**Практическая работа: Cоздание элементов GitHab**

Основные теоретические сведения

Git - **система контроля версий** (*version control system, VCS*), созданная программистом Линусом Торвальдсом для управления разработкой ядра Linux в 2005 году. Хорошо, а что это всё-таки значит?

Представьте, что вы с коллегами вместе пишете научную статью. У вас на компьютере есть папка, где лежат текстовые документы, картинки, графики и прочие нужные файлы; то же самое есть и у ваших коллег. Когда кто-то из вас изменяет, добавляет или удаляет файлы, остальные этих изменений не видят. Вы пишете друг другу об изменениях, пересылаете обновленные версии файлов, но в процессе работы непременно возникает путаница: какая версия текста - последняя? Куда и когда исчезла пара абзацев? Кто внес те или иные правки? Избежать таких проблем и помогают системы контроля версий. Устроено это так:

* Ваша папка на компьютере - это не просто папка, а локальный репозиторий.
* Она является копией удалённого репозитория, который лежит на веб-хостинге (например, GitHub или BitBucket).
* Eсли вы работаете над проектом с коллегами, то своя локальная копия есть у каждого.
* Kогда вы внесли некоторое количество изменений, вы можете их сохранить, и это действие запишется в журнал; это называется **commit** (коммит).
* После этого можно отправить изменения в удалённый репозиторий; это называется **push** (пуш, пу́шить, запу́шу, пуши́ от души).
* Актуальная версия проекта, учитывающая последние изменения всех участников, будет храниться в удалённом репозитории.
* Если вы увидели, что ваши коллеги запушили в удалённый репозиторий что-то новенькое, то можно “скопировать” это себе на компьютер (в локальный репозиторий); это называется **pull** (пулл).

Чем-то похоже на Dropbox, Google Drive и прочие облачные хранилища, правда? Только в данном случае ваши файлы синхронизируются не автоматически, а по команде, и возможностей управления ими гораздо больше.

Понятно, что для совместной работы над текстом научной статьи вполне хватит и Google Docs, но вот если, например, вы хотите опубликовать результаты исследования в интернете и сделать для этого собственный сайт, то без VCS обойтись сложно. И ещё раз, системы контроля версий хороши тем, что:

* они позволяют работать над проектом в команде;
* вы видите, кем и когда были внесены те или иные изменения;
* их всегда можно откатить назад;
* вы не потеряете проделанную работу, даже если что-то удалите на своем компьютере;
* ваши наработки могут быть полностью открыты для других (а это доступность знаний и ускорение развития технологий, ура!);
* GitHub и GitLab позволяют не только хранить и просматривать файлы проекта, но и публиковать веб-сайты, документацию и т.п.

Существует много систем управления версиями, но мы будем пользоваться самой распространенной - **git**. Также нам нужно как-то отдавать гиту команды, и делать это можно двумя способами: с помощью командной строки и через графический интерфейс (graphical user interface, GUI). Графический интерфейс программы - это все те окошки с кнопочками, которые мы привыкли видеть. Существует много графических интерфейсов для Git, например:

* [TortoiseGit](https://tortoisegit.org/)
* [GitKraken](https://www.gitkraken.com/)
* [GitHub Desktop](https://desktop.github.com/)
* [Fork](https://git-fork.com/)
* [Git GUI](https://git-scm.com/docs/git-gui)
* [Git Extensions](https://gitextensions.github.io/)
* [SourceTree](https://www.sourcetreeapp.com/)

Мы будем пользоваться программой GitHub Desktop.

Итого:

* Git - разновидность системы контроля версий (самая популярная). Его можно скачать и установить, далее использовать через командную строку.
* Можно использовать графический интерфейс для работы с Git. При этом скачивать и устанавливать сам Git отдельно не нужно, он обычно идет в комплекте с графическим интерфейсом (но не во всех GUI).
* Репозиторий - это место где мы храним наш код проекта и всю информацию по файлам, их изменения и т.д. Репозиторий должен где-то хранится, чтобы у всех был доступ к нему и они могли видеть изменения. Его можно хранить и на домашнем компьютере, но не всегда удобно держать компьютер включенным целыми сутками, поэтому используют хостинги для репозиториев. Одними из самых известных являются **[GitHub](https://github.com/)** и **[GitLab](https://gitlab.com/)**.

Ход выполнения работы

Задание 1: Регестрация

Для работы нам нужен аккаунт. Чтобы зарегистрироваться, идём по ссылке <https://github.com/>, выбираем имя пользователя и пароль. Теперь у вас есть собственная страничка: <https://github.com/username,> где username - имя пользователя, которое вы указали.

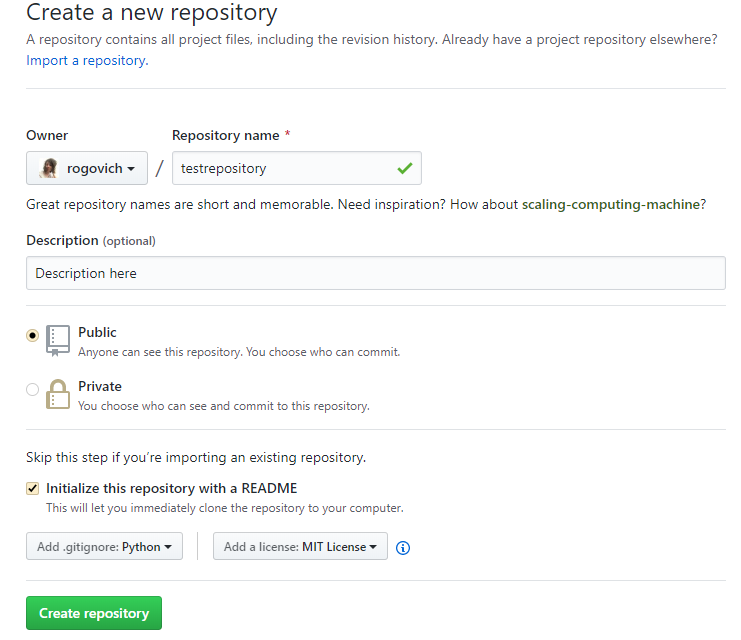
После регистрации Вы попадете на приветственную страницу, где сначала нужно, ничего не меняя, нажать зеленую кнопку **Continue**, а потом **Skip this step**. Далее подтвердите свой аккаунт на указанной ранее почте.

**Задание 2: Создание репозитория**

Создать репозиторий можно двумя способами:

* на сайте;
* через GitHub Desktop.

Создадим через сайт. Чтобы создать репозиторий, нажимаем кнопку **Start a project** и выбираем название. Оно может быть любым, но должно отражать суть того, что лежит внутри, например, “homeworks”. Также в специальном поле можно добавить описание. Для публичных репозиториев хорошей практикой является заполнение всех полей, чтобы другие пользователи (или люди, проходящие по ссылке из резюме) могли сразу понять, о чём конкретно данный репозиторий.



У нас есть выбор между **Public** и **Private**. Разница между ними в том, что публичные репозиторий видно в поиске, в вашем профиле, любой может просмотреть весь код и предложить свои исправления (**pull request**, пулл-реквест, ПР, пи-ар). Приватный репозиторий доступен только определённым пользователям, хозяин репозитория сам выбирает, кто видит репозиторий и кто может делать коммиты. На обычном (бесплатном) аккаунте возможность создавать приватные репозитории обычно ограничена несколькими.

Далее у нас есть возможность инициализировать репозиторий с файлом README (Заполнять обязательно). В нем может быть отображена информация о репозитории, о его использовании, установке файлов и т.д. Описание происходит в формате **Markdown**. Также за этой галочкой скрывается команда **init**, которая превращает пустую папку в Git-проект.

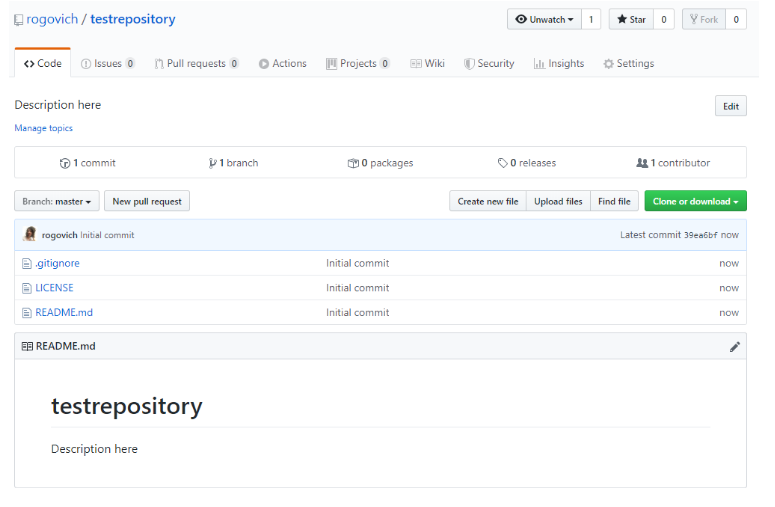
Также стоит упомянуть про файл **.gitignore** и **LICENSE**. Файл **.gitignore** нужен для того, чтобы в репозиторий не попадали разные временные файлы или сборки, например, при сборке проекта в Visual Studio создается множество временных бинарных файлов, которые при каждом изменении исходного кода программы, будут другими, поэтому для репозитория (хранилища *исходного* кода) это по факту *мусор*. Поэтому в этом файле прописано, что определенные папки и файлы не будут учитываться при подготовке коммитов и, следовательно, загрузке в удалённый репозиторий. При создании репозитория можно выбрать уже заранее созданные файлы под язык программирования или среду разработки. Также его можно прописать или дополнить и указать какие файлы включить или убрать из репозитория. Файл **LICENSE** указывает на то, по какой лицензии распространяется код. Про каждую лицензию можно почитать отдельно и в основном они отличаются тем, что можно делать с кодом: продавать, распространять, изменять и т.д. При создании нашего репозитория можно либо выбрать эти файлы, либо оставить их пустыми.

Популярные лицензии (в сторону уменьшения количества ограничений):

* GNU GPL;
* MIT;
* Unlicense;
* WTFPL (do whatever you want public license).

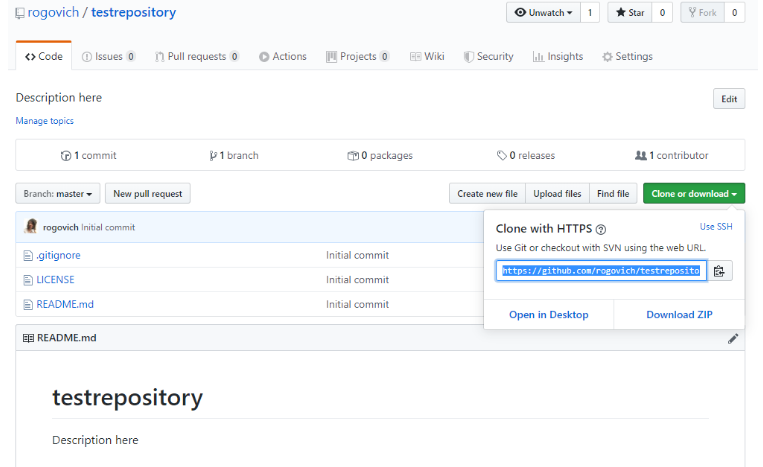
Текст лицензии понадобится скопировать в файл **LICENSE**.

Далее нажимаем на зеленую кнопку **Create repository**. Вы увидите список файлов в своем репозитории (пока это только автоматически сгенерированный файл README с описанием проекта) и содержание README, если он есть, а также файлы **.gitignore** и **LICENSE**, если они создавались. Ссылка на репозиторий будет выглядеть так: <https://github.com/username/your>\_repo\_name.git .



**Клонируем репозиторий**

Теперь нам нужно сделать локальную копию, нашего удалённого репозитория. Мы снова воспользуемся кнопкой **Clone or download**, но теперь используем полную ссылку на репозиторий; эту ссылку нужно скопировать (Если у вас окошко выглядит не так как на картинке, то нажмите в окне на ссылку справа сверху Use HTTPS).



Откройте терминал и перейдите в директорию для копирования репозитория. Например, для перехода на Рабочий стол напечатайте вот это:

cd Desktop

Затем клонируйте туда репозиторий по следующей команде:

git clone <то,\_что\_вы\_только\_что\_скопировали>

Новый GitHub-репозиторий, склонированный на рабочий стол, готов! Данная команда создает точную копию репозитория в вашей системе. Здесь вы сможете с ним работать, редактировать, индексировать изменения, создавать коммиты с изменениями и отправлять их на GitHub.

Совсем не обязательно создавать репозиторий на Рабочем столе. Клонировать можно в любое место на компьютере. Команду git clone можно выполнять и сразу после открытия терминала.

В появившееся окошко мы можем либо вставить ссылку на репозиторий, которую мы скопировали раньше или, если вы вошли в свой аккаунт на GitHub, выбрать нужный репозиторий по ссылке. Также нам нужно указать папку, в которой будет располагаться наш локальный репозиторий.

### Добавляем файлы в проект

Выбираем файлы, которые вы бы хотели разместить в новом репозитории. Отыщите их на компьютере и перетащите в новую папку репозитория на Рабочем столе.

Проверьте статус проекта.

Откройте терминал и перейдите в папку репозитория. Для проверки обновлений выполните:

git status

Если вы перетаскивали файлы в папку проекта, то потребуется обновить состояние репозитория. Добавлять файлы в репозиторий можно по одному:

git add <имя\_файла>

Либо все сразу:

git add — all

или даже:

git add .

Это ваши предлагаемые изменения. Операцию можно повторить с новыми файлами либо с уже существующими, но измененными. По сути, ничего нового в сам проект вы не добавляете. Вы всего лишь загружаете новые файлы и указываете Git на эти изменения.

Процесс создания коммитов с изменениями начинается с выполнения команды:

git commit -m “<сообщение\_о\_коммите>”

Коммиты изменений добавляются в head (указатель), а не в удаленный репозиторий. Не забудьте заменить текст в скобках и убрать <>. После внесения изменений создается снимок состояния репозитория, для чего используется командаcommit. А через –m добавляется сообщение об этом снимке.

Сохраненные изменения и называются коммитом. При создании коммита вы добавляете сообщение о том, что именно менялось и почему. Так другие люди смогут лучше понять суть изменений.

Теперь ваши изменения сохранены в указателе локальной копии проекта. Для отправки изменений на удаленный репозиторий выполните команду:

git push

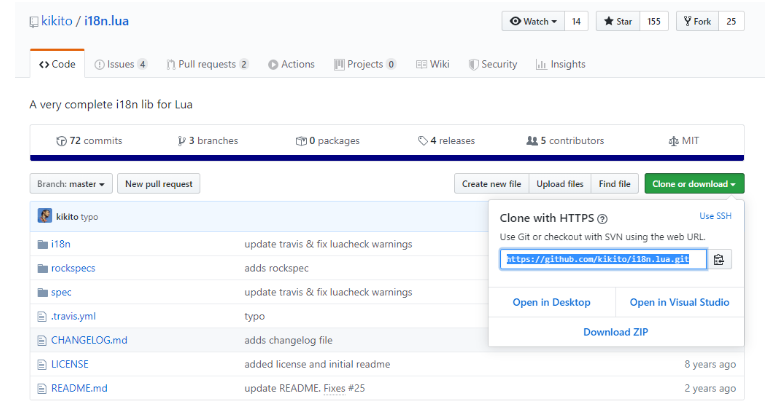
Тем самым вы отправляете изменения напрямую в репозиторий. Если вы работаете на локальном компьютере и хотите, чтобы коммиты отображались в онлайн, то необходимо своевременно отправлять эти изменения на GitHub по команде git push.

Актуальность версии можно проверить в любое время через команду git status.

Итог: у вас есть свой GitHub репозиторий, вы научились добавлять и изменять в нем файлы.

**Задание 3: Клонирование чужих репозиториев**

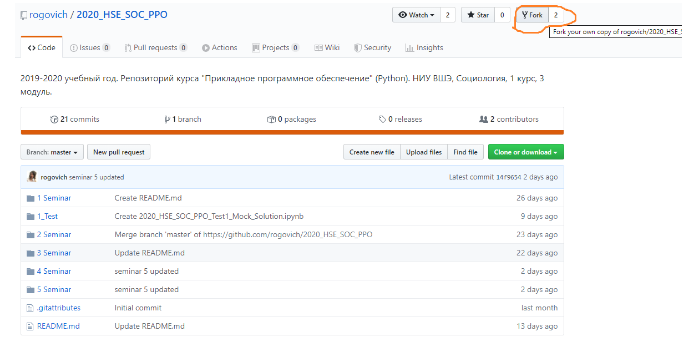
Клонировать можно не только свои репозитории, но и чужие. Для этого найдите нужный репозиторий в поиске на github. И выбираем **Clone or Download**.



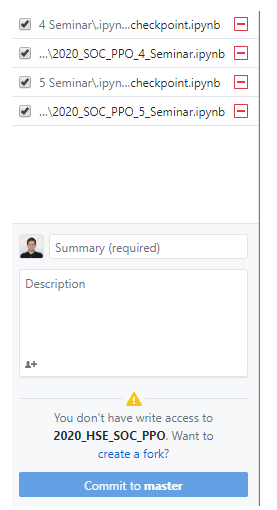
Далее делаем все, как и при копировании своего репозитория. Это позволяет получать файлы, сразу после их добавления или изменения и не требует захода на сайт и ручной проверки на изменения.

**Fork репозитория**

Fork (форк) репозитория это возможность скопировать *чужой* репозитория на свой аккаунт и вносить любые изменения в него, без изменения оригинального репозитория. Можно сделать форк любого доступного репозитория. При создании форка нас спросят, в какой аккаунт мы хотим его добавить.



При клонировании мы только *используем* файлы оригинального репозитория и при создании коммита с какими-то изменениями, GitHub скажет нам, что у нас нет доступа на запись и сам предложит сделать форк. (Если доступ к этому репозиторию у нас есть, то сделать коммит мы сможем.) А если мы сделали форк, то изменения уйдут в нашу копию в нашем аккаунте.



Fork может быть полезен при разработке открытого ПО, например, мы сделали форк алгоритма сжатия, в нем мы изменили функцию сжатия и теперь алгоримт сжимает в 10 раз лучше. Мы можем сделать **Pull request**, т.е. запросить у хозяина оригинального репозитория с алгоритмом сжатия, интегрировать наши изменения в его репозиторий.

**Ветки**

В **git** есть понятие branch (ветка). Ветка - это покоммитный “путь” до некоторого коммита, называемого “концом” (tip) ветки. Мы можем иметь несколько независимых веток при работе. Коммит делается в конкретную ветку, по умолчанию это ветка **master**. Создать новую ветку можно как на сайте, так и в приложении GitHub Desktop. Для этого нужно выбрать вкладку **Current branch** и нажать на **New branch**:

<https://pythonhelp.ru/post/2020-10-21-git-guide/>

<https://zen.yandex.ru/media/zigtelus/kak-dobavit-proekt-na-github-5ea9e8e8e62e151463c491ad>

Выбираем имя и в эту ветку пойдет вся информация с ветки **master** (точнее, новая ветка будет “смотреть” на тот же коммит, что и **master**), в том числе и все файлы:

И теперь мы можем переключать ветки и вносить изменения в конкретную ветку, не затрагивая основную, в данном случае **master**. Например, мы удалим один файл, и изменим другой. Удалённый файл будет отмечен красным минусом, а изменённый - желтой точкой. При этом справа видно, что мы работаем в ветке **Features**.

Делаем коммит в новую ветку и смотрим, что произошло. Как мы видим, в ветке **master** всё осталось, как прежде. Она по прежнему указывает на тот же коммит, что и раньше.

А вот в ветке **Features** удалённого файла уже нет. Переключить ветку можно, нажав на кнопку **Branch** с названием ветки:

Ветки удобно использовать для добавления новых функция, что они не ломали *рабочий* код до новой функции. После разработки ветку можно объединить с **master** (merge, смёржить, слить) сделав так называемый **Pull request**.

**Создание репозитория из GitHub Desktop**

Как говорилось ранее, новый репозиторий можно создать и из самого приложения. Для этого идем в **File/New repository**:

Указываем все данные аналогично тому как создавали на сайте и нажимаем **Create repository**:

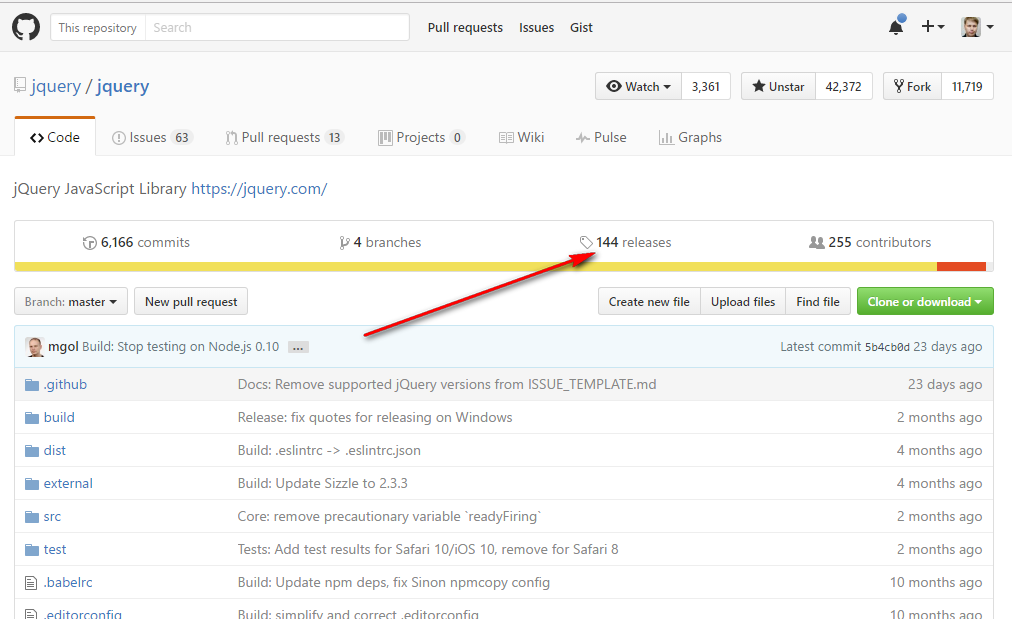
Не забудьте нажать на **Publish repository**, чтобы он ушёл на сайт.

Далее еще раз укажите имя уже на сайте и всё.

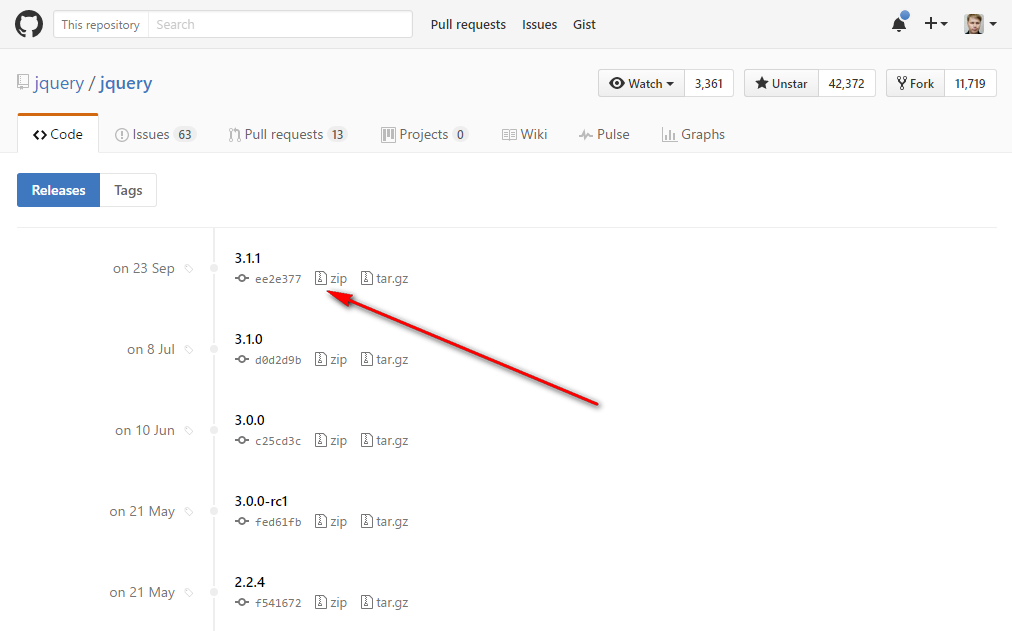
**Практическая работа: Cкачивание различных элементов с Gethab**

Задание 1: [Скачайте релиза проекта](http://blog.harrix.org/article/933#a_menu)

1. Если вам нужен не сам исходный код проекта, а уже готовый вариант приложения, то лучше посмотреть релизы данного проекта.
2. Перейдите по ссылке к релизам проекта:

[](http://blog.harrix.org/wp-content/uploads/2014/01/2016-11-23_211943.png)

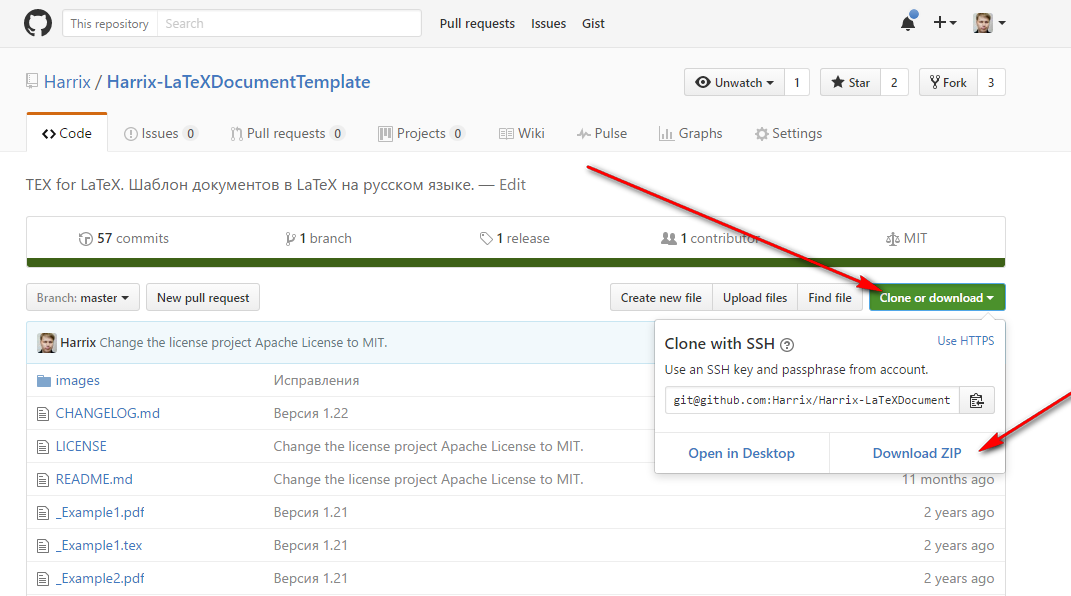
1. А там скачивайте ту версию релиза, которая вам нужна:

[](http://blog.harrix.org/wp-content/uploads/2014/01/2016-11-23_212113.png)

1. Прошу обратить внимание, что не всякий проект на GitHub обладает релизами. Некоторые разработчики или не знают об этой функции, или же не занимаются релизами, а всё сохраняют в главном репозитории.

**Задание 2:** [**Скачивание целого проекта**](http://blog.harrix.org/article/933#a_menu)

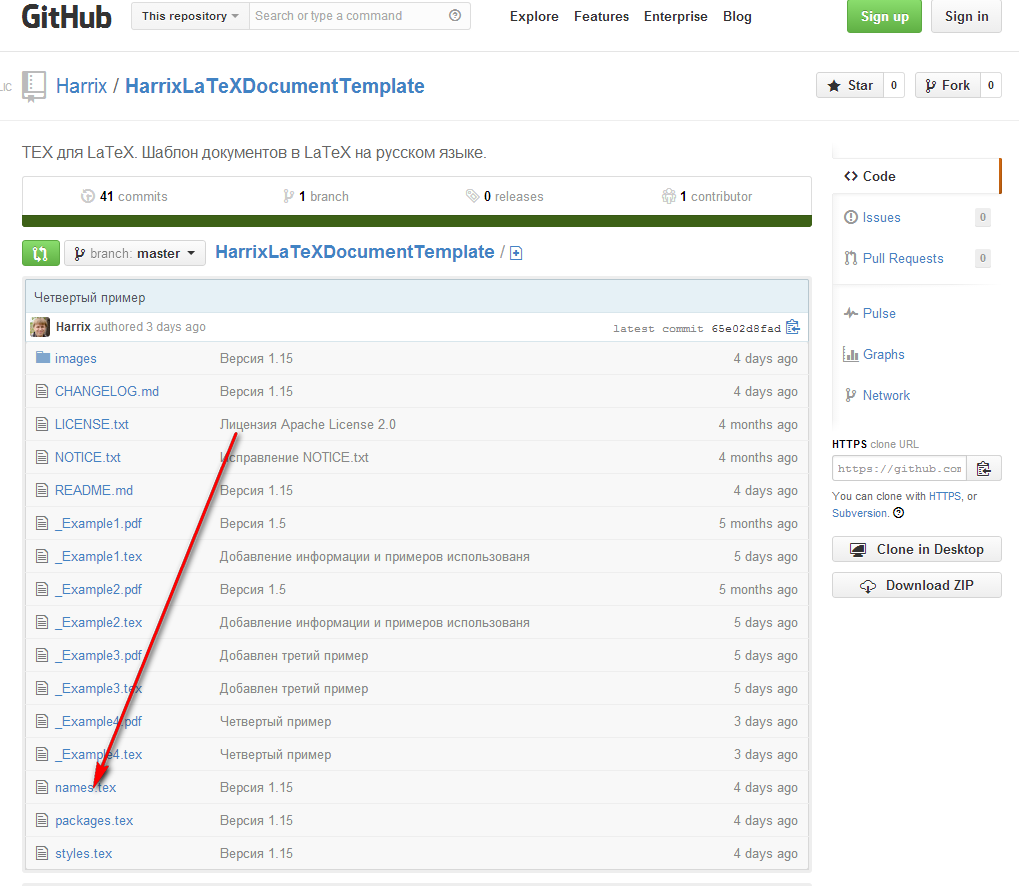
1. Например Вы заходите на проект [Harrix LaTeXDocumentTemplate](https://github.com/Harrix/Harrix-LaTeXDocumentTemplate). И вам нужно скачать его весь.
2. Справа наверху есть кнопка Download ZIP:

[](http://blog.harrix.org/wp-content/uploads/2014/01/2016-11-23_211756.png)

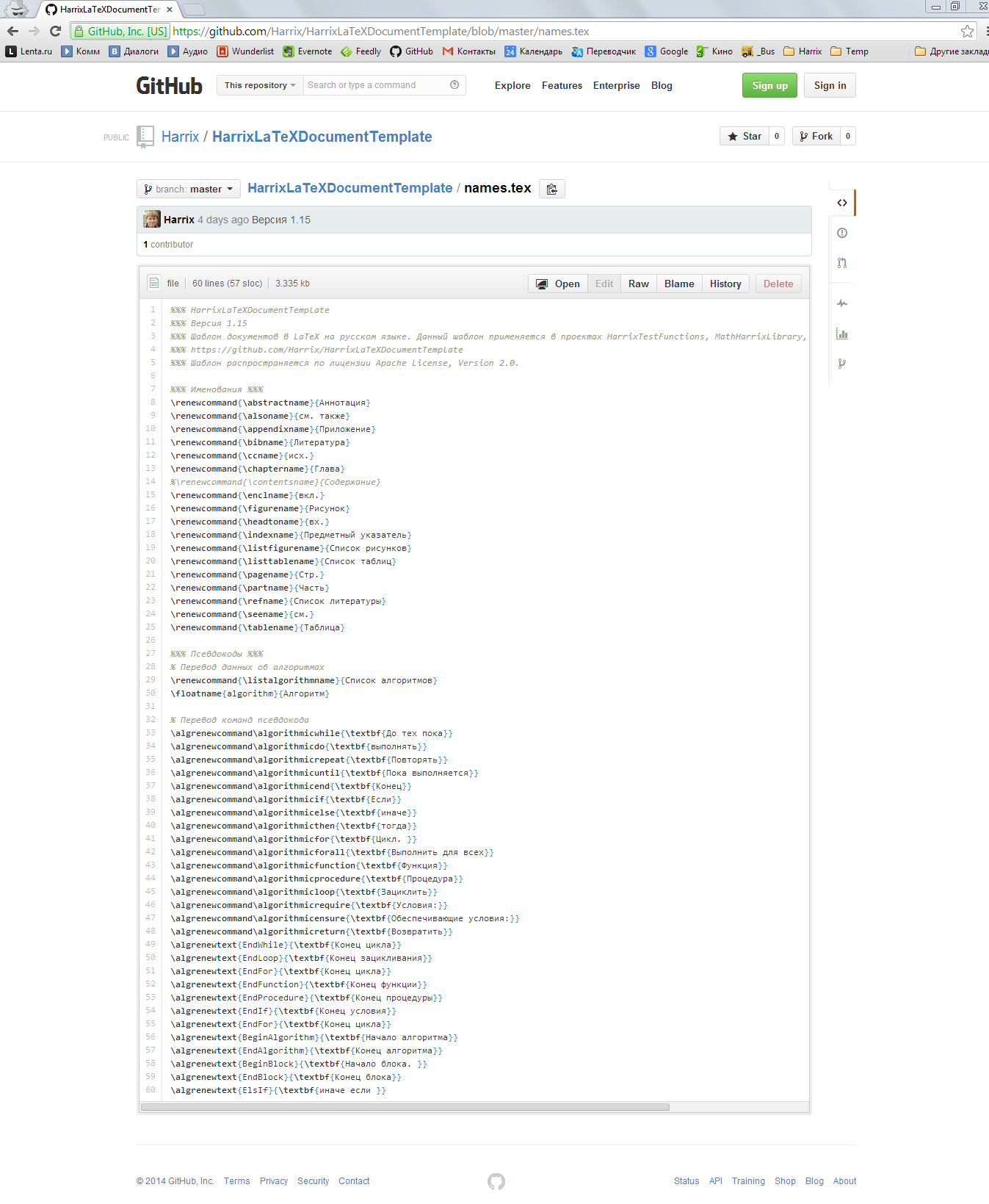
1. Щелкайте по ней, и zip архив с проектом будет скачен к вам на компьютер.

Задание 3: [Скачивание отдельного файла (текстового)](http://blog.harrix.org/article/933#a_menu)

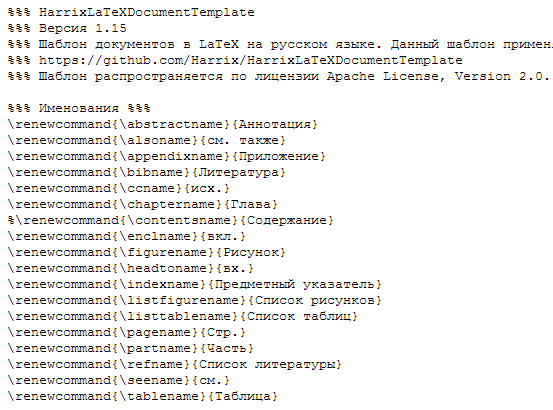
1. Например, Вы заходите на проект [Harrix LaTeXDocumentTemplate](https://github.com/Harrix/Harrix-LaTeXDocumentTemplate). И вам нужно скачать отдельный файл, например, [names.tex](https://github.com/Harrix/Harrix-LaTeXDocumentTemplate/blob/master/names.tex). Конечно, лучше весь проект скачать, как показано выше, но если не хотите, то читайте дальше.
2. Щелкните по данному файлу (или перейдите по ссылке, если у вас есть прямая ссылка, как приведена выше):

[](http://blog.harrix.org/wp-content/uploads/2014/01/036.png)

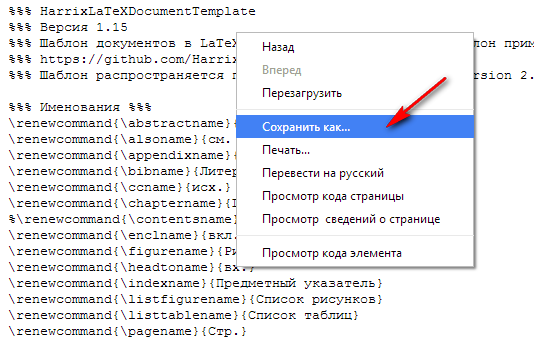
1. Откроется просмотр содержимого файла.

[](http://blog.harrix.org/wp-content/uploads/2014/01/044.png)

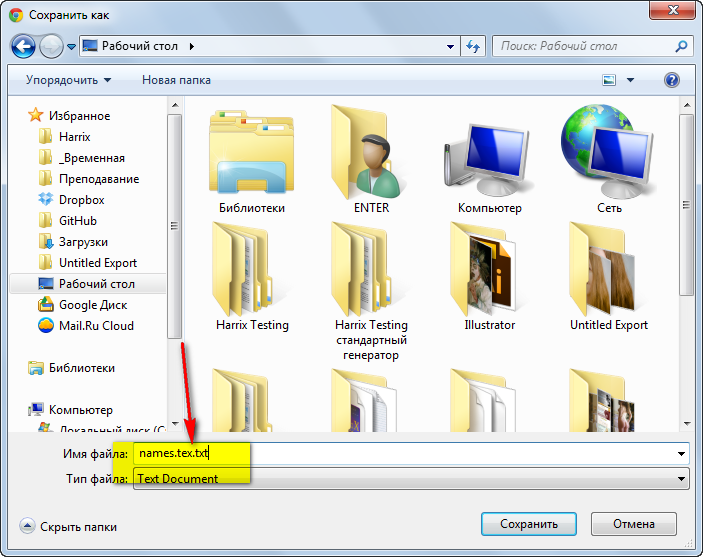
1. Кликнете по кнопке Raw. Откроется чистый код файла:

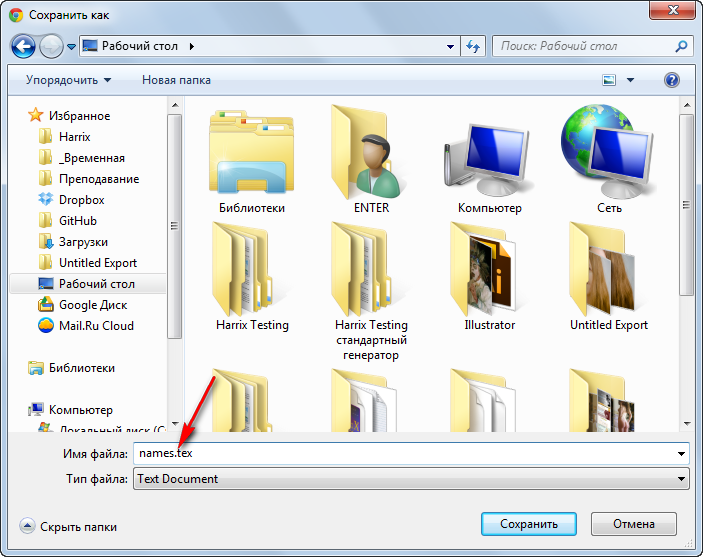
[](http://blog.harrix.org/wp-content/uploads/2014/01/063.png)

1. Правая кнопка мыши нажимается, и выбираем пункт меню Сохранить как....

[](http://blog.harrix.org/wp-content/uploads/2014/01/073.png)

1. В имени файла убираем лишнее расширение:

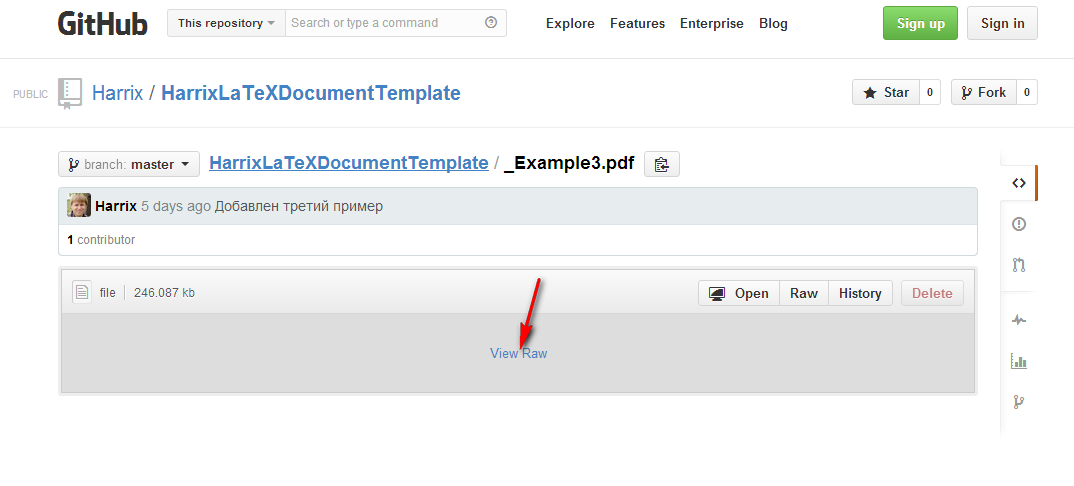
[](http://blog.harrix.org/wp-content/uploads/2014/01/082.png)

[](http://blog.harrix.org/wp-content/uploads/2014/01/093.png)

1. Теперь у вас есть сохраненный файл.

[Скачивание отдельного файла (pdf)](http://blog.harrix.org/article/933#a_menu)

1. Например Вы заходите на проект [Harrix LaTeXDocumentTemplate](https://github.com/Harrix/Harrix-LaTeXDocumentTemplate). И вам нужно скачать отдельный файл, например, [\_Example3.pdf](https://github.com/Harrix/Harrix-LaTeXDocumentTemplate/blob/master/_Example3.pdf). Конечно, лучше весь проект скачать, как показано выше, но если не хотите, то читайте дальше.
2. Щелкните по данному файлу (или перейдите по ссылке, если у вас есть прямая ссылка, как приведена выше)
3. Откроется просмотр содержимого файла. Щелкайте по View Raw, и файл или откроется (а там через правую кнопку мыши сохраняйте), или же через начнет скачиваться.

[](http://blog.harrix.org/wp-content/uploads/2014/01/111.png)

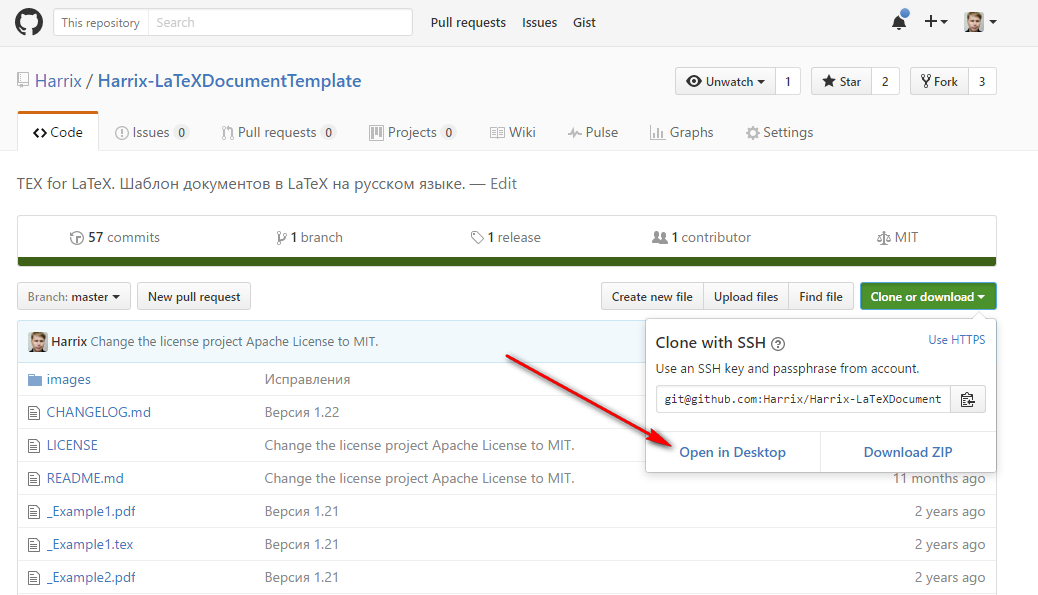
1. Теперь у вас есть сохраненный файл.

С остальными форматами файлов всё аналогично.

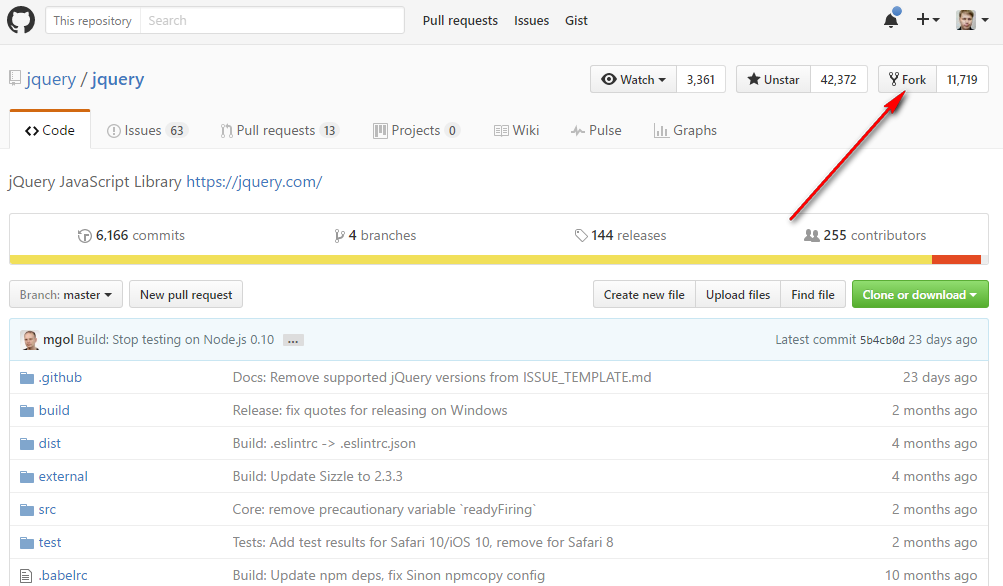
[Стандартный метод для разработчиков](http://blog.harrix.org/article/933#a_menu)

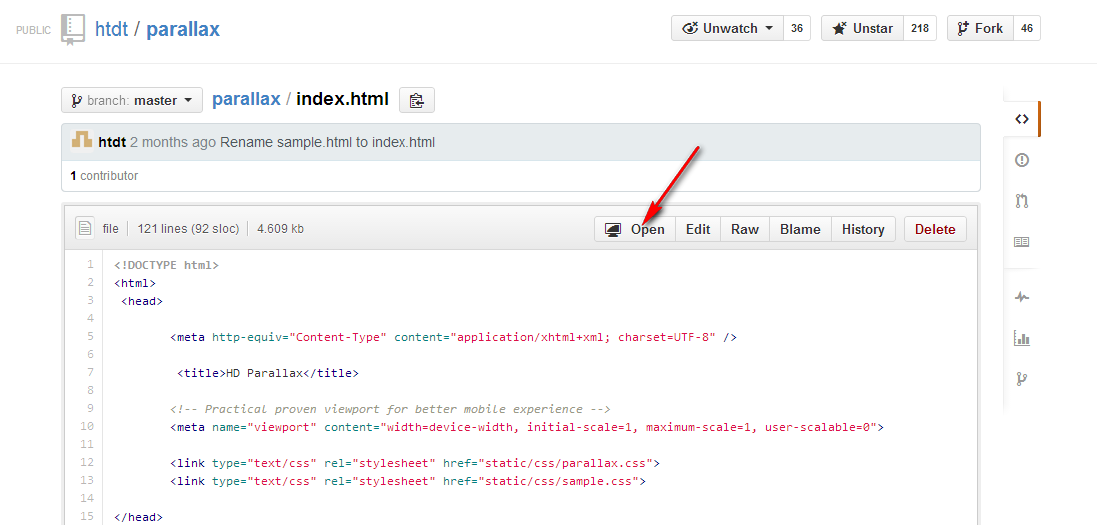
Если вы активный пользователь Github и у вас есть уже свой репозиторий, то вы и так все знаете. Но для полноты способов я здесь его приведу.

1. Заходите на сайт под своей учетной записью.
2. Переходите на нужный вам репозиторий (проект).
3. Если не хотите создавать отдельную ветку для развития проекта, то нажимайте Open in Desktop. А там уже через клиент Github обновляете его и копия проекта появляется у вас на компьютере.

[](http://blog.harrix.org/wp-content/uploads/2014/01/2016-11-23_212242.png)

1. Если хотите создать свою ветку продукта, то делаете Fork проекта. И проект в виде копии появляется у вас в коллекции репозиториев. А там уже через клиент Github обновляете его и копия проекта появляется у вас на компьютере.

[](http://blog.harrix.org/wp-content/uploads/2014/01/2016-11-23_212412.png)

Также отдельные файлы открываются через клиент Github через кнопку Open:  
[](http://blog.harrix.org/wp-content/uploads/2014/01/101.png)

https://docs.microsoft.com/ru-ru/visualstudio/version-control/git-with-visual-studio?view=vs-2019