УДК 004.021

Управление версиями программного обеспечения на примере GitHub

Козич Полина Александровна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Большинство крупных программных продуктов написаны командой разработчиков. В процессе разработки, команде необходимо обмениваться изменениями, которые делает каждый член команды, при этом, постоянно сохраняя актуальность кодовой базы. Также в процессе разработки есть необходимость хранить и использовать несколько версий программного продукта, например, стабильная версия для релиза, версия для разработки, где изменения продукта только дорабатываются, тестовая версия, версия с каким-то отдельным крупным изменением. Для этих целей существуют системы контроля версий. В данной статье будет подробно рассмотрено, как работает система контроля версий на примере Git, для чего она нужны, создадим и поработает с удаленным репозиторием, предоставленным веб-сервисом GitHub.
Ключевые слова: Система контроля версий, Git, GitHub, разработка программных продуктов, репозиторий.

Version control of the software on the example of GitHub

Kozich Polina Alexandrovna
Sholom-Aleichem Priamursky State University
Student

Abstract
The majority of large software products are written by a design team. In development process, a command it is necessary to exchange changes which are done by each member of the team, at the same time, permanently saving relevance of code base. Also in development process there is a need to store and use several versions of a software product, for example, stable release for release, the version for development where changes of a product are only improved, the test version, the version with some separate large change. For these purposes there are control systems of versions. In this article explicitly it will be considered as the control system of versions on the example of Git works for what it are necessary, we will create and will work with the remote repository provided by GitHub web service.
Keyword: Control system of versions, Git, GitHub, development of software products, repository.

Рассмотрим гипотетическую ситуацию. Есть задача реализовать программный проект. Для реализация выделен один разработчик. Тут в принципе никаких проблем нет. Разработчик просто берет и пишет программный код. Теперь представим, что разработчик сделал первую версию интерфейса и взялся реализовывать вторую. Заказчик требует показать интерфейс, но в данный момент, он в промежуточном состоянии, когда внесена только часть изменений и выглядит все это в целом ужасно. Данную проблему решила бы возможность, сохранить кодовую базу в том состоянии, когда первая версия интерфейса была готова, для того, чтобы при запросе, показать ее заказчику.

Вторая ситуация, все тот же проект, но теперь в команде два разработчика. Ситуация выше может повториться и при данных условиях, но допустим, что команда фиксирует состояние проекта в ключевых моментах. Проблема в том, что два разработчика работают над проектом параллельно и как вследствие у каждого из них на руках своя версия программного проекта. Каким образом они могут решить данную проблему. Возьмем примитивное решение, где один из разработчиков отправляет файлы, в которые внес изменения другому. Тот другой просматривает изменения, вносит их в свою кодовую базу и уже проект с изменениями первого и второго отправляет обратно. Задача выполнена, у двух разработчиков одна и та же версия программного проекта. Основная проблема – трата драгоценного времени разработчика. В коммерческой разработке, самое дорогое – это время разработчика. Именно на оплату его труда тратятся большие деньги, поэтому это не только утомительно для разработчика, но и дорого для компании, в которой ведется разработка. Вторая проблема, способ становиться еще более дорогим, с учетом роста команды разработчика.

Данные ситуации, всего лишь пример. Проблема актуальна не только в сфере разработки, но и в других, где применяется электронный документооборот и работы нескольких человек с одним документом. Для решения данной проблемы была придумана система контроля версий или как еще ее называют система управления версиями. 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.

Цель исследования в данной статье: описать в контексте работы разработчиков программного обеспечения использование системы контроля версий.

Т.Ф. Гусева и О.Р. Комиссарова применили системы контроля версий в образовательном процессе [1]. Системы контроля версий в электронном документообороте рассмотрел А.А. Штанюк [2]. Также он подробно описал системы управления версиями при изучении программирования [3]. Д.В. Шагбазян поделился опытом использования систем контроля версий в учебном процессе [4]. Е.Н. Давыдова и А.А. Головлев рассмотрели системы управления версиями в командной разработке программного обеспечения [5]. Системы контроля версий программного обеспечения для организации

взаимодействия команды при разработке программного проекта показал Н.Р. Нурлыгаянов [6]. В.В. Яворский, А.О. Чванова и Н.В. Байдикова изучили возможности использования сервиса GitHub в учебном процессе [7]. Зарубежные ученые также рассмотрели данный сервис [8].

Существует много систем контроля версий: SVN, Git, Mercurial, Team Foundation Server, CVS, Bazaar – это самые известные.

Все они содержат, как минимум, базовый функционал состоящий из:

1. Клонирование проекта
2. Модификация проекта
3. Ветвление
4. Слияние версий

Прежде чем приступить к работе, необходимо разобраться в таких понятиях как локальный репозиторий и удаленный репозиторий. Репозиторий – это хранилище где хранятся и поддерживаются какие-либо данные, в данном случае исходный код программного продукта. Удаленный репозиторий – это репозиторий, к которому имеет доступ вся команда разработки, именно он содержит версию с изменениями всей команды. Локальный репозиторий – это репозиторий, расположенный у каждого разработчика. Именно с ним он работает на своем компьютере. Актуальность локальных репозиториев поддерживается путем постоянного обновления из удаленного репозитория. Таким образом вся команда работает примерно с одним и тем же состоянием проекта.

Для примера будет использована систем контроля версий Git и веб-сервис, хостинг It-проектов использующийся в качестве удаленного репозитория GitHub.

На данный момент GitHub доступен бесплатно для проектов с открытым исходным кодом. Приватные репозитории является платной услугой. Недавно (4 июня 2018 года) GitHub был куплен компанией Microsoft, что возможно приведет к изменению политики сервиса.

Для того, чтобы приступить к работе, необходимо зарегистрироваться на GitHub (https://github.com). Окно регистрации представлено на рисунке 1.

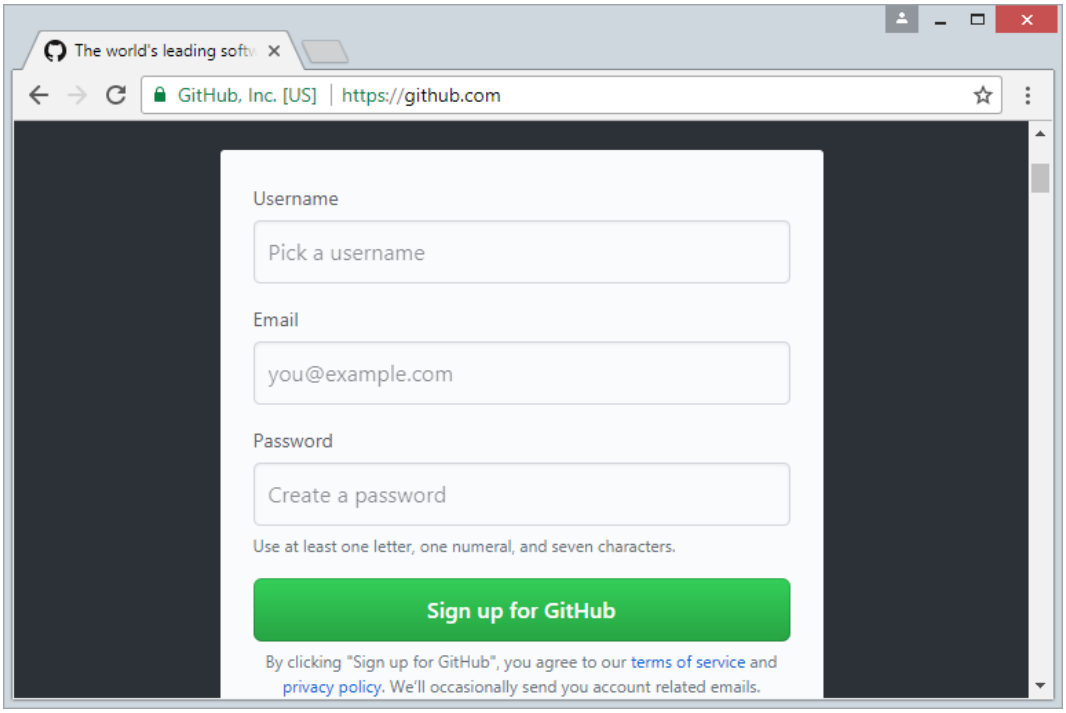

Рисунок 1 – Окно регистрации сервиса GitHub

Далее необходимо создать новый репозиторий. Данный репозиторий и есть то, что выше определен, как удаленный репозиторий. Кнопка для создания репозитория на GitHub показана на рисунке 2 в правом верхнем углу.

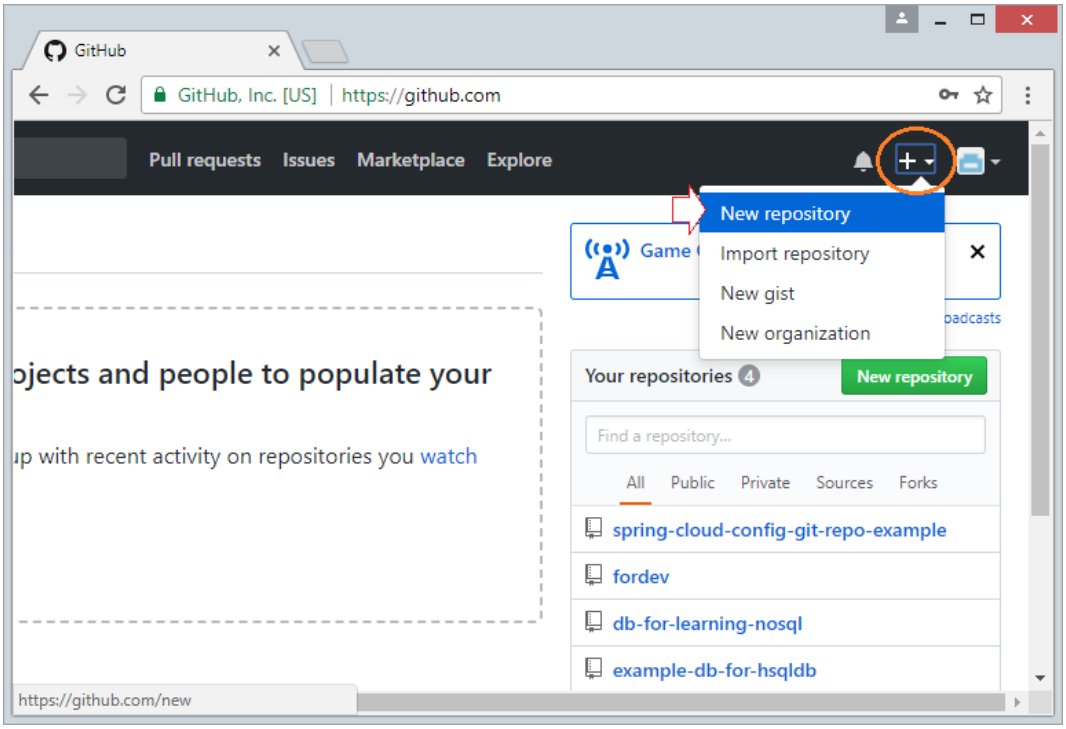

Рисунок 2 – Кнопка создания нового репозитория

Далее необходимо ввести имя репозитория и тип. В данном случае необходимо выбрать публичный (public). Пример изображен на рисунке 3.

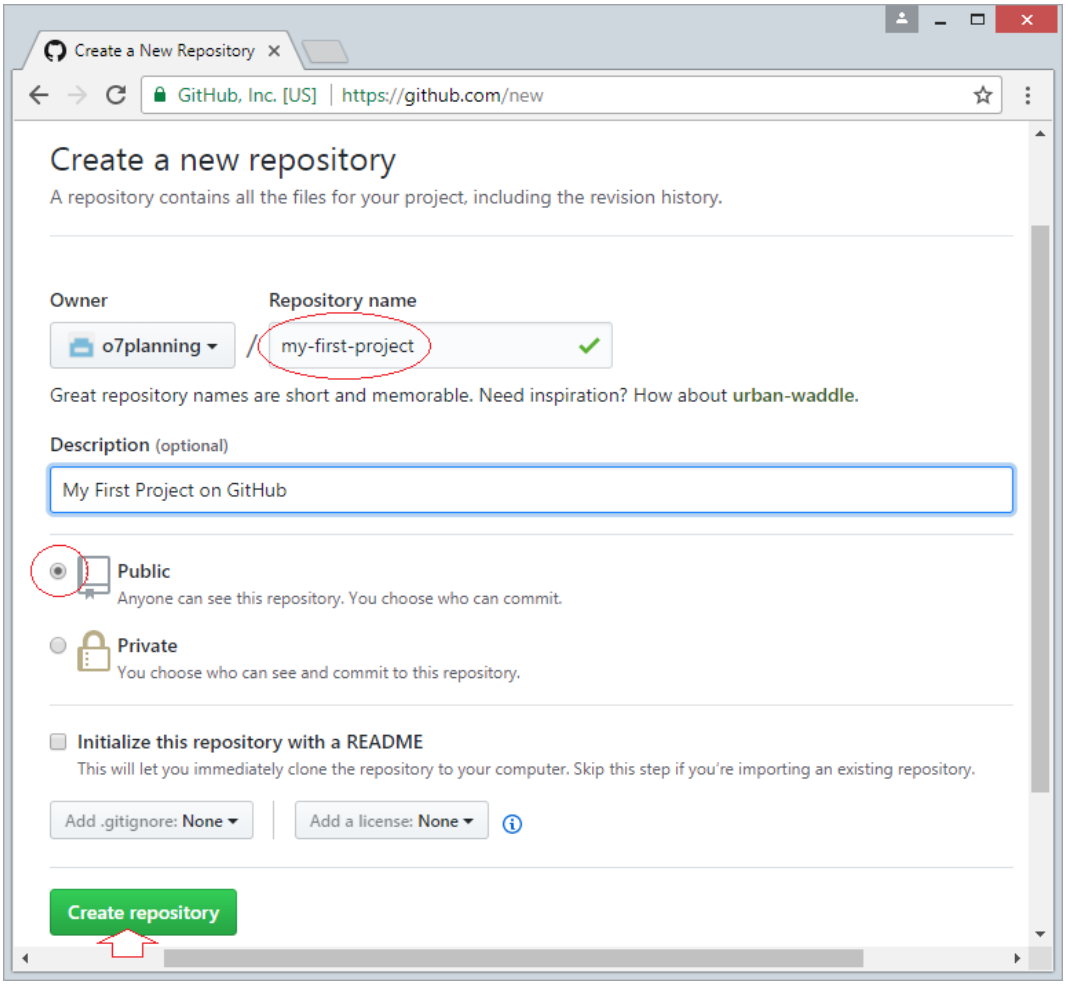

Рисунок 3 – Окно создания нового репозитория

Далее для работы нам необходимо скачать специальный клиент GitHub Desktop (https://desktop.github.com/). После установки нам необходимо войти под той же учетной записью, которую зарегистрировали на GitHub. Затем ввести имя и почту. Необходимо это для того, чтоб данная информация отображалась при дальнейших изменениях проекта.

Теперь необходимо склонировать удаленный репозиторий, таким образом, будет создан локальный, являющийся точной копией удаленного. Для этого, предварительно необходимо создать папку, где будет храниться локальный репозиторий, затем необходимо нажать File -> Clone repository. Откроется окно, изображенное на рисунке 4. Необходимо ввести путь для сохранения репозитория в виде ранее созданного каталога и нажать кнопку Clone.

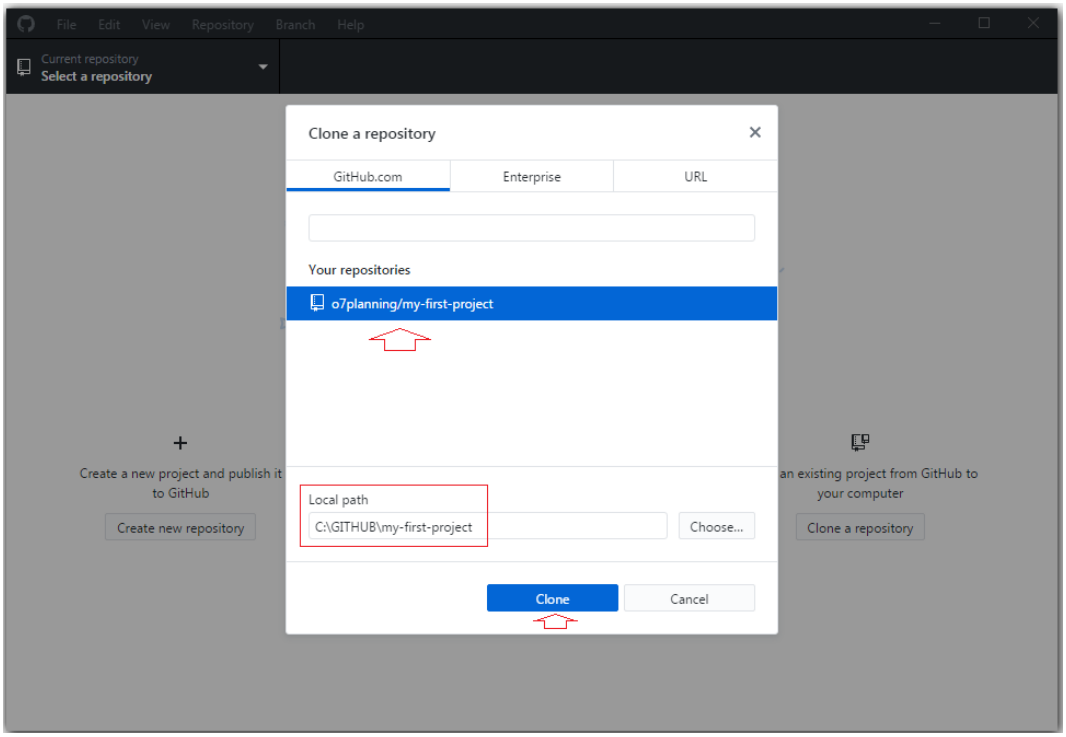

Рисунок 4 – Окно клонирования репозитория

Теперь на компьютере имеется локальный репозиторий. Изменим его добавив в каталог два файла, пример на рисунке 5.

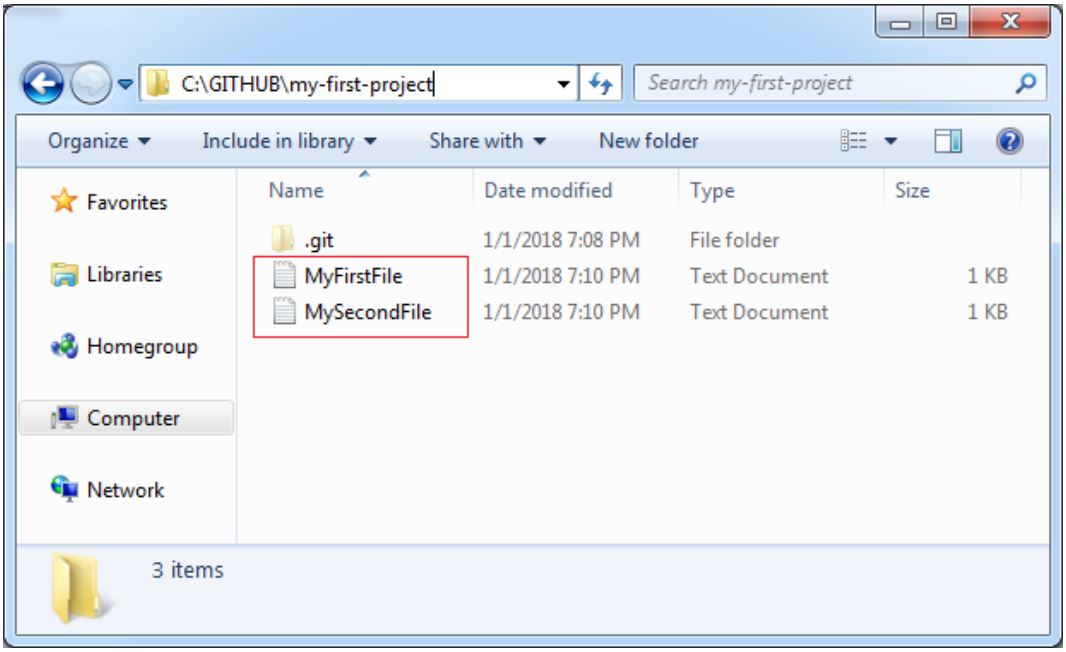

Рисунок 5 – Структура каталога после добавления файлов

Данные изменения также отобразятся в GitHub Desktop (рисунок 6).

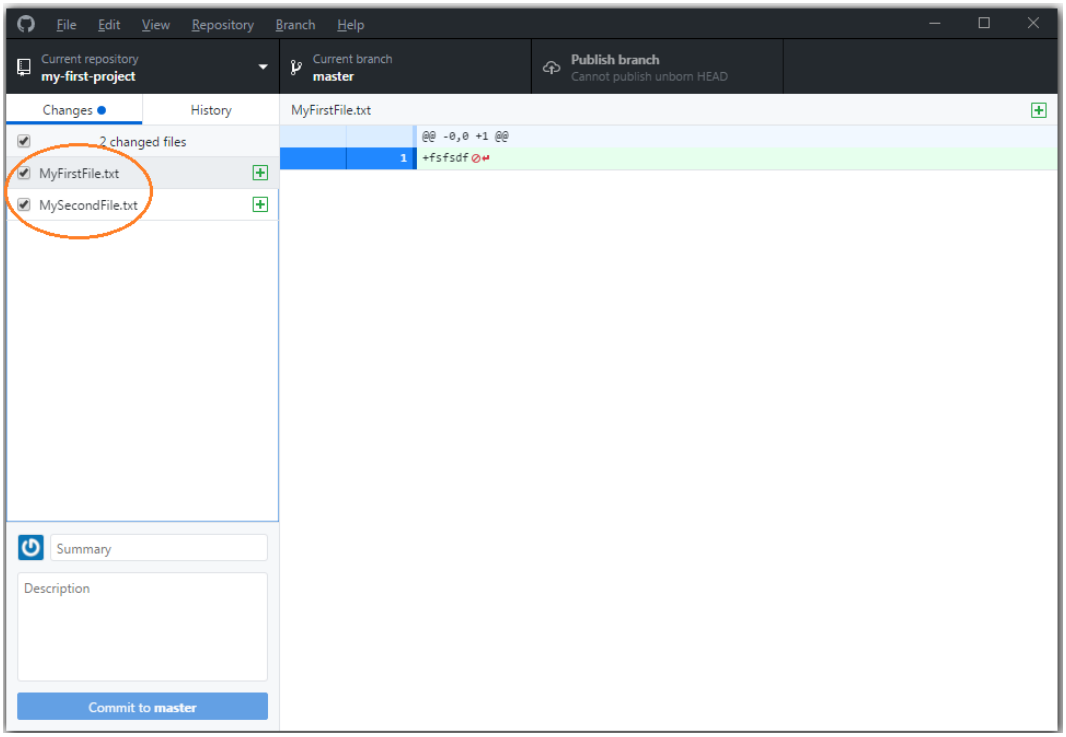

Рисунок 6 – Отображение добавленных файлов в GitHub Desktop

Необходимо зафиксировать данные изменения. Данные фиксации называют коммит(commit). Для этого необходимо дать название коммиту и хорошим тоном считается описание, чтоб команда разработчиков понимала, что было сделано в том или ином изменении. Но в данном случае, это опустим. На рисунке 7 изображено как сделать коммит.

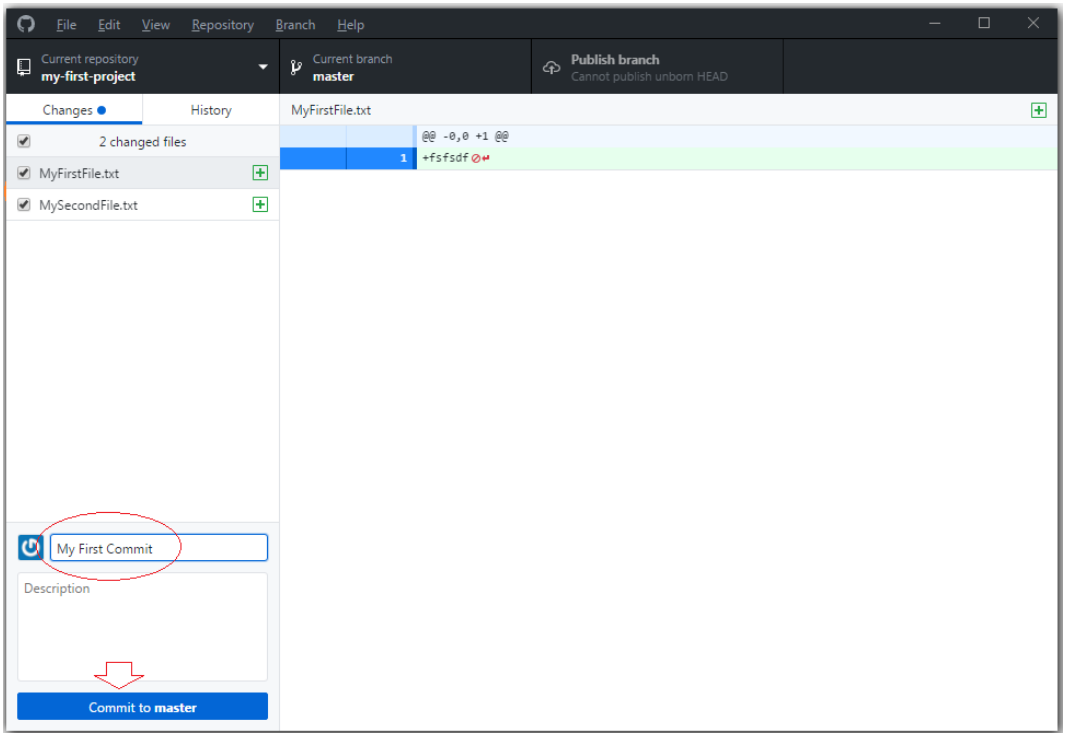

Рисунок 7 – Фиксация изменений. Отправка коммита

На рисунке 7 мы видим на кнопке надпись Commit to master. Master в данном случае, название ветви, в которую фиксируются изменения. В данный момент она одна. Но в реальных проектах чаще всего используется более одной ветви. Для чего необходимы ветви, будет рассмотрено позже.

Во вкладке History, справа, можно просмотреть историю коммитов. Чем больше их будет, тем длиннее будет список. Учтем, что пока изменения зафиксированы только в локальном репозитории, а значит, что команда не увидит данных изменений. Проверить это можно зайдя на GitHub, где создавали репозиторий и убедиться, что добавленных файлов там нет. Для того, чтобы отправить изменения в удаленный репозиторий, необходимо нажать кнопку publish branch (рисунок 8).

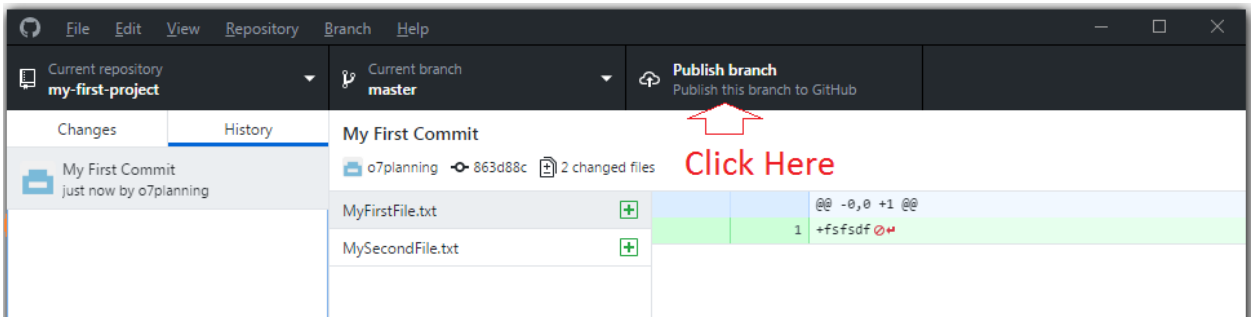

Рисунок 8 – Публикация изменений в удаленный репозиторий

Данная операция так же известна как пуш (push). Теперь если зайти на GitHub, можно увидеть добавленные файлы (рисунок 9).

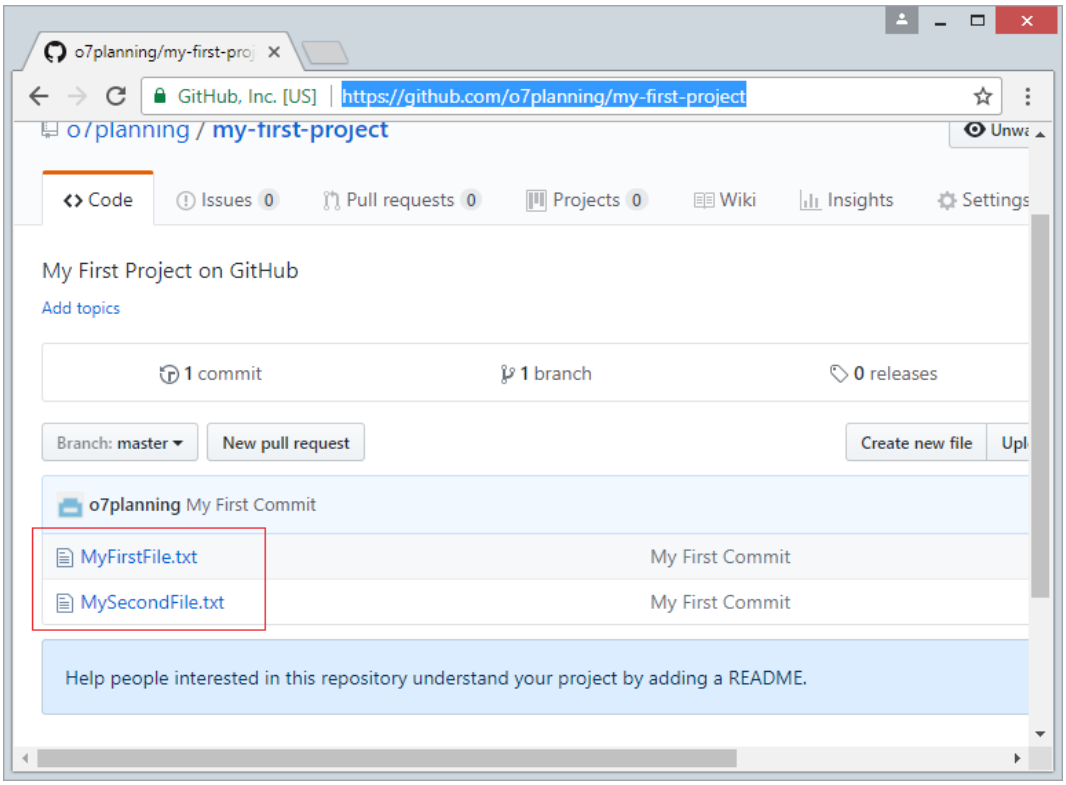

Рисунок 9 – Удаленный репозиторий GitHub после отправки изменений

Теперь данные изменения доступны всем желающим, кто хочет работать с данным репозиторием.

Вернемся к ветвям и их предназначенью. Рассмотрим ситуацию, где для программы проектируется интерфейс. Необходимо сделать три прототипа и показать их заказчику. Для каждого из них будет создана своя ветвь. Таким образом, у нас будет три версии продукта, с разными интерфейсами. Отметим, что изменения, не утвержденные и не факт, что заказчику понравиться хотя бы один прототип. Таким образом, не имеет смысла засорять кодовую базу этими изменениями, пока нет одобрения. Следовательно, имеется четыре версии продукта. Версия, где фиксируются только стабильные и оговоренные изменения, так сказать основная ветвь, и три ветви с прототипами интерфейсов. Допустим, заказчик одобрил один из них. Теперь необходимо влить изменения из ветви с этим интерфейсом в основную ветвь. Данный функционал входит в системы контроля версий, в том числе и в Git. И с помощью GitHub Destop это также возможно сделать. Данная операция называется мёрдж (merge) ветвей.

Для того чтобы создать ветвь в локальном репозитории с помощью GitHub Desktop необходимо нажать на вкладку Current branc вверху, затем кнопку new (рисунок 10).

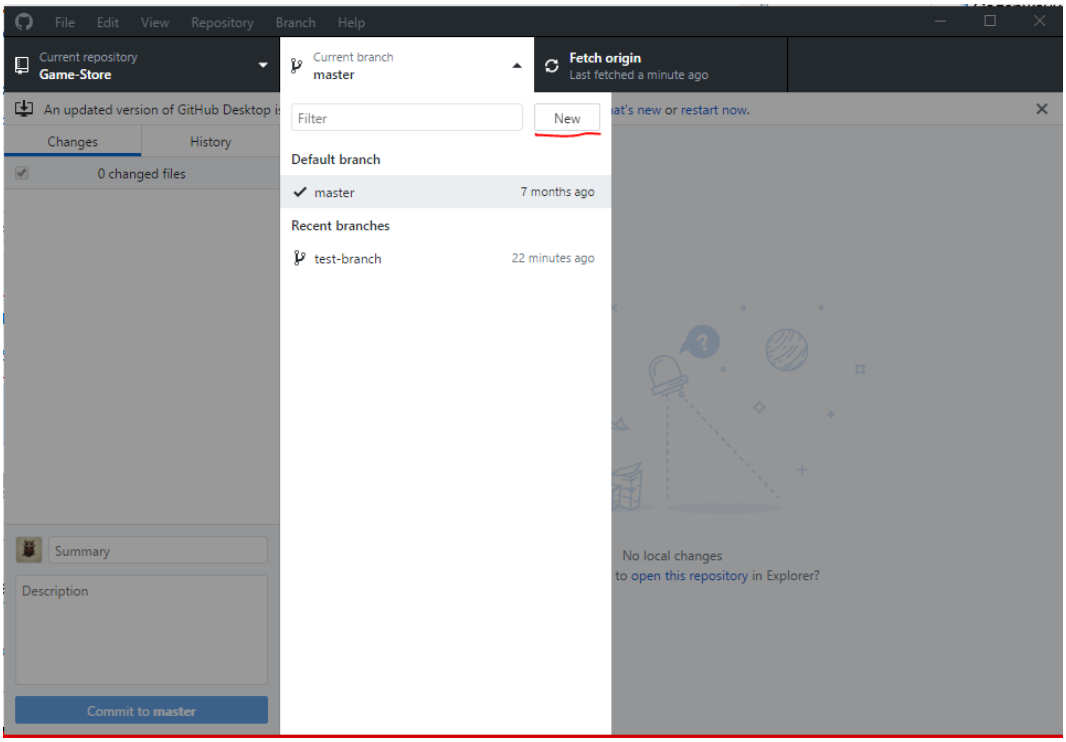

Рисунок 10 – Создание новой ветви

Далее появиться окно, где необходимо ввести название ветви и нажать на кнопку Create branch. Отметим, что создание ветви – это такое же изменение, как и добавление/изменение файлов, которое так же требует фиксации и отправки в удаленный репозиторий.

Для того чтобы влить одну ветвь в другую, необходимо вверху во вкладке current brunch выбрать ветвь, в которую требуется влить изменения. Затем выбрать вкладку Branch -> Merge into current branch (рисунок 11).

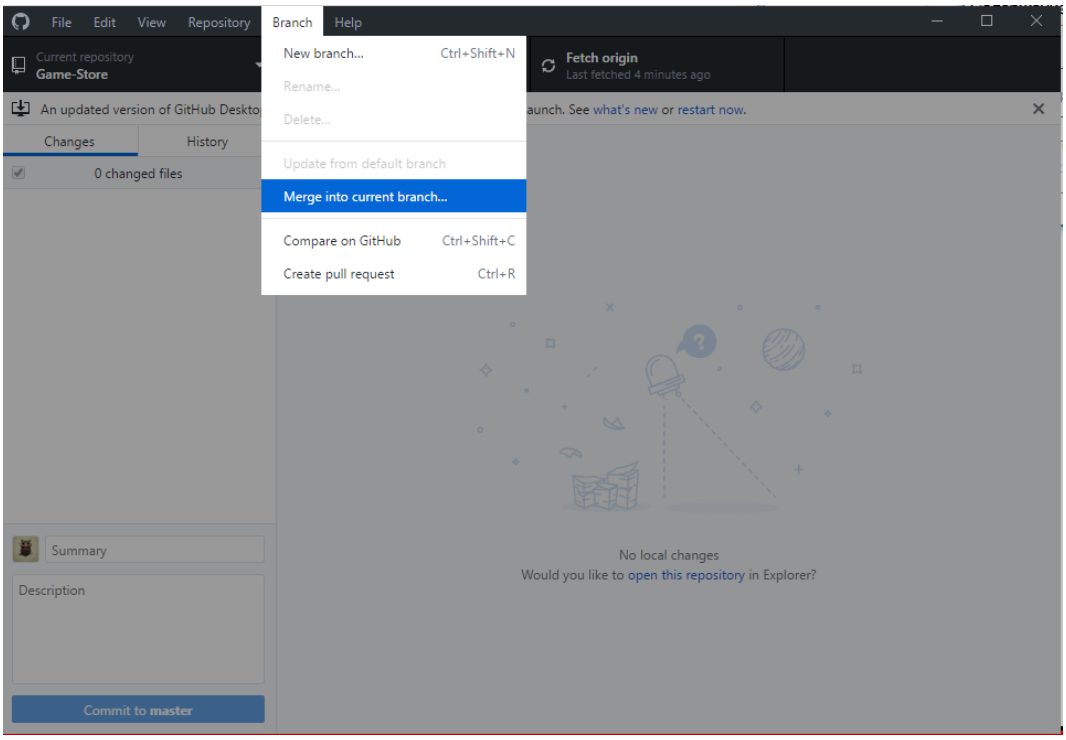

Рисунок 11 – Команда для слияния ветвей

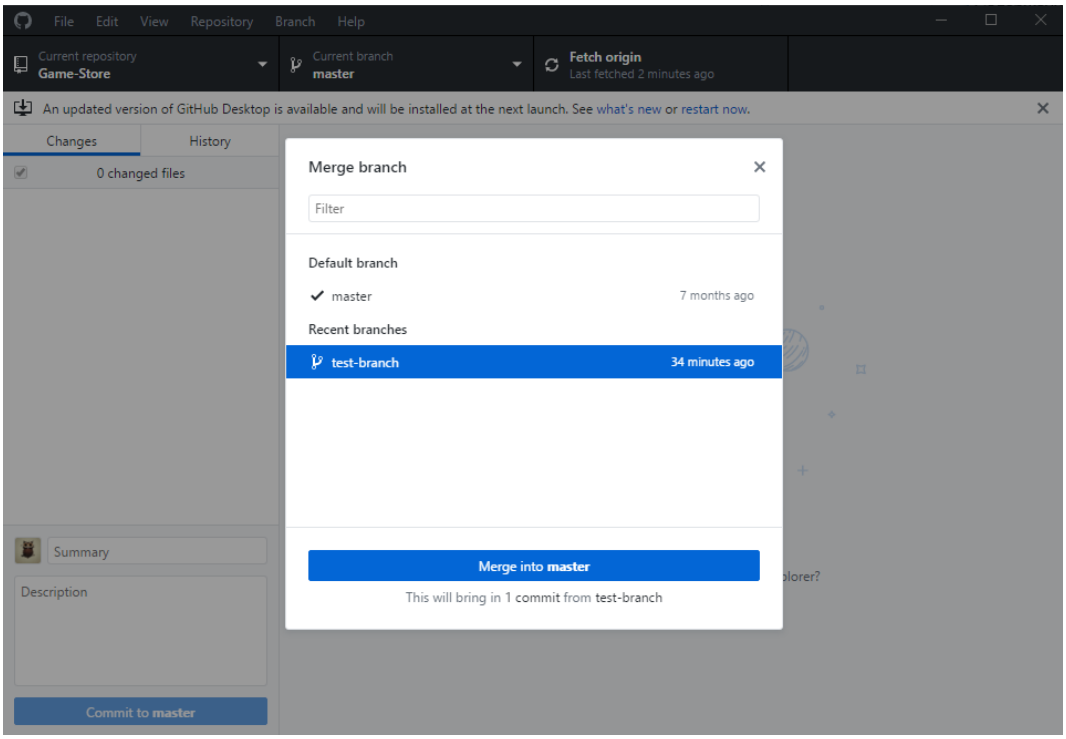

Рисунок 12 – Выбор ветви для слияния с текущей

В данной статье были продемонстрированы основные операции в системе контроля версий Git с использованием GitHub и GitHub Desktop. Стоить отметить, что это далеко не весь функционал системы контроля

версий и имеет смысл углубиться в более подробное изучение данной темы. Так же стоит отметить, что несмотря на красивый и понятный интерфейс GutHub Desktop рекомендуется также изучить работу с консольными командами, так как освоение этого инструмента даст возможность для более быстрого исполнения команд и избавит от зависимости от визуального интерфейса того или иного инструмента.

Библиографический список

1. Гусева Т.Ф., Комиссарова О.Р. Применение системы контроля версий в образовательном процессе // Современная техника и технологии. 2016. № 11-2 (63). С. 128-132.
2. Штанюк А.А. Системы контроля версий в электронном документообороте // Международное научное издание Современные фундаментальные и прикладные исследования. 2013. № 4 (11). С. 60-62.
3. Штанюк А.А. Системы управления версиями при изучении программирования // Международное научное издание Современные фундаментальные и прикладные исследования. 2015. № 3 (18). С. 19-21.
4. Шагбазян Д.В., Штанюк А.А. Опыт использования систем контроля версий в учебном процессе // Постулат. 2017. № 12 (26). С. 29.
5. Давыдова Е.Н., Головлев А.А. Системы управления версиями в командной разработке программного обеспечения // В сборнике: Вузовская наука - региону Материалы XV Всероссийской научной конференции с международным участием. 2017. С. 72-74.
6. Нурлыгаянов Н.Р. Системы контроля версий программного обеспечения для организации взаимодействия команды при разработке программного проекта // Современные информационные технологии. 2016. № 23. С. 47-51.
7. Яворский В.В., Чванова А.О., Байдикова Н.В. Возможности использования сервиса GitHub в учебном процессе // В сборнике: Современное образование: повышение профессиональной компетентности преподавателей вуза - гарантия обеспечения качества образования Материалы международной научно-методической конференции. 2018. С. 159-160.
8. Jiang J. et al. A first look at unfollowing behavior on GitHub // Information and Software Technology. 2019. Т. 105. С. 150-160.