Руководство по **Git,** Часть 2

1. VCS

Git – система контроля версий проекта (Version Control System).

2. Первоначальная настройка:

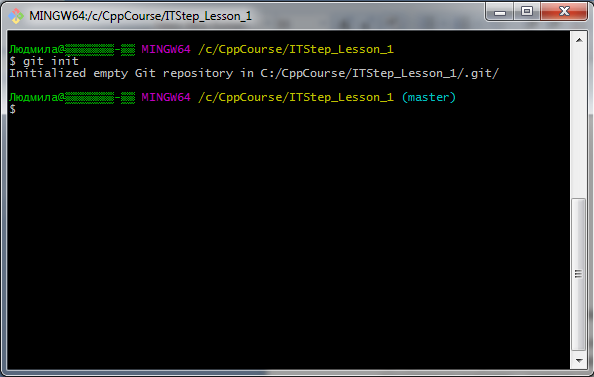
Первоначальная настройка обсуждалась в документе **git\_howto**. Краткий пересказ:

Каждое домашнее задание – это отдельный проект, в котором несколько файлов .cpp с программами – решениями каждой из задач домашнего задания. Каждому такому проекту – домашнему заданию – должен соответствовать один и только один *удаленный git-репозиторий* на сайте <http://github.com/>

Каждый проект – домашнее задание – долен располагаться в отдельной папке на вашем компьютере, например в папке **C:\CppCourse\ITStep\_Lesson\_1\**

В каждой такой директории – **один раз для всего проекта** – нужно инициализировать *локальный git-репозиторий*:

**git init**



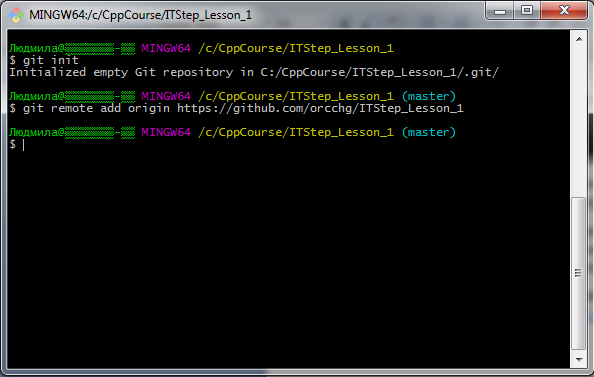
Затем в этой же директории – для этого проекта, и тоже только **один раз** – нужно настроить точку доступа к *удаленному git-репозиторию*, который находится на сайте <http://github.com/>

Прежде всего нужно зарегистрировать аккаунт на сайте <http://github.com/> и создать там удаленный git-репозиторий – по одному отдельному репозиторию на каждый проект – домашнее задание. Как это делать было описано в предыдущем документе **git\_howto**.

Предположим, у нас создан такой удаленный git-репозиторий по адресу: <https://github.com/orcchg/ITStep_Lesson_1>

Связать удаленный git-репозиторий на сайте <http://github.com/> с вашим локальным git-репозиторием следует командой (выполнить **один раз** для проекта):

**git remote add origin** <https://github.com/orcchg/ITStep_Lesson_1>



3. Постоянная работа над проектом:

3.1. Добавление нового файла – решения задачи домашнего задания в проект.

Пусть мы написали решение одной из задач этого домашнего задания в файле **hw\_1\_1.cpp**.

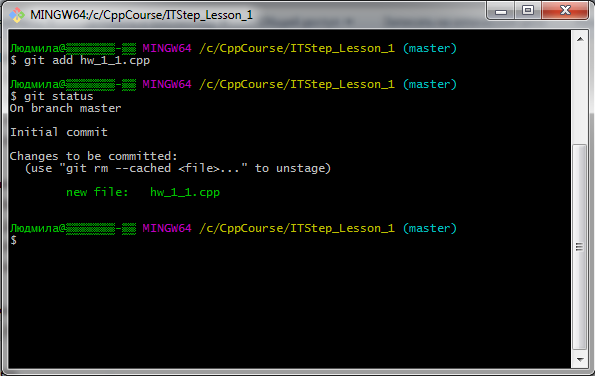
Этот файл необходимо добавить *под версионный контроль*, чтобы

а) отправить его в удаленный git-репозиторий

б) сохранять историю изменений, проделанных над этим файлом

Это делается командой:

g**it add hw\_1\_1.cpp**

****

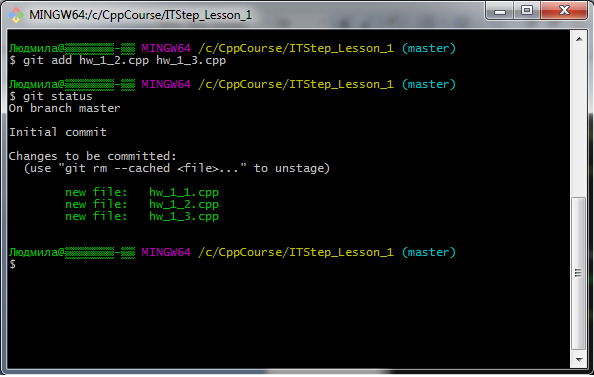
С помощью команды:

git status

можно проверить текущее состояние локального git-репозитория. Сейчас статус показывает, что под версионный контроль только что был добавлен новый файл (new file) **hw\_1\_1.cpp**.

Давайте теперь создадим еще два файла с решениями других задач и добавим их под версионный контроль:

**git add hw\_1\_2.cpp hw\_1\_3.cpp**

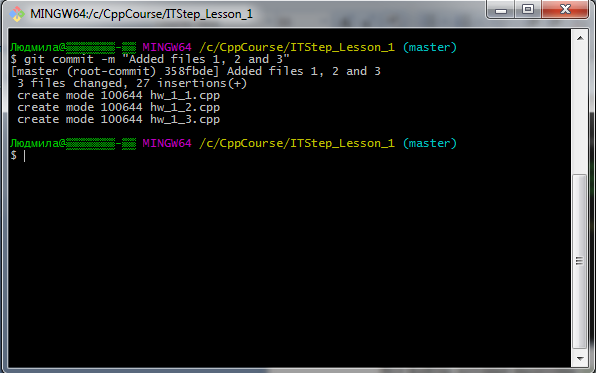


Все файлы, которые выделены зеленым цветом, называются *подготовленными* (staged) *к коммиту*. На момент выполнения команды **git add** содержимое этих файлов стало зафиксированным – в таком виде они войдут в историю проекта.

Коммит (commit) – это точка во времени, описывающая состояние проекта (т.е. содержимое всех его файлов) в этот момент времени. Все файлы, которые выделены зеленым цветом, войдут в новый коммит, и именно в том состоянии, в котором они находились в момент выполнения **git add**.

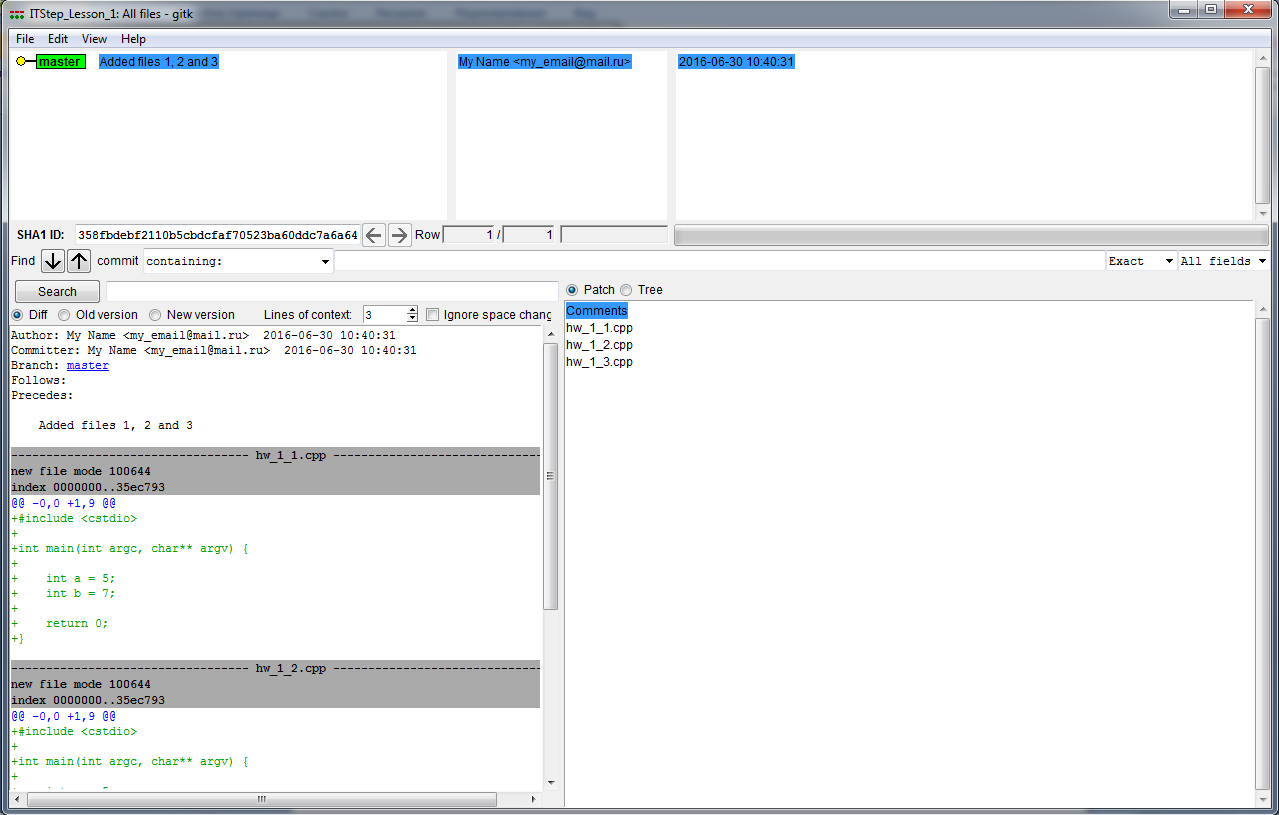
Теперь нужно сделать *git-коммит*:

**git commit –m “Added files 1, 2 and 3**”



Посмотреть наглядную историю проекта можно с помощью команды:

**gitk**



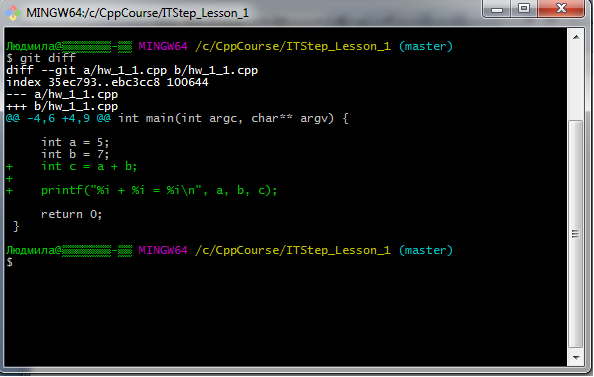
Здесь в левом верхнем углу можно увидеть историю проекта в виде цепочки сделанных коммитов. Каждый коммит – состояние проекта в конкретный момент времени

Чуть ниже – информация о коммите – можно увидеть автора коммита и его email, дату и время коммита (на момент выполнения команды **git commit**), и сообщение коммита. Еще ниже – изменения над файлами. Зеленым выделены новые или измененные строки, а красным удаленные. Так как в этом коммите все файлы новые, и вообще сам коммит – самый первый – то файлы не с чем сравнивать, т.к сейчас они в своей *первоначальной версии* – поэтому все строки выделены зеленым (все новые). Справа внизу – список файлов, добавленных или измененных в рамках данного коммита (все файлы, переданные команде **git add**).

3.2. Внесение изменения в файл:

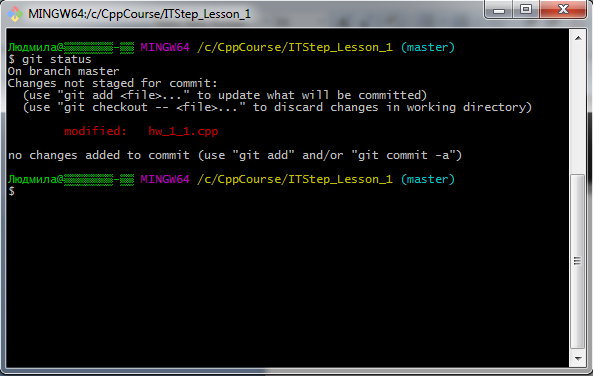
Добавим несколько новых строк в файл **hw\_1\_1.cpp**, тем самым изменив его. Посмотреть, какие же мы сделали изменения этого файла по сравнению с его *предыдущей зафиксированной версией* (в рамках первого коммита) можно командой:

**git diff hw\_1\_1.cpp**

****

Команда «статус» тоже покажет, что в проекте есть изменения:

**git status**



Красным выделены файлы, для которых верно все вместе нижеперечисленное:

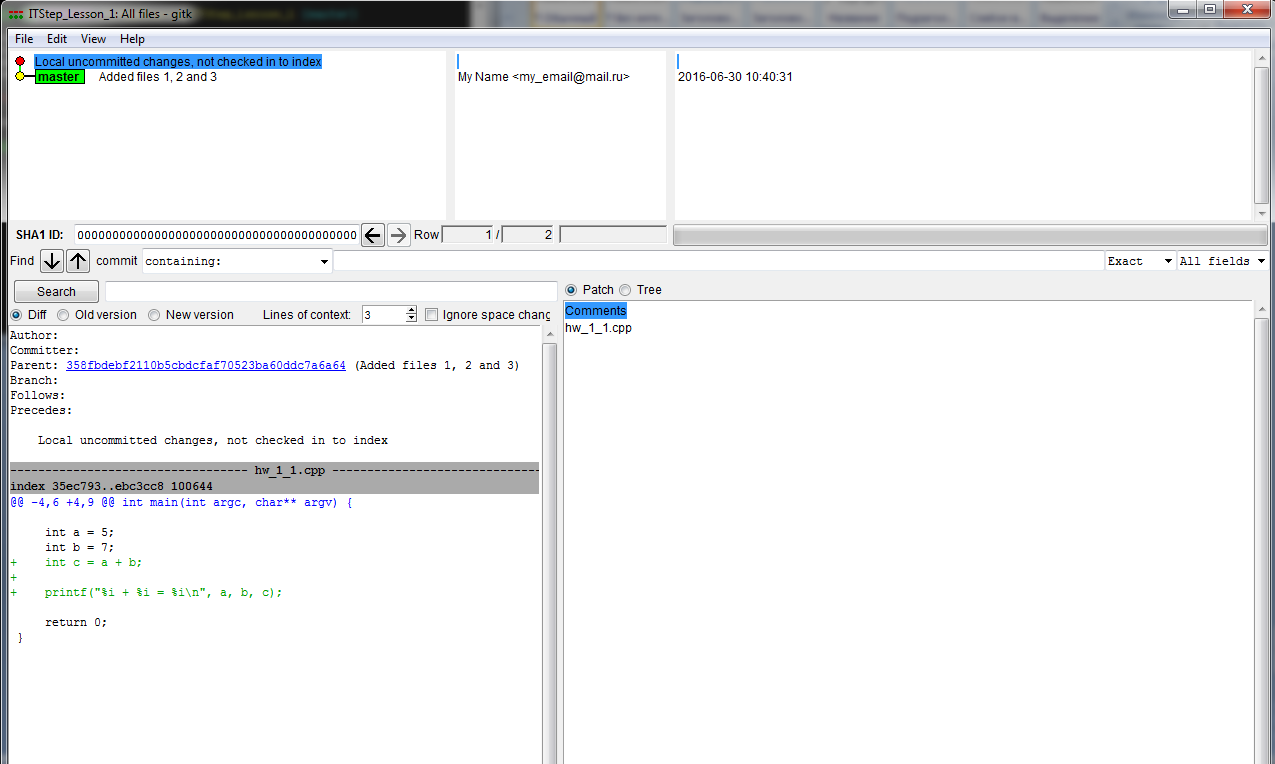
а) файлы находятся под версионным контролем (т.е. над ними хотя бы 1 раз ранее была

выполнена команда **git add**).

б) файлы изменились со времени последнего коммита, в который они входили

Команда **git diff** показывает, что именно было изменено в файле по сравнению с предыдущей версией этого файла. В **gitk** также можно увидеть изменения:

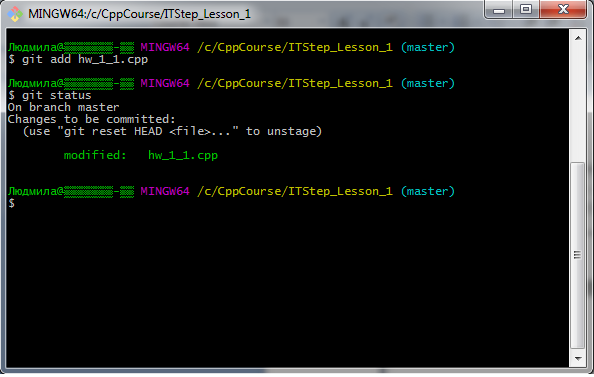
**gitk**



Такие файлы, которые в «статусе» помечены красным, находятся в так называемой *unstaged area*.

Теперь, чтобы зафиксировать эти изменения над файлами в истории проекта – то есть сделать очередной коммит – эти файлы сначала нужно добавить в *staged area* – подготовить к коммиту:

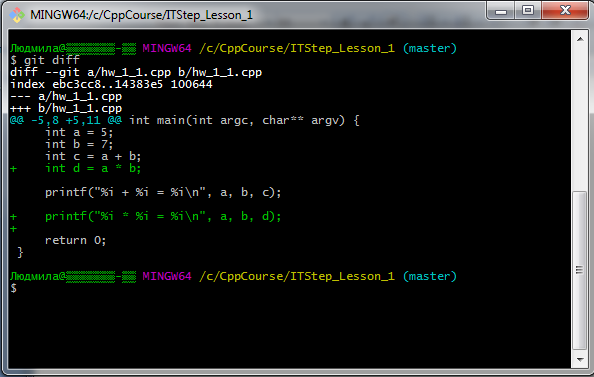
**git add hw\_1\_1.cpp**

****

Как показывает «статус», изменения в файле **hw\_1\_1.cpp** подготовлены к очередному коммиту, файл находится в *staged area* (changes to be committed), а *unstaged area* теперь пуста.

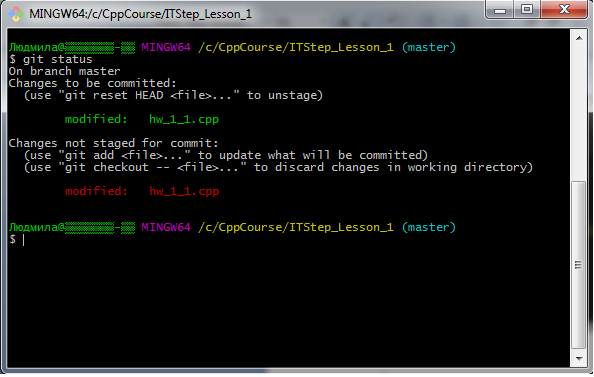
Не будем пока делать новый коммит командой **git commit**, а сделаем еще одно изменение над тем же файлом **hw\_1\_1.cpp**.

**git diff hw\_1\_1.cpp**



Здесь мы видим только изменения файла между его *staged* и *unstaged* версиями. Действительно, посмотрим «статус»:

**git status**

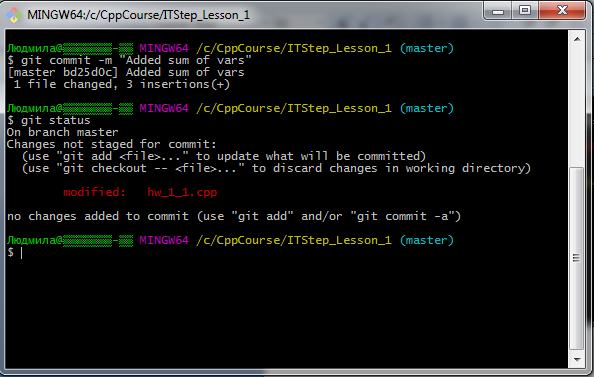


Файл **hw\_1\_1.cpp** находится одновременно в двух областях – в *staged* (зеленым) – и в *unstaged* (красным). Состояние файла в *staged* области – когда мы сделали первое изменение над файлом – подготовлено для следующего коммита, а вот вторые изменения над тем же файлом – не войдут в коммит.

**В коммит входят всегда только файлы в *staged* области!**

Т.е. первые изменения войдут в коммит, а вторые изменения в этот коммит не войдут. Сделаем коммит:

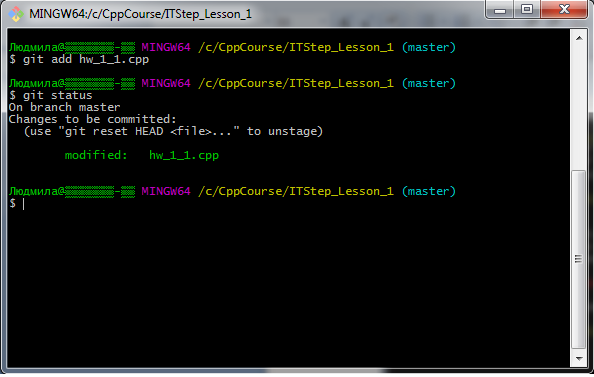
**git commit –m “Added sum of vars”**



Сделан второй коммит в истории проекта. В него вошли первые изменения над файлом **hw\_1\_1.cpp.**

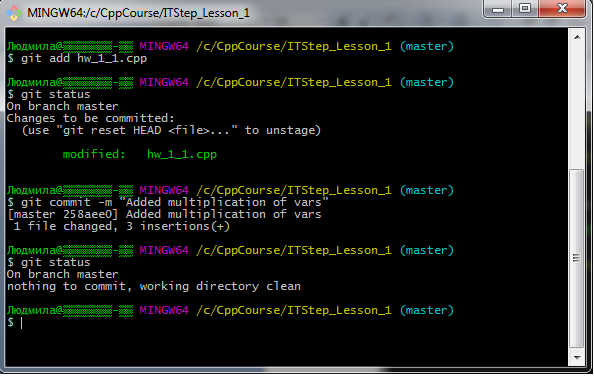
Новые изменения над этим файлом пока не вошли в коммит. Подготовим их для третьего коммита:

**git add hw\_1\_1.cpp**

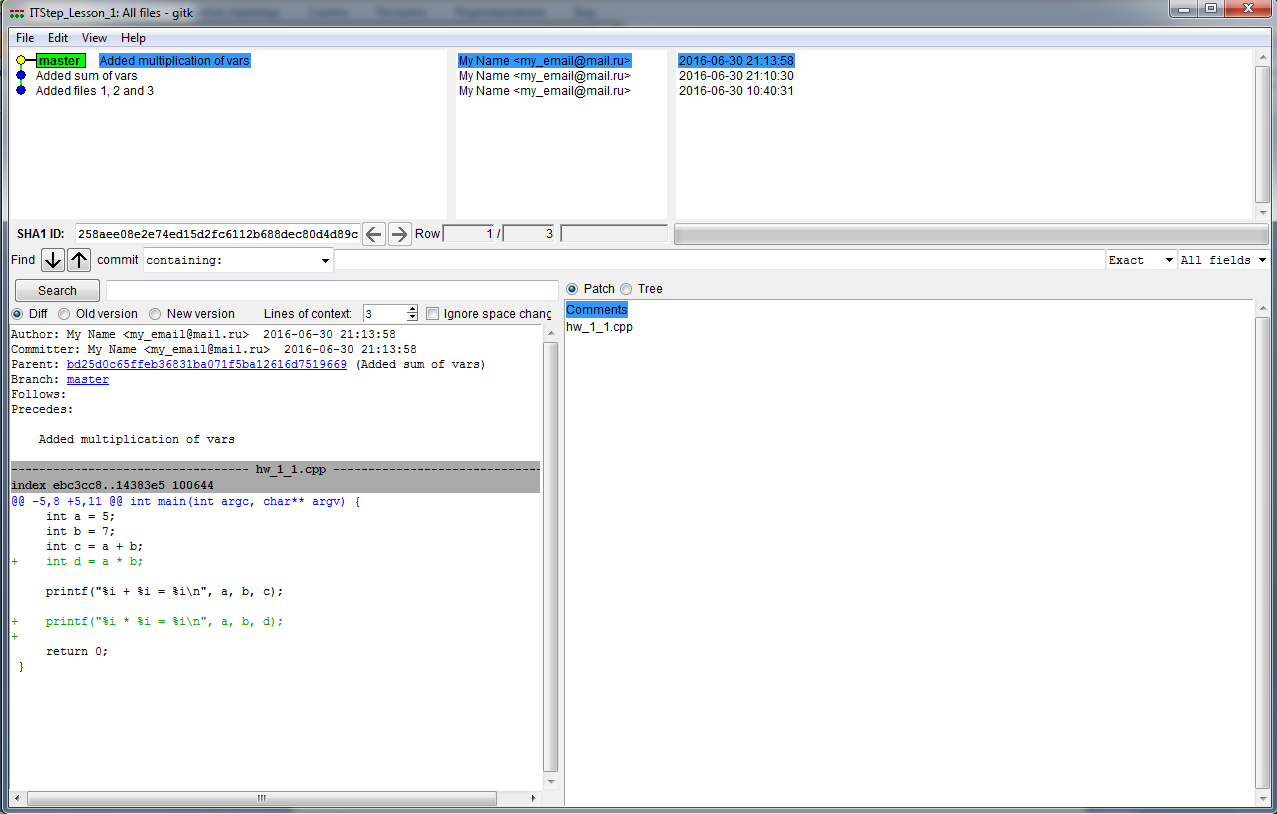


Теперь сделаем третий коммит с последними изменениями:

**git commit –m “Added multiplication of vars”**



В истории проекта теперь можно увидеть три коммита – исходное состояние проекта и всех его файлов, а также последовательные изменения над файлом **hw\_1\_1.cpp**, сделанные в различные моменты времени:



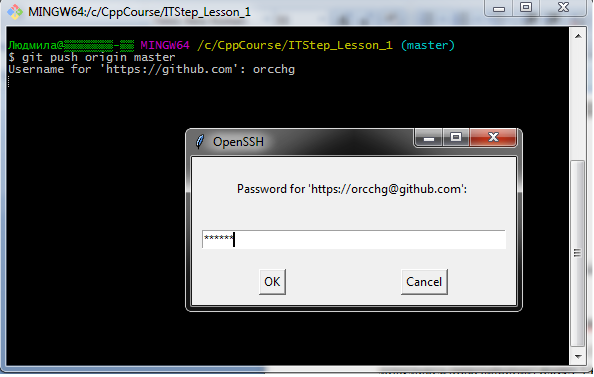
4. Отправка всех локальных коммитов в удаленный репозиторий.

Мы только что сделали *локально* 3 коммита – проект на нашем компьютере (т.е. в нашем *локальном репозитории*) имеет 3 версии с указанием автора, времени-даты,комментария и самих изменений файлов проекта.

Теперь этими изменениями необходимо поделиться с теми, у кого есть доступ к удаленному репозиторию проекта. Все локальные коммиты нужно *запушить* (push), чтобы состояния проекта (т.е. цепочки коммитов) локально и удаленно стали идентичными.

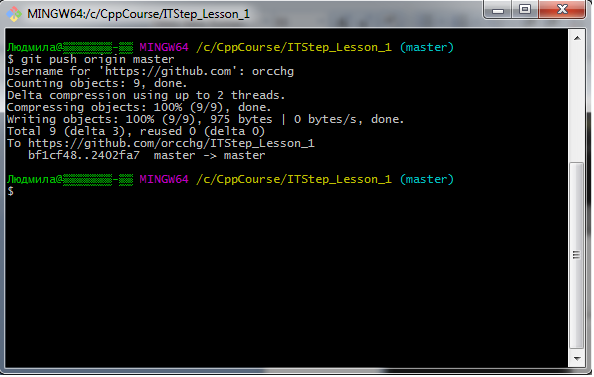
Для этого выполняется команда:

**git push origin master**



Нужно ввести логин и пароль от аккаунта на сайте <http://github.com/>

Успех:



История проекта видна на сайте <http://github.com/> в репозитории (<https://github.com/orcchg/ITStep_Lesson_1/commits/master>):

