GeekBrains

Углубляемся в контроль версий

Оглавление

[00.01.03]

Приветствие

Добрый день, дорогие друзья. Рады приветствовать вас на третьей лекции нашего курса «Введение в контроль версий». Сегодня мы поговорим про работу с удалёнными репозиториями.

Слово «удалённые» не означает удалённые нами репозитории. На самом деле мы работаем с репозиториями, которые находятся не на нашем компьютере. То есть у нас есть локальные репозитории, находящиеся на нашем ноутбуке или компьютере, и удалённые, располагающиеся удалённо от нас, где-то в другом месте. Это может быть какой-то сервер, компьютер нашего друга или ещё что-то. Главное, что эти репозитории находится не на нашем компьютере. Поэтому сегодня мы с этим познакомимся чуть ближе.

[00.01.03]

На этом уроке

1. Изучим команду git clone.
2. Познакомимся с командой git pull.
3. Рассмотрим команду git push.

[00.02.42]

Learn Git Branching

Итак, прежде чем мы пойдём дальше, рассмотрим один интерактивный тренажёр, на котором можно отработать всё, что сейчас разбираем, и узнать много нового. Этот тренажёр — книга и инструмент для обучения. Посмотрим, как он работает.

Для этого вам надо перейти на сайт **LearnGitBranching**. Означает «изучи ветвление в Git». На самом деле здесь есть не только ветвление, но и работа с удалённым репозиторием, а также перемещения по разным коммитам. Всё, что мы делали, здесь тоже можно сделать. Есть и дополнительные команды. Они необязательны для изучения прямо сейчас, но если у вас есть время и вам понравилось работать с Git, можете пройти весь тренажёр. Говоря о получении необходимого объёма знаний, информации в этом тренажёре и той, что разбирается сейчас,

практически достаточно для любого Junior-программиста. Зачастую даже начинающие программисты не так хорошо знают материал, который разбираем мы.

Итак, переходим на сайт learngitbranching.js.org. Ссылка будет под этой лекцией. Далее просто следуем всем инструкциям. Сайт русифицирован и очень удобен. Пропустим демо и посмотрим, что здесь происходит.

Сам процесс обучения строится на модулях. У нас есть модуль «Введение», «Едем дальше» и так далее. Каждый модуль состоит из нескольких уроков. Есть работа не только с локальным репозиторием в блоке «Основы», но и работа с удалёнными репозиториями. Это мы будем изучать на текущей лекции в ближайшее время.

Работаем с сайтом, последовательно выбирая все уроки. Например, возьмём первый урок «Коммиты в GIT». Здесь есть теоретическое описание, которое можно прочитать, и визуализация того, что будет происходить с деревом коммитов.

Допустим, у нас есть два коммита. Мы находимся в ветке, расположенной на слайде справа. Когда совершаем новый коммит, в дереве появляется дополнительное сохранение, то есть фиксация. Здесь мы уже не занимаемся редактированием файлов, не пишем ничего на Markdown, а только смотрим, что происходит с деревом коммитов, с деревом сохранений, когда запускаем те или иные команды на Git.

Есть набор заданий. Задания выглядят следующим образом:

- есть текущее состояние проекта;
- два коммита: C0, C1;
- и цель уровня.

Надо то, что располагается справа, привести к тому, что находится в цели. Видим, что надо сделать ещё два коммита. Должен появиться коммит C2 и коммит C3. Выполнять задания можно так же, как мы сейчас делали в терминале.

Слева есть стилизованный терминал. Здесь можно писать те команды, которые мы изучали. Например, написать команду git commit. Дополнительный месседж писать не обязательно, потому что мы просто смотрим, как меняется структура проекта. Если самостоятельно запускаем команду git commit и делаем всё без ошибок и опечаток, в текущем состоянии трафика появится дополнительный коммит. Можно сделать это ещё раз. Обратите внимание, что терминал позволяет переключаться клавишами так же, как и терминал в VS Code.

Когда получилось то же состояние, которое надо было получить в цели текущего уровня, система автоматически понимает, что мы решили задачу. Она говорит, сколько команд используется для решения и из скольки команд состоит решение. Если у нас столько же или даже меньше команд, всё отлично. А если мы использовали больше команд, система предложит перерешать этот уровень, чтобы решение стало оптимальным.

Если не можете решить какую-то задачку, посмотрите на решение авторов этого интерактивного учебника. Но лучше всё-таки аккуратно прорешать все задачи, которые здесь есть. Посмотрите теорию, это будет дополнительным касанием. Ещё здесь есть дополнительные команды, которые мы в рамках текущего курса, в рамках знакомства с контролем версий, не изучали. Они будут полезны в вашей дальнейшей деятельности. Таким образом, на этом сайте потребуется пройти все занятия.

[00.07.23]

Работа с удалёнными репозиториями в git

Теперь перейдём к теме сегодняшней лекции. Сейчас с репозиториями мы работаем только локально.

На экране видно, что мы делаем. Например, берём какую-то папку. Эта папка может быть пустой или содержать какие-то файлы. После этого внутри указанной папки запускаем команду **git init**. То есть мы инициализируем репозиторий на своём компьютере. Он появляется локально. Далее производим какие-то изменения, возможно, в одной ветке или в нескольких разных ветках, как делали на второй лекции. Затем добавляем в этот репозиторий изменения.

На нашем компьютере всё происходит локально только в этой папке. Ничего не выходит наружу. Посмотрим, как поделиться результатами своей работы с кем-то.

Например, у нас есть репозиторий одного человека и репозиторий другого человека. И мы хотим всё передать третьему. Для этого можно воспользоваться флешкой, заархивировать свою папку, передать её по почте или переложить в облачное хранилище. Можно переслать папку физически, что зачастую не очень удобно, но технически возможно. Например, можете направить свою версию одногруппнику, он в ней что-то поменяет и переделанный вариант

целиком отправит обратно. Да, это не подход к контролю версий, но в целом так уже работать можно.

[00.08.57]

Git vs GitHub

Но если захотим работать так же, как те, кто придумывал системы контроля версий, потребуется воспользоваться удалёнными репозиториями. И здесь мы впервые встретимся с сервисом **GitHub**. Надо понимать и чётко осознавать, что Git и GitHub — разные вещи.

1. **Git** — это программа, которая устанавливается на компьютер, где локально выполняет указанные вами команды.

2. **GitHub** — это сервис компании Microsoft, который позволяет интегрироваться с программой Git и настроить удалённую работу с вашим репозиторием.

Таким образом, вы можете работать локально или хранить свой репозиторий на GitHub, используя команды, входящие в программу Git.

На слайде есть несколько основных параметров, которые позволяют лучше понимать, что такое Git и GitHub. Ознакомьтесь с тем, что мы здесь написали и запомните: **Git** — программа, а **GitHub** — сервис, который позволяет удобнее работать с этой программой и настроить удалённую работу с репозиториями.

GitHub — далеко не единственный сервис. По ходу вашего обучения, а уж тем более дальнейшей работы, вы познакомитесь со многими другими сервисами. Но GitHub в плане бесплатных и доступных сегодня считается самым популярным сервисом.

На этом сервисе есть много программ, просматриваемых исходных кодов, написанных разными программистами, и проекты, в которых можно поучаствовать. Сегодня мы немного коснёмся темы участия в разработке какого-либо большого проекта, если он находится на сервисе GitHub.

[00.11.03]

Командная работа

Вернёмся к уже хорошо знакомой нам схеме, где есть наши черновики и черновики от какого-нибудь заказчика, научного руководителя, редактора, коллеги или ещё кого-то. На прошлом занятии мы рассмотрели, как организовывать подобную схему с использованием ветвлений Git — создавать ветки, выводить часть работы в отдельную ветку, а после выполнения всех действий заводить результат обратно в основную ветку.

У нас была ветка с чистовиком, мы называли её «Мастер», и соседние ветки, где выполнялись какие-то действия. Соответственно, эта схема локально уже может работать. Но на слайде видно, что изначально сверху идёт черновик наш, а снизу — черновик с комментариями заказчика. То есть помимо нас кто-то ещё работает с этим репозиторием. И настроить процесс таким же образом, если весь репозиторий находится только на нашем компьютере, невозможно. Нам надо, чтобы и у других людей был доступ к этому репозиторию. Этим мы сегодня и займёмся.

[00.12.15]

Практика

Работа с сайтом github.com

Посмотрим, как всё работает на практике. Для начала создадим папку на нашем рабочем столе. Назовём её Lesson 3 и откроем в редакторе VS Code. Так мы начинали все лекции, но создавали локальный репозиторий с использованием git init. Теперь поступим несколько иначе, не будем создавать свой репозиторий, а возьмём готовый. Для этого перейдём на сайт github.com. Практически все, так или иначе, знакомы с этим сайтом, как минимум слышали о нём.

На этом сайте есть поиск. Вы можете находить здесь разных авторов и проекты. Посмотрим на один из готовых репозиториев.

В этот репозиторий я добавил файлы второй лекции. Скачаем их так, чтобы на нашем компьютере они стали локальным репозиторием.

Эти файлы можно посмотреть и на самом сайте. Открывая их, можно увидеть, как они выглядят в редакторе после обработки Markdown. Есть, например, файл .gitignore, здесь мы, соответственно, игнорируем нашу Тефтельку. И есть зелёная кнопка Code. Если мы нажмём на

неё, появится строка с адресом. Здесь написано Use Git or checkout with SVN. **SVN** — это системы контроля версий.

Скопируем эту строку и перейдём в код. Выберем строку HTTPS, нажмём «Копировать», и после этого нам понадобится новая команда внутри VS Code. Откроем терминал. В терминале дадим новую команду Git: скажем git clone и укажем тот адрес, который скопировали.

После этого Git скопирует репозиторий, находящийся на сервисе GitHub, в наш локальный репозиторий. То есть на нашем компьютере появится полная копия этого репозитория. Но прежде чем мы это сделаем, сначала вызовем команду git status. Убедимся, что Cit не настроен. Появилась надпись, что это не Git-репозиторий, возникла какая-то фатальная ошибка, и наша программа отказывается работать дальше. То есть мы убедились, что здесь лежит пустая папка без каких-либо репозиториев.

Снова вызовем команду git clone. Вставим туда скопированный адрес и нажмём на Enter. Внутри папки Lesson 3 появилась новая папка с именем version_control_lection_3. Узнаем, почему у неё такое название.

Дело в том, что сам репозиторий на GitHub называется именно так. То есть Git скачал репозиторий, который находился на GitHub, создал папку с таким же названием и поместил всё, что находилось в репозитории. Обратите внимание, внутри этой папки есть файл .gitignore и файл Markdown instruction.md. Именно эти файлы находились в нашем репозитории на сайте GitHub.

[00.16.37]

Локальная работа с файлами

Теперь поработаем с этими файлами локально. Можно посмотреть, например, на текущий статус. Нажимаем Enter и видим ту же фатальную ошибку. Разберёмся, почему возникает такая ошибка.

Обратите внимание, что в последней строке указано, кто выполняет операции. То есть здесь указан мой аккаунт и указан компьютер, на котором я работаю. Затем идёт папка, в которой мы находимся.

Сейчас команда git status запускается в папке Lesson 3, но в ней нет репозитория. В папке Lesson 3 есть другая папка, где уже лежит репозиторий. Поэтому чтобы с Git работать

аккуратно, надо поместить всё в папку version_control_lection_3. В разных операционных системах это делается по-разному. В Linux и Mac, чтобы поменять местоположение, выполняется команда **cd — change directory — поменять директорию**. Далее надо просто указать, куда перейти. Перейдём в папку version_control_lection_3.

Обратите внимание, что выдача терминала поменялась. Там, где раньше было написано Lesson 3, теперь указано имя нашего репозитория. И если сейчас мы запустим команду git status, то увидим уже знакомую выдачу. У нас есть информация, что мы находимся на ветке master и нам не надо ничего коммитить. Соответственно, можно спокойно продолжать работу.

Посмотрим, что хранится в нашем репозитории. Для этого снова очистим выдачу терминала и посмотрим на сами файлы. Файл Teftelka здесь нет, но есть текст, замещающий его.

В тексте есть картинка. У нас есть информация, что здесь должна быть Тефтелька, но, так как самого файла в репозитории нет, мы видим не фотографию, а замещающий текст. Можно посмотреть на лог изменений и увидеть:

- какие изменения происходили;
- какие коммиты были в этом файле, когда и кем сделаны;
- и что вообще происходило.

То есть вся история здесь есть.

Мы можем:

- перемещаться по всем коммитам;
- переходить с одного на другое;
- проследить, как выполнялась работа;
- посмотреть на лог в виде графа;
- посмотреть, есть ли ветки, и если есть, то как осуществлялась работа.

Обратите внимание, что последние два коммита уже сделаны в одной ветке, до этого было слияние с разрешением конфликта версий. Наблюдается также расхождение, когда стартовала другая ветка. То есть можно посмотреть всю историю работы над этим файлом.

По факту у вас появляется полная копия представленной работы, будто вы делали её самостоятельно. То есть вся история у вас есть. Вы можете остановиться, продолжить работу с этим файлом, и у вас появится собственная версия репозитория. Однако всё происходит локально. С версией, которая находится на GitHub, она не связана.

Мы рассмотрели второй способ создания репозиториев. Первый способ осуществлялся через git init, когда мы создавали собственный репозиторий. Второй способ реализуется через git clone: мы можем взять чужой репозитории, который, кстати, тоже был когда-то создан через git init, но продолжить работу с ним уже локально.

[00.20.57]

Другие операции

Посмотрим, как выполнять другие операции и сделать так, чтобы созданный нами локально репозиторий оказался на GitHub. Мы узнали, как взять чужой репозиторий, который уже находится на GitHub. Теперь выясним, как переместить наш репозиторий в интернет.

Для этого сделаем новую папку и назовём её Lesson 3_1. Откроем эту папку и создам в ней новый репозиторий. Сделаем это, воспользовавшись знакомыми нам операциями:

1. Откроем терминал.
2. Убедимся, что здесь нет никакого репозитория.
3. Создадим его через git init. Эта операция нам знакома.
4. Создадим здесь какой-нибудь файл и назовём его Hello world.md.
5. Напишем «Привет».
6. Сделаем коммит и узнаем статус.
7. Добавим файл Hello world.
8. Сделаем свой первый коммит.
9. Оставим месседж Initial commit.

У нас есть какой-то репозиторий, в котором мы что-то делали. Теперь нам надо, чтобы он оказался в интернете.

Для этого сначала создаём на каком-либо сервисе, на который хотим направить репозиторий, свой аккаунт. Обратите внимание, что скачать чужой репозиторий можно и без создания собственного аккаунта на этом сервисе. Если у вас есть ссылка, как на экране, вам, по сути, ничего другого не требуется. А если хотите залить свой репозиторий в какой-то сервис, на этом сервисе придётся создать собственный аккаунт. Создадим аккаунт.

Аккаунт можно назвать как угодно. Затем в правом верхнем углу надо нажать плюсик, выбрать новый репозиторий и дать ему какое-то имя. Назовём его пока Test. Так делать нежелательно, но важно знать, что имя может быть любым. Остальные параметры оставляем по умолчанию, ничего не редактируем, создаём новый репозиторий.

GitHub предлагает советы. Он подсказывает, что можно сделать, чтобы начать работать с этим репозиторием. У нас есть два варианта. GitHub говорит, что на самом деле вариантов три.

1. Можно создать новый репозиторий через терминал и начать с этим работать.
2. Уже существующий репозиторий привязать к удалённому репозиторию.
3. Импортировать код из другого репозитория.

У нас уже есть репозиторий, который работает локально. И сейчас мы хотим эту информацию отправить на GitHub, чтобы она появилась в интернете. Для этого здесь есть подсказки.

Нам надо ввести git remote add origin и строчку, указанную на слайде. После этого указываем, что основная ветка — main, и отправляем изменения в репозиторий. Подробно все эти команды мы сейчас разбирать не будем.

У нас всегда есть возможность скопировать перечисленные команды и просто вставить в свой проект, и всё прекрасно заработает. Сделаем только пометку, что git remote удалённый. То есть мы говорим программе Git, что появляется новый удалённый репозиторий. Далее указываем ссылку с адресом на удалённый репозиторий, с которой надо будет работать. Каждый раз, когда мы захотим что-то отправить в интернет, Git будет знать, что отправлять требуется на указанный адрес. По сути, мы связываем наш локальный репозиторий с удалённым репозиторием. Далее указываем, какая ветка будет основной, а затем направляем то, что у нас есть, на локальный репозиторий в интернет. Проведём все эти операции последовательно.

Слово remote мы разобрали. Origin — просто название, которое даём удалённому репозиторию, branch — ветка, a push означает «толкать» или «направлять вперёд».

Соответственно, когда мы вызываем команду git push, то даём команду Git протолкнуть всё, что мы сделали у себя локально, куда-то на удалённой репозиторий, в интернет. Отправится в репозиторий origin, который мы задали выше.

[00.26.49]

Шаги операции

1. Скопируем первую строку.
2. Введём её в наш терминал. Теперь сообщений об ошибках нет, значит, всё в порядке. Но очистим терминал, что было проще смотреть.
3. Скопируем вторую строку. Вторая строка укажет, какая ветка основная.
4. Третья строка направит в интернет (в репозиторий) то, что мы делали.

Обратите внимание, что при первом связывании Git, который находится локально, с сервисом GitHub, их потребуется «подружить». Надо сделать так, чтобы прошла авторизация. Это важно, так как команда push записывает изменения в тот репозиторий, куда их направляете. И если бы мы нигде не вводили логин и пароль от этого репозитория, то в любой репозиторий смогли бы направлять собственные изменения. Очевидно, такого быть не должно. Поэтому сначала понадобится сделать так, чтобы GitHub убедился, что мы действительно владельцы этого репозитория и имеем право на внесение туда изменений.

При первой попытке сделать push VS Code поможет пройти эту авторизацию. На экране она уже пройдена, поэтому git push сработает. Однако помните, при первом push в новый репозиторий придётся авторизоваться. Надо подружить ваш Git локально с GitHub, или другим сервисом, куда будете направлять свой код.

В результате всё прошло успешно. Взглянем, что получилось в нашем репозитории. Обновляем страничку и видим, что текст (файл), созданный в нашем репозитории, теперь оказался в интернете. Он находится на нашем репозитории на GitHub.

[00.29.10]

Работа с публичным репозиторием

Если вы работаете в команде из нескольких человек или, например, хотите кому-то передать сделанное на своём репозитории, сдать практическое задание, то архивировать и

прикладывать папку не обязательно. Её можно выгрузить на GitHub и передать ссылку, указанную на слайде.

Если ваш репозиторий публичный, доступный для всех, есть надпись public около имени репозитория, то любой человек, владеющий этой ссылкой, сможет скопировать и сохранить себе ваш репозиторий. Соответственно, ваш друг, преподаватель или кто-либо ещё, обладая указанной ссылкой, сможет увидеть всё, что вы делали в репозитории, скачать локально через команду git clone и посмотреть историю изменений проделанной вами работы. Очистим то, что здесь происходит.

Мы научились направлять в интернет информацию с нашего локального репозитория. Узнали, как копировать чужие репозитории или собственные, если начали работать на другом компьютере. Например, используя команду git clone, копируем, а через git push, если всё настроено, направляем в интернет.

Пока перед нами разовая акция. Мы могли скачать один раз и направить уже созданное. Но работа идёт итеративно. Вы сделали какие-то изменения, направили их в интернет, и ваш друг посмотрел получившуюся версию. Спустя какое-то время вы снова произвели какие-то изменения, после чего вам опять потребовалось направить в интернет сделанное.

Сделаем несколько итераций и посмотрим, как это делается и что происходит на GitHub, когда производим эти операции. Для этого добавим некоторые изменения в файл. Например, напишем «Тефтелька — замечательный котик» и поставим смайлик. Пусть эта информация попадёт в интернет.

Ещё раз пройдём знакомый путь и посмотрим на статус. Всё нормально: у нас один модифицированный файл, соответственно, можем его добавить через git add, написать Hello world, посмотреть на текущий статус и сделать коммит. Далее сделаем коммит и запишем «Важный текст про Тефтельку». Кто пропустил предыдущую лекцию или уже забыл, Тефтелька — это котик, фотографию которого мы размещали в Markdown.

Итак, git commit прошёл. Посмотрим, что происходит в статусе. Всё нормально, ничего нового для коммита нет, наше рабочее дерево чистое.

Теперь посмотрим, что происходит на GitHub. Нетрудно заметить, что ничего на GitHub не происходит.

Все изменения, которые мы делаем, только локальные. То есть вносим какие-то изменения на своём компьютере, в локальной копии репозитория. Если хотим, чтобы они попали в интернет, на удалённый репозиторий, надо дать команду Git. Эта команда вам уже известна.

Когда мы привязали наш удалённый репозиторий к локальному, а Git установился между двумя этими репозиториями, можем просто вызвать команду git push. Эта команда направит в интернет все только что произведённые изменения. То есть наша локальная копия перейдёт в интернет.

Убедимся в этом, перейдя на GitHub. Для этого обновим страницу — появляется информация «Тефтелька — замечательный котик», которая теперь есть в интернете.

То же самое происходит и в обратном направлении. Предположим, что вы с одного компьютера, сидя дома, написали некоторый текст, а потом куда-то ушли, где на другом компьютере внесли какие-то изменения. То есть сделав локальный репозиторий на другом компьютере, что-то в нём поменяв и загрузив на GitHub, у вас получилась более актуальная версия, чем на текущем компьютере.

Теперь изменим наш файл напрямую в GitHub. Добавим, например, информацию: «Совершенно с этим согласен!». После этого сохраним эти изменения. Добавим коммит «Изменения с другого компьютера».

Предположим, что на другом компьютере вы сделали локальный репозиторий, добавили изменения, то есть новую строчку, и закоммитили их сообщением. Направили всё на GitHub, после чего там появилась версия из трёх строчек, а на локальном компьютере, с которого мы только что работали, вышло всего две строчки.

Обновим файл, то есть закроем. Откроем его заново — опять ничего не происходит. Версия старая.

На GitHub мы можем также посмотреть на историю изменений. Есть ветки, между которыми можно переключаться. А также можно посмотреть на все коммиты, которые были сделаны.

Например, здесь есть кнопка history, нажав на которую мы увидим три коммита: Initial commit, «Важный текст про Тефтельку» и «Изменения с другого компьютера». И если первые два коммита совершал один пользователь, то последний коммит — уже другой пользователь. У них разные аккаунты, это делали разные люди, возможно, с разных компьютеров. Соответственно, если мы опять очистим выдачу и вызовем git log, то увидим только два коммита: «Важный текст про Тефтельку» и Initial commit. То есть третьего коммита, который был совершён, нет.

[00.36.20]

Синхронизация

Теперь возьмём актуальную версию с GitHub и сделаем так, чтобы на локальном компьютере наш локальный репозиторий был полностью синхронизирован. Сделаем это через команду git pull. Pull значит «тянуть», то есть стянуть с удалённого репозитория, который мы создали, а push — «толкать». Таким образом, из репозитория remote origin стянутся все изменения.

Однако pull служит составной командой. Она не только подгрузит все изменения, но и попытается смержить наши ветки. То есть указанная команда попытается произвести слияние состояния, которое было на GitHub, с состоянием, происходящим локально. Запустим эту команду и посмотрим, что произойдёт.

Запускаем git pull и видим, что появился текст «Совершенно с этим согласен». То есть всё пришло на наш компьютер, merge прошёл успешно, никаких конфликтов не возникло, и у нас получилось то же состояние файла, что было на удалённом репозитории.

Очистим выдачу и взглянем на git log. Теперь он показывает «Изменения с другого компьютера», «Важный текст про Тефтельку» и Initial commit.

Обратите внимание на экран: в одном месте указан один пользователь, а выше — другой. Именно поэтому, когда вы начинаете пользоваться Git, требуется представиться, чтобы Git знал, кто производит изменения. Если изменения делаются разными людьми, то, вы всегда сможете узнать, кто сделал тот или иной коммит. При работе в команде это важно. Теперь очистим выдачу терминала.

Вам потребуется проделать все вышеуказанные действия локально. Если аккаунта ещё нет, и вы ни разу не пушили из терминала в какой-либо репозиторий, это займёт некоторое время. Поэтому по окончании лекции проведите все эти операции.

[00.38.57]

Действия

1. Создать аккаунт на GitHub. *Если у вас этого нет, запишем полный адрес GitHub.com.*
2. Создать локальный репозиторий. *Это должны сделать вы. Для этого надо создать папку, внутри неё вызвать команду git init, совершить какие-то действия, создать файлы, чтобы появился хотя бы один коммит.*

3. «Подружить» ваш локальный и удалённый репозитории. *GitHub при создании нового репозитория подскажет, как это можно сделать.*

4. Отправить (push) ваш локальный репозиторий в удалённый (на GitHub), при этом вам, возможно, потребуется авторизоваться на удалённом репозитории. *Если сделаете это один раз, «подружите» ваш редактор VS Code с GitHub, в дальнейшем эту операцию проводить уже не понадобится.*

5. Провести изменения «с другого компьютера». *Естественно, не надо искать другой компьютер. Можете сделать это на GitHub, как сделали мы.*

6. Выкачать (pull) актуальное состояние из удалённого репозитория.

Набор действий, описанный сейчас на экране, вам потребуется проделать. И этих действий достаточно, чтобы настроить совместную работу над репозиторием. Обязательно запишите и зафиксируйте эту информацию, потому что этот файл к лекции приложен не будет.

Можете создать аккаунт на GitHub, сделать локальную копию собственного репозитория и дать другим людям доступ к этому репозиторию, чтобы они тоже направляли свои изменения. Вам также понадобится настроить работу с ветками. Ещё потребуется человек, который будет делать merge и решать, что делать при возникновении конфликтов. Но в целом работа может уже вестись.

[00.43.00]

Работа программистов

Теперь посмотрим, что происходит, когда работают программисты. Например, как это происходит в крупных компаниях или в проектах open source, то есть в проектах с открытыми исходными кодами.

В этом случае создаётся один основной аккаунт, основной репозиторий. Доступ к этому репозиторию есть только у ограниченного числа лиц. Но другие люди тоже могут поучаствовать в проекте. Вспомним, например, как Линус Торвальдс создавал Linux.

Можно создать конкретный репозиторий. Например, десяток программистов будет иметь доступ к этому репозиторию, отправлять туда изменения и выкачивать их. Но при этом 1 000 других энтузиастов тоже хотят поучаствовать в этом процессе, чтобы исправлять ошибки или добавлять какие-то незначительные улучшения. Однако давать им доступ на внесение изменений в основной репозиторий неправильно, так как это могут быть неопытные

программисты. Они могут что-то испортить, а мы не хотим, чтобы в нашем проекте сломался какой-либо элемент. Однако полностью отказываться от их помощи тоже нехорошо. Эти люди могут найти какие-то ошибки, которые прошли мимо нас, и придумать дополнительные улучшения. Поэтому нам нужен инструмент, который позволит другим людям предлагать изменения в наш проект. В Git это pull request, где request — «запрос». Таким образом, pull request — это запрос на вливание в какой-то репозиторий.

Посмотрим, как это работает. Для этого возьмём репозиторий на одном из аккаунтов GitHub, которым используется при работе со студентами. На этом аккаунте есть много репозиториев, они не так хорошо оформлены, так как не все студенты на мехмате проходят контроль версий. Но мы всё равно познакомимся с этими репозиториями.

Итак, здесь есть что-то на Java, но программирование нас не интересует. Мы будем работать с текстами. Добавить текст напрямую мы не сможем, потому что в этот аккаунт у нас нет доступа. Мы авторизованы в другом аккаунте. Но чтобы поучаствовать в этом проекте, репозитории, мы сначала должны сделать копию этого репозитория. Для этого используется кнопка **Fork**.

Например, в этом репозитории есть group_903 некого пользователя ShafigullinIK. Сделаем полную копию этого репозитория на своём аккаунте. То есть к этому аккаунту у нас доступа нет. Но если нажмём на кнопку Fork, появится такой же репозиторий, только уже на нашем аккаунте.

Здесь есть некоторые пояснения, что новый репозиторий создан на основе исходного. Но это уже наш репозиторий, к которому имеем полный доступ. И с этим репозиторием мы можем делать всё, что захотим.

Fork с английского переводится как «вилка». Соответственно, когда мы делаем Fork, появляется ответвление от изначального репозитория. Программисты тоже решили, что это похоже на вилку, поэтому назвали Fork.

Так как теперь у нас есть собственный аккаунт, заберём указанный репозиторий, скопировав его ссылку. Далее мы можем клонировать этот репозиторий в наши локальные папки. У нас есть папка Lesson 3, где уже есть один клонированный репозиторий, рядом положим ещё один. Для этого просто откроем в VS Code нужную папку и рядом с существующим репозиторием положим ещё один, клон того, что форкнули. То есть мы взяли чей-то репозиторий с какого-то другого аккаунта и сделали его полную копию на нашем аккаунте.

Далее командой git clone сделаем репозиторий — он появился. Сейчас мы находимся в папке Lesson 3, которая не считается репозиторием. Это обычная папка. Внутри лежат два других репозитория. Тот, что нам нужен, называется group_903. Перейдём внутрь папки group_903. Внутри этой папки есть какой-то проект, и мы можем помочь этому проекту.

[00.49.14]

Описание файла

На GitHub принято, чтобы у каждого проекта был файл read me, то есть файл с описанием того, что происходит внутри проекта. Создадим внутри файл. Но перед этим запомним, когда мы хотим внести или предложить изменения в чей-то проект, это всегда делается в отдельной ветке.

Создадим отдельную ветку, а в этой ветке — файл с описанием проекта. Тому, как создать новую ветку, посвящена вся вторая лекция. На семинарах вы также подробно это разобрали.

Для начала посмотрим, какие ветки существуют — есть только ветка master. Соответственно, мы можем положить рядом ветку description. У нас появляется новая ветка. Посмотрим, что она действительно появилась. Для этого введём git branch — да, она есть. Перейдём на неё, введя git checkout description. Теперь убедимся, что мы находимся на ней. Далее продолжим работу с этого места: очистим выдачу и создадим новый файл внутри папки group_903.

Опытные программисты делают это, не отрывая рук от клавиатуры. Мы же будем создавать файлы пока через указатель.

Таким образом, появляется новый файл README.md. Принято называть этот файл с использованием клавиши Caps lock, так как это общепринятый способ наименования. Теперь что-нибудь напишем. Желательно посмотреть, что происходит в этом проекте, и дать описание, но сейчас погружаться в это не будем. Просто дадим некоторое описание, это может быть совершенно любой текст: «Это какой-то проект на гитхаб-аккаунте».

Теперь надо всё закоммитить. Для этого добавим указанный файл. Введём git status, чтобы убедиться, что файл появился. Да, он появился, но пока не отслеживается. Указано также, что мы находимся на ветке description. Всё в порядке, значит, прописываем команду add, добавляем файл README.md и git commit. Добавляем некоторое сообщения, например: «Добавили описание к проекту». Всё прошло успешно.

Теперь у нас есть основная ветка, в которой происходит работа, и новая. Последней ветки в изначальном репозитории нет, но это именно то, что сейчас требуется. Мы создали дополнительную ветку, произвели в ней какие-то изменения и хотим предложить их человеку, владеющему основным репозиторием.

Для этого направим то, что сделано сейчас, в наш репозиторий. Git как раз подсказывает, что надо набрать, чтобы это сделать. Так как делаем это из соседней ветки, выполним команду git push. Всё можно не запоминать, так как Git почти всегда подсказывает, что надо делать.

Посмотрим на текущее состояние — появилась дополнительная ветка и новая кнопка (Compare & pull request) на нашем GitHub-аккаунте. Compare — сравнить, pull request — отправь запрос на вливание того, что сделано в основном репозитории. Основным считается тот, что мы взяли — ShafigullinIK/group_903.

Сейчас мы работаем в своём репозитории, а ниже указан чужой репозиторий. Сделали fork И теперь, когда мы сделали какие-то изменения, добавили новую ветку, можем предложить эти изменения изначальному хозяину репозитория. Для этого нажимаем на Compare & pull request.

GitHub проанализировал то, что мы сделали. Он говорит, что можно без проблем слить сделанное и то, что было в основном репозитории. То есть никаких конфликтов он не обнаружил и предлагает добавить некоторое описание. В первом поле напишем «Добавили описание к проекту», а во втором поле добавим информацию «Мы хотели бы помочь вашему проекту». После этого направим pull request в изначальный аккаунт.

Затем в аккаунте, копию которого мы делали, появляется новое поле. С ним будет работать уже хозяин этого аккаунта. Он посмотрит на изменения, которые мы сделали. Если произведённые изменения его устроят, он может влить эту ветку в основной репозиторий. Именно так происходит работа над проектами open source. Open source — открытый исходный код, открытый исходник.

Когда группа энтузиастов работает над большим проектом, например, пишет какую-то программу, сервис, который всем интересен, другие люди могут в этом поучаствовать. Например, предложить какие-то дополнительные изменения. Это участие осуществляется именно через систему pull request.

1. Делаем копию аккаунта, который интересен, к себе.

2. Создаём локальный репозиторий, то есть через git clone берём его на свой локальный компьютер.
3. Создаём ветку, где делаем изменения, которые хотели бы предложить.
4. Направляем свои изменения в собственный аккаунт, и появляется кнопка для отправки — pull request.

[00.56.00]

Создание кнопки pull request

Запишем все эти действия. Прикладывать этот файл к лекции не будем, поэтому зафиксируйте указанную информацию самостоятельно.

1. Делаем форк (fork) интересующего нас репозитория.
2. Делаем git clone для нашей версии этого репозитория. *Так появляется версия на нашем аккаунте, и именно эту версию мы клонируем.*
3. Создаём ветку с предлагаемыми изменениями.
4. Производим все изменения только в этой ветке.
5. Отправляем эти изменения на свой аккаунт (push).
6. В окне на GitHub появляется возможность отправить pull request.

Понятно, что эти операции с проектами open source мы пока выполнять не будем. Чтобы внести изменения в существующие проекты, требуется квалификация программиста. Особенно это необходимо при работе с большими проектами. Эти операции мы будем проводить, когда займёмся программированием напрямую.

Свои практические задания можете сдавать через pull request, это позволит наладить автоматическую проверку. Далее настроим систему автотестов, которые будут проверять ваши практические задания.

Процесс, который мы сейчас рассмотрели — реальный рабочий процесс программистов. И если вы привыкнете к нему на простых задачах, в дальнейшем вам станет проще работать в правильной культуре программистов, в их правильном инженерном подходе.

[00.59.17]

Заключение

Что узнали на уроке

1. **Изучили команду git clone.**

   Команда git clone позволяет копировать или клонировать к себе репозитории из интернета. Для этого мы обращаемся к программе Git, указываем параметр clone и после этого даём адрес того репозитория, который надо скачать. Далее в папке, где вызывается команда git clone, появляется новая папка. И уже внутри этой папки будет лежать полная копия указанного нами репозитория.

2. **Познакомились с командой git pull.**

   Эта команда позволяет скачать всё актуальное из нашего удалённого репозитория. Команда очень простая. Мы указываем программу git и параметр pull. Она позволяет получить из интернета актуальное состояние удалённого репозитория и скачать его на наш репозиторий. Обратите внимание, что pull — составная команда. Помимо выкачивания из интернета, она ещё и мержит, то есть сливает, изменения с нашими. Если возникают какие-то конфликты, то окошко будет таким же, как при конфликтах обычного git merge. То есть коменда состоит из двух частей: первая часть скачивает изменения с удалённого репозитория, а вторая — сливает эти изменения с текущим репозиторием.

3. **Рассмотрели команду git push.**

   Эта команда позволяет отправить то, что есть на нашем репозитории, на удалённый репозиторий. **Требует авторизации**. Если у нет прав на внесение изменений в репозиторий, Git не позволит это делать. На слайде команда указана в усечённом виде, она так работает, когда всё настроено. Если ещё не до конца всё настроено, Git подскажет, как сделать это правильно. Там будет параметр set upstream origin и прочие параметры, которые потребуется указать, Git расскажет об этом.

[01.01.42]

Инструкция по созданию pull request

1. Делам **fork** понравившегося репозитория.

   Смотрим на понравившийся репозиторий, куда хотим внести свой вклад, и делаем fork. В это время на нашем аккаунте возникает полная копия этого репозитория. К ней у нас уже будет свой доступ.

2. **Клонируем свою** версию репозитория, ту, которая появилась после fork.

3. Создаём **новую ветку** для внесения изменений. Именно в этой ветке **фиксируются** все изменения.

4. Направляем свою версию на **собственный GitHub-аккаунт**, так как на чужой доступа нет.

5. На сайте GitHub нажимаем на появившуюся кнопку **pull request**.

[01.02.45]

Вывод

На этом мы завершаем курс «Введение в контроль версий». Просмотрите интерактивный учебник **LearnGitBranching**, чтобы наглядно представлять, что происходит при тех или иных командах. Ждём вас на следующих лекциях, до встречи.