слиянии правок

3.5.4. Завершить процедуру разрешения конфликтов.

1. Завершить слияние ветви division в ветвь master, написав осмысленный комментарий к слиянию и совершив commit.
2. Убедиться, что программа компилируется и верно работает. Если это не так, исправить все ошибки и добиться правильной работы. Совершить commit.

*Замечание-указание.* Ситуация,когда после слияния программа все-такиоказывается не вполне корректной, случается на практике довольно часто. В этом случае commit, созданный при слиянии, оказывается логически неправильным, он не имеет ценности без последующего исправления. В Git имеется возможность изменить (amend) уже совершенный commit, пока он не передан на сервер. Это делается при следующем commit-исправлении: следует установить флажок «Amend Last Commit» в диалоге commit — нового commit не появится, а вместо этого изменения будут приписаны предыдущему пункту истории. Можно воспользоваться данной возможностью при выполнении пункта.

3.5.7. Передать все изменения всех ветвей в удаленное хранилище.

*Указание.* По умолчанию передаются только изменения текущей ветви, для передачи изменений всех ветвей следует отметить флажок «Push all branches» диалога push.

*Замечание.* В данном задании отрабатывается навык слияния ветвей,существующих тольков локальном хранилище и вступивших в конфликт с ведома единственного автора.

Постоянно возникают и ситуации, когда одну и ту же ветвь, но в локальном и удаленном хранилище независимо изменяют разные авторы. В этом случае действия push и rebase приведут к конфликтам. Их разрешение выполняется совершенно аналогично.