МИНИСТЕРСТВО НАУКИ и высшего образования РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

филиал «РКТ» МАИ в г. Химки Московской области

**Специальность 09.02.03 «Программирование в компьютерных системах»**

**ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

**ПМ.02«Разработка и администрирование баз данных**»

**Студент**

**Группы МП-32\_\_\_\_\_\_\_\_\_Дударев Д.А.\_\_ / ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )**

**Руководитель**

**практики от организации \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)**

**Руководитель**

**практики от филиала Шумаев А.Ю. / ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )**

**2020г.**

**ПРОГРАММА ПРОИЗВОДСТВЕННОЙ ПРАКТИКИ**

По специальности 09.02.03 «Программирование в компьютерных системах»

|  |  |  |
| --- | --- | --- |
| Дата | Наименование выполняемых работ | Количество дней практики |
| 11.06.20 | Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. | 4 |
| 12.06.20 | Описание структуры сети, в которой может функционировать разработанная база данных. | 4 |
| 13.06.20 | Описание сетевого оборудования необходимого для работы с базой данных. | 4 |
| 15.06.20 | Описание программного обеспечения необходимого для работы с базой данных по сети. | 4 |
| 16.06.20-18.06.20 | Разработка клиентских интерфейсов для клиент-серверных приложений. | 16 |
| 20.06.20 | Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. | 4 |
| 22.06.20-24.06.20 | Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. | 16 |
| 25.06.20-01.07.20 | Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. | 50 |
| 02.07.20-03.07.20 | Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. | 12 |
| 03.07.20-04.07.20 | Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. | 12 |
| 06.07.20 | Составление отчёта по практике | 6 |
| 07.07.20 | Подготовка к защите отчёта по практике. | 6 |
| 08.07.20 | Итоговая аттестация по производственной практике - зачёт | 6 |

Руководитель практики от филиала «РКТ» МАИ преподаватель

Шумаев А.Ю.

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)*

**АТТЕСТАЦИОННЫЙ ЛИСТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ**

студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Дударев Дмитрий Александрович\_\_\_\_\_\_\_\_\_\_\_

*ФИО*

обучающийся (аяся) на 3-ем курсе по специальности СПО 09.02.03 «Программирование в компьютерных системах»

успешно прошел(ла) производственную практику по профессиональному модулю   
**ПМ.02 «Разработка и администрирование баз данных»**

в объеме 144 часа с 11.06.2020г. по 08.07.2020г.

в организации филиала «РКТ» МАИ

Виды и качество выполнения работ

|  |  |  |
| --- | --- | --- |
| Виды работ, выполненных обучающимся(ейся) во время практики | Объем работ | Качество выполнения работ в соответствии с технологией и (или) требованиями организации, в которой проходила практика |
| Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. | 4 |  |
| Описание структуры сети, в которой может функционировать разработанная база данных. | 4 |  |
| Описание сетевого оборудования необходимого для работы с базой данных. | 4 |  |
| Описание программного обеспечения необходимого для работы с базой данных по сети. | 4 |  |
| Разработка клиентских интерфейсов для клиент-серверных приложений. | 16 |  |
| Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. | 4 |  |
| Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. | 16 |  |
| Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. | 50 |  |
| Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. | 12 |  |
| Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. | 12 |  |
| Составление отчёта по практике | 6 |  |
| Подготовка к защите отчёта по практике. | 6 |  |
| Итоговая аттестация по производственной практике - зачёт | 6 |  |

Руководитель практики от филиала «РКТ» МАИ Шумаев А.Ю.

|  |  |  |
| --- | --- | --- |
| Дата\_\_\_\_\_\_\_\_\_\_\_\_\_ | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *подпись* | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  *Расшифровка подписи* |

**ДНЕВНИК** **ПРОХОЖДЕНИЯ ПРАКТИКИ**

|  |  |  |
| --- | --- | --- |
| Дата | Наименование выполняемых работ | Подпись  руководителя |
| 11.06.20 | Ознакомление с программой практики. Прохождение инструктажа по технике безопасности и охране труда, изучение внутреннего распорядка организации и правил работы. Разработка задания на производственную практику. |  |
| 12.06.20 | Описание структуры сети, в которой может функционировать разработанная база данных. |  |
| 13.06.20 | Описание сетевого оборудования необходимого для работы с базой данных. |  |
| 15.06.20 | Описание программного обеспечения необходимого для работы с базой данных по сети. |  |
| 16.06.20-18.06.20 | Разработка клиентских интерфейсов для клиент-серверных приложений. |  |
| 20.06.20 | Создание инфокоммуникационной системы.  Выполнение сетевых настроек для взаимодействия с СУБД. |  |
| 22.06.20-24.06.20 | Создание концептуальной модели данных.  Создание логической модели данных.  Создание физической модели данных.  Описание механизмов обеспечения целостности базы данных. |  |
| 25.06.20-01.07.20 | Описание СУБД представленной базы данных и ее возможностей. Внесение различных данных в базу данных.  Структурирование запросов базы данных.  Определение методов создания хранимых процедур и триггеров. Создание хранимых процедур и триггеров базы данных. |  |
| 02.07.20-03.07.20 | Определение способов управления правами пользователей.  Описание распределения прав пользователей и управления ими в базе данных. |  |
| 03.07.20-04.07.20 | Определение методов создания и синхронизации реплик базы данных. Описание существующих механизмов репликации в базе данных. Определение методов создания резервных копий базы данных. Описание существующих механизмов резервного копирования в базе данных. |  |
| 06.07.20 | Составление отчёта по практике |  |
| 07.07.20 | Подготовка к защите отчёта по практике. |  |
| 08.07.20 | Итоговая аттестация по производственной практике - зачёт |  |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 7](#_Toc53924956)

[1. GITHUB 8](#_Toc53924957)

[1.1. О системе контроля версий 8](#_Toc53924958)

[1.2. Локальные системы контроля версий 8](#_Toc53924959)

[1.3. Централизованные системы контроля версий 9](#_Toc53924960)

[1.4. Распределённые системы контроля версий 10](#_Toc53924961)

[2. ОСНОВНЫЕ ОПЕРАЦИИ GITHUB 12](#_Toc53924962)

[2.1. Репозиторий 12](#_Toc53924965)

[2.2. Лес git’а 12](#_Toc53924966)

[2.3. Ветвление 14](#_Toc53924967)

[2.4. Создание новой ветки 14](#_Toc53924968)

[2.5. Переключение между ветками 14](#_Toc53924969)

[2.6. Слияние веток 14](#_Toc53924970)

[2.7. Фичи git’а 15](#_Toc53924971)

[3. Jquery 16](#_Toc53924972)

[3.1. Введение в Jquery 16](#_Toc53924973)

[3.2. Основные понятия 19](#_Toc53924974)

[3.3. Интеграция с другими продуктами 20](#_Toc53924975)

[Заключение 21](#_Toc53924976)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 22](#_Toc53924977)

[ПРИЛОЖЕНИЯ 23](#_Toc53924978)

# ВВЕДЕНИЕ

Практическая выполнена в соответствии с рабочей программой ПМ.02 «Разработка и администрирование баз данных». В работе представлены основные этапы анализа и разработки предметной области «Разработка и администрирование базы данных», навыки работы с GitHub и Jquery.

База данных играет важную роль в большинстве предметных областей. Благодаря динамической природе автоматизированных отраслей сейчас приложения требуют некоторых механизмов хранения, доступа и изменения данных. Поскольку важность баз данных стремительно растёт, реляционные системы управления базами данных набирают свою популярность.

В качестве программного обеспечения для создания интерфейса выбран редактор Notepad++. А для написания кода выбраны: язык гипертекстовой разметки HTML, язык таблиц стилей CSS и для работы с базами данных PHP. Использован веб-интерфейс phpmyadmin и СУБД SQLite.

В процессе выполнения проекта должен быть освоен ряд следующих профессиональных компетенций: ознакомление со структурой Git, реализация базы данных в конкретной системе управления базами данных. Получение знаний и опыта в области в технологии Jquery. Также мною будут решаться вопросы администрирования базы данных и реализовываться методы и технологии её защиты.

# GITHUB

Эта глава о том, как начать работу с Git. Вначале изучим основы систем контроля версий, затем перейдём к тому, как запустить Git на вашей ОС и окончательно настроить для работы. В конце главы вы уже будете знать, что такое Git и почему им следует пользоваться, а также получите окончательно настроенную для работы систему.

* 1. О системе контроля версий

Что такое «система контроля версий» и почему это важно? Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Для контроля версий файлов в этой книге в качестве примера будет использоваться исходный код программного обеспечения, хотя на самом деле вы можете использовать контроль версий практически для любых типов файлов.

Если вы графический или web-дизайнер и хотите сохранить каждую версию изображения или макета (скорее всего, захотите), система контроля версий (далее СКВ) — как раз то, что нужно. Она позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда и многое другое. Использование СКВ также значит в целом, что, если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить. В дополнение ко всему вы получите всё это без каких-либо дополнительных усилий.

* 1. Локальные системы контроля версий

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.



Рисунок 1. Локальный контроль версий.

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

* 1. Централизованные системы контроля версий

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.



Рисунок 2. Централизованный контроль версий.

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы: когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

* 1. Распределённые системы контроля версий

Здесь в игру вступают распределённые системы контроля версий (РСКВ). В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.



Рисунок 3. Распределённый контроль версий.

Более того, многие РСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

## ОСНОВНЫЕ ОПЕРАЦИИ GITHUB

Начнем, естественно, с загрузки. Надеемся, какая у вас операционная система, вы знаете. И сразу предупредим новичков: не путайте git и GitHub — это разные вещи. Нас интересует именно git, а GitHub (или ему подобные сервисы вроде Bitbucket или GitLab) — это по сути хостинг для проектов, использующих git.

2. 1. Репозиторий

Итак, вот у вас уже есть git. Теперь нужно создать хранилище версий для него. Запомните, это хранилище называется репозиторий (англ. repository) — при случае можете вставить где-нибудь это словечко. В зависимости от того, какая у вас оболочка, соответствующей командой создайте новую директорию, откройте ее (в командной строке, она же оболочка, а не проводником или чем-то подобным) и выполните:

git init

Все, локальный репозиторий в этой папке создан. То, что здесь сейчас хранится, будет бекапом, поэтому, чтобы его не испортить, создадим рабочую копию (англ. check out) локальной версии:

git clone [url]

Где [url] — это путь до клонируемого репозитория. Мы разбираем сейчас случай, когда вы создаете рабочую копию собственного репозитория, поэтому в качестве [url] здесь вам нужно указать путь до директории, для которой мы выполняли git init.

Но если вы крутой чувак и уже работаете с удаленным сервером, то вот такая команда будет для вас в самый раз:

git clone username@host:/path/to/repository

* 1. Лес git’а

Немного теории. Git в своей работе управляет тремя структурами, которые называются деревьями. Первое — это рабочая директория, в ней хранятся файлы, с которыми вы прямо сейчас работаете. Ну, она ж рабочая, логично. Второе — это Index, этакий чек-поинт, который позволяет вам вносить изменения и ничего не портить. А третье — это HEAD, который указывает на последний сделанный вами коммит. (Чтобы вы не запутались в терминологии: коммит (англ. commit) — это сохранение состояния проекта в репозиторий. Короче, считайте, новая версия.)

Так вот, чтобы вы не заблудились в этих трех соснах, запомните две крутые команды: add и commit. Они позволят вашей работе спокойно бродить по git’у, сохраняясь, куда надо. Если вы придумали что-то гениальное и тут же внесли изменение в рабочую копию проекта, то не спешите сразу коммитить! Сначала испытайте в Index’е, для этого выполните:

git add [имя\_файла]

если вы внесли изменение только в один файл, или

git add \*

если вы хорошо потрудились поменяли сразу кучу исходников. Изменения положительны? Хорошо потестили? Тогда скорее коммитить:

git commit -m "Commit message"

Вы, конечно же, плохой парень и не оставляете комментариев в коде. Но git — другое дело. Не ленитесь оставлять поясняющие сообщения: будьте уверены, вам хватит других проблем, кроме как разбираться, что же поменялось в этом коммите по сравнению с прошлой версией. Поясняющая картинка:

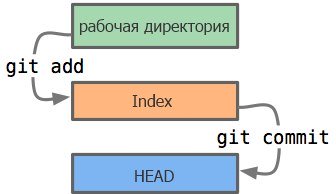
[](https://s3.tproger.ru/uploads/2016/06/EXcwUk86PWs.jpg)

Рисунок 4. Git add и commit

Теперь файл(-ы) прочно обосновались в HEAD вашей рабочей локальной копии. Оттуда их не выгнать, но в вашем удаленном репозитории их все еще нет. Давайте сунем их еще и туда! Используйте:

git push origin master

Только вместо master напишите название нужной ветки. Ах да, вы же еще не знаете, что такое ветки. Ну ладно, пока что запомните это место, а когда прочитаете про ветвление, вернетесь сюда.

Ах да, для крутых чуваков, работающих с серверами (как раз тут уместно говорить про GitHub, например), команда будет такой:

git remote add origin [сервер]

* 1. Ветвление

По-английски эта штука зовется branching — лучше как следует вникните в этот вопрос и почитайте про ветвление подробнее, я вас с ним только познакомлю. Ветвление используется для одновременной и независимой разработки разных фич (ну, или накопления большего количества багов, ведь исходного кода становится больше). Основной веткой является master — она появляется при создании репозитория. Другие ветки — это песочницы, когда достаточно в них наиграетесь, слейте в единое целое в master. Сейчас поясню, как это делается.

* 1. Создание новой ветки

Вот вы решили проработать какую-нибудь новую фичу. Создайте для нее новую ветку:

git checkout -b [новая\_ветка]

Ах да, фантазия-то у вас, наверное, работает на полную катушку, ну да поумерьте её в деле именования веток: назвать ветку можно только именем, допустимым для переменной в вашем любимом языке.

* 1. Переключение между ветками

Надо сделать перерыв в работе с этой фичей и переключиться на другую ветку? Используйте (если работаете с локальным репозиторием, то указывать его имя не обязательно):

git checkout [репозиторий]/[ветка]

Ну, а если вы уже совсем не хотите с ней работать, то удалите ее совсем:

git branch -d [ветка]

Со своей веткой вы можете творить любые непотребства: ее никто не увидит, пока вы сами ее не пропушите в удаленный репозиторий командой:

git push origin [ветка]

* 1. Слияние веток

Чтобы слить ветку в ту, с которой вы сейчас работаете, используйте:

git merge [ветка]

Но, понятное дело, это все приводит к конфликтам. И это реально проблема. Так что попробуйте исправлять все ручками прямо в директории с репозиторием. Только потом не забудьте пометить, что вы их «слили»:

git add [имя\_файла]

Кстати, ветки можно сравнить:

git diff [одна\_ветка] [другая\_ветка]

Так, теперь приступим к более решительным действиям. Будем обновлять свой репозиторий в соответствии с самым свежим коммитом. Сделать это очень просто (а вот вернуть обратно не очень, поэтому трижды подумайте, прежде чем совершать эту ужасную ошибку):

git pull

Я, конечно, понимаю, что вы слишком круты, чтобы оставлять какие-либо пометки на будущее — все держите в голове — но все-таки рекомендую вам оставлять тэги. И это не моя выдумка, так делают многие:

git tag [tag] [первые\_десять\_символов\_соответствующего\_коммита]

Вы не знаете, какие первые символы у имени нужного коммита? Не беда, смотрите в историю репозитория — его лог:

git log

Там есть куча разных параметров для использования этой полезной штуковины, ну да погуглите их сами. Ах да, кстати, мы уже писали как-то про то как сделать git log более информативным.

* 1. Фичи git’а

Если вы ленивый, и вам не охота по-трупрогерски все писать в оболочке своей ОСи, то можете использовать GUI git’а:

gitk

В источнике найдете еще кучу других GUI-шек.  
Если вам стандартный вывод git’а кажется скучным, раскрасьте его:

git config color.ui true

Ну, и есть еще такая штука — интерактивное индексирование. Когда у вас будет уже достаточно большой проект, то ужать представление index’а в log’е можно будет так:

git add -i

# Jquery

jQuery — это замечательный JavaScript Framework, который подкупает своей простотой в понимании и удобством в использовании.  Он содержит [набор функций JavaScript](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_JavaScript), фокусирующийся на взаимодействии [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript) и [HTML](https://ru.wikipedia.org/wiki/HTML). Библиотека jQuery помогает легко получать доступ к любому элементу [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model), обращаться к атрибутам и содержимому элементов [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model), манипулировать ими. Также библиотека jQuery предоставляет удобный [API](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) для работы с [AJAX](https://ru.wikipedia.org/wiki/AJAX). Разработка jQuery ведётся командой добровольцев на пожертвования.

DOM (Document Object Model — «объектная модель документа») — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

API - Интерфейс прикладного программирования (иногда интерфейс программирования приложений) (Application Programming Interface, API) — набор готовых классов, функций, структур и констант, предоставляемых приложением (библиотекой) для использования во внешних программных продуктах   
Подключение библиотеки jQuery происходит так же, как и подключение любой другой

JavaScript библиотеки.

<head> <script type="text/javascript” src="путь\_к/jquery.js"> </script> </head>

# Введение в Jquery

Процесс изучения JavaScript всегда был болезненным для разработчиков, и по мере того, как всеобщее чувство недовольства нарастало, некоторые из них взялись создавать библиотеки JavaScript, которые часто называют JavaScript - фреймворками.  
Библиотеки этого типа призваны упростить использование языка JavaScript и сделать его более доступным как для новичков, так и для разработчиков со стажем за счет предоставления простых в применении функций, облегчающих решение повседневных задач.

Существует довольно много библиотек JavaScript. Самыми популярными из тех, которые используются разработчиками в настоящее время, являются библиотеки Prototype, MooTools, а также jQuery, которая в настоящее время является самой популярной.

Основная часть jQuery — библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими. Также библиотека jQuery предоставляет удобный API для работы с AJAX.

Основная цель создания jQuery - возможность закодировать многоразовые куски кода, которые позволят упростить JavaScript и использовать их так, чтобы не беспокоиться о кросс-браузерных вопросах.

Преимущества:

1.получение доступа к любому элементу

2.обращение к атрибутам и содержимому элементов DOM и их манипуляция

3.синтаксис селекотора элементов схож с CSS

4.предоставление удобного API для работы с AJAX

5.возможность анимации полученных элементов на странице

6.небольшой размер файла

7.чрезвычайно простой синтаксис

8.возможность объединения последовательно вызываемых методов в цепочки

9.простая архитектура подключаемых модулей, расширяющих базовые возможности библиотеки

10. огромное сетевое сообщество пользователей

11. полезные расширения, такие как jQuery UI, предоставляющие дополнительную функциональность.

Работу с jQuery можно разделить на 2 типа:

Получение jQuery-объекта с помощью функции $(). Например, передав в неё CSS-селектор, можно получить jQuery-объект всех элементов HTML, попадающих подкритерий и далее работать с ними с помощью различных методов jQuery-объекта. В случае, если метод не должен возвращать какого-либо значения, он возвращает ссылку на jQuery объект, что позволяет вести цепочку вызовов методов согласно концепции текучего интерфейса.

Вызов глобальных методов у объекта $, например, удобных итераторов по массиву.  
Типичный пример манипуляции сразу несколькими узлами DOM заключается в вызове $ функции со строкой селектора CSS, что возвращает объект jQuery, содержащий некоторое количество элементов HTML-страницы. Эти элементы затем обрабатываются методами jQuery.

Например:  
$("div.test").add("p.quote").addClass("blue").slideDown("slow");  
находит все элементы div с классом test, а также все элементы p с классом quote, и затем добавляет им всем класс blue и визуально плавно спускает вниз. Здесь методы add, addClass и slideDown возвращают ссылку на исходный объект $("div.test"), поэтому возможно вести такую цепочку.  
Методы, начинающиеся с $., удобно применять для обработки глобальных объектов. Например:  
$.each([1,2,3], function() {  
document.write (this + 1);  
});  
добавит на страницу 234.

Подключение библиотеки jQuery происходит так же, как и подключение любой другой JavaScript библиотеки:  
<head>  
<script type="text/javascript” src="путь\_к/jquery.js"></script>  
</head>  
Вызывается работа библиотеки символом $:  
$(function() {  
// Тут код скрипта  
})

В редких случаях вызывается командой jQuery, это бывает в тех случаях, когда браузер не может обработать или понять символ $:  
jQuery(function($) {  
// Тут код скрипта, где в $ будет jQuery  
})

Пример работы jQuery с Ajax:  
$.ajax({  
dataType : "json",  
url: "some.php",  
data: {name: 'John'},  
complete:cbComplete,  
error: cbError,  
success: cbSuccess  
});  
error — если произошла ошибка  
success — если ошибок невозникло  
complete — срабатывает по окончанию запроса

* 1. Основные понятия

DOM (Document Object Model — «объектная модель документа») — это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.  
API - Интерфейс прикладного программирования (иногда интерфейс программирования приложений) (Application Programming Interface, API) — набор готовых классов, функций, структур и констант, предоставляемых приложением (библиотекой) для использования во внешних программных продуктах

# Интеграция с другими продуктами

28 сентября 2008 года на официальном блоге jQuery сообщили о том, что компании Microsoft и Nokia собираются сотрудничать с группой разработчиков. Компания Microsoft собирается интегрировать в свой продукт ASP.NET листинги кода и примеры jQuery, а компания Nokia собирается интегрировать jQuery для своих мобильных виджетов.

# Заключение

Мы рассмотрели большинство основных способов настройки клиента и сервера Git с тем, чтобы он был максимально соответствовал вашим рабочим процессам и проектам.

Мы узнали о всевозможных настройках, атрибутах файлов и о перехватчиках событий, а также рассмотрели пример настройки сервера с соблюдением политики. Теперь вам должно быть по плечу заставить Git подстроиться под практически любой тип рабочего процесса, который только можно вообразить.

Библиотека jQuery, детище Джона Резига, была впервые представлена общественности на компьютерной конференции BarCamp в Нью-Йорке в начале 2006 года. Как отметил на своем сайте сам Резиг, к созданию jQuery его подтолкнули неудовлетворенность существовавшими на то время библиотеками и осознание того факта, что их можно значительно улучшить, уменьшив объем "синтаксической шелухи" и введя специфические элементы управления для часто выполняемых операций.  
Библиотека jQuery сразу же завоевала признание сообщества разработчиков и быстро распространилась среди его членов. В процесс усовершенствования библиотеки включились другие разработчики, итогом чего стал выпуск первой устойчивой версии 1.0, состоявшийся 26 августа 2006 года.

С тех пор библиотека jQuery непрерывно улучшается (на данный момент уже выпущена версия jQuery 2.0), и в настоящее время она снабжена множеством дополнений (подключаемых модулей), предложенных сообществом разработчиков.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Github: [Электронный ресурс]/ https://tproger.ru/translations/difference-between-git-and-github - статья в интернете.
2. SQLite: [Электронный ресурс]/ https://ru.bmstu.wiki/SQLite - статья в интернете.
3. Руководство SQLite: [Электронный ресурс]/ https://proglib.io/p/sqlite-tutorial - статья в интернете.

# ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

( function( global, factory ) {

"use strict";

if ( typeof module === "object" && typeof module.exports === "object" ) {

module.exports = global.document ?

factory( global, true ) :

function( w ) {

if ( !w.document ) {

throw new Error( "jQuery requires a window with a document" );

}

return factory( w );

};

} else {

factory( global );

}

} )( typeof window !== "undefined" ? window : this, function( window, noGlobal ) {

var arr = [];

var getProto = Object.getPrototypeOf;

var slice = arr.slice;

var flat = arr.flat ? function( array ) {

return arr.flat.call( array );

} : function( array ) {

return arr.concat.apply( [], array );

};

var push = arr.push;

var indexOf = arr.indexOf;

var class2type = {};

var toString = class2type.toString;

var hasOwn = class2type.hasOwnProperty;

var fnToString = hasOwn.toString;

var ObjectFunctionString = fnToString.call( Object );

var support = {};

var isFunction = function isFunction( obj ) {

return typeof obj === "function" && typeof obj.nodeType !== "number";

};

var isWindow = function isWindow( obj ) {

return obj != null && obj === obj.window;

};

var document = window.document;

var preservedScriptAttributes = {

type: true,

src: true,

nonce: true,

noModule: true

};

function DOMEval( code, node, doc ) {

doc = doc || document;

var i, val,

script = doc.createElement( "script" );

script.text = code;

if ( node ) {

for ( i in preservedScriptAttributes ) {

val = node[ i ] || node.getAttribute && node.getAttribute( i );

if ( val ) {

script.setAttribute( i, val );

}

}

}

doc.head.appendChild( script ).parentNode.removeChild( script );

}

function toType( obj ) {

if ( obj == null ) {

return obj + "";

}

return typeof obj === "object" || typeof obj === "function" ?

class2type[ toString.call( obj ) ] || "object" :

typeof obj;

}

var

version = "3.5.1",

jQuery = function( selector, context ) {

return new jQuery.fn.init( selector, context );

};

jQuery.fn = jQuery.prototype = {

jquery: version,

constructor: jQuery,

length: 0,

toArray: function() {

return slice.call( this );

},

get: function( num ) {

if ( num == null ) {

return slice.call( this );

}

return num < 0 ? this[ num + this.length ] : this[ num ];

},

pushStack: function( elems ) {

var ret = jQuery.merge( this.constructor(), elems );

ret.prevObject = this;

return ret;

},

each: function( callback ) {

return jQuery.each( this, callback );

},

map: function( callback ) {

return this.pushStack( jQuery.map( this, function( elem, i ) {

return callback.call( elem, i, elem );

} ) );

},

slice: function() {

return this.pushStack( slice.apply( this, arguments ) );

},

first: function() {

return this.eq( 0 );

},

last: function() {

return this.eq( -1 );

},

even: function() {

return this.pushStack( jQuery.grep( this, function( \_elem, i ) {

return ( i + 1 ) % 2;

} ) );

},

odd: function() {

return this.pushStack( jQuery.grep( this, function( \_elem, i ) {

return i % 2;

} ) );

},

eq: function( i ) {

var len = this.length,

j = +i + ( i < 0 ? len : 0 );

return this.pushStack( j >= 0 && j < len ? [ this[ j ] ] : [] );

},

end: function() {

return this.prevObject || this.constructor();

},

push: push,

sort: arr.sort,

splice: arr.splice

};

jQuery.extend = jQuery.fn.extend = function() {

var options, name, src, copy, copyIsArray, clone,

target = arguments[ 0 ] || {},

i = 1,

length = arguments.length,

deep = false;

if ( typeof target === "boolean" ) {

deep = target;

target = arguments[ i ] || {};

i++;

}

if ( typeof target !== "object" && !isFunction( target ) ) {

target = {};

}

if ( i === length ) {

target = this;

i--;

}

for ( ; i < length; i++ ) {

if ( ( options = arguments[ i ] ) != null ) {

for ( name in options ) {

copy = options[ name ];

if ( name === "\_\_proto\_\_" || target === copy ) {

continue;

}

if ( deep && copy && ( jQuery.isPlainObject( copy ) ||

( copyIsArray = Array.isArray( copy ) ) ) ) {

src = target[ name ];

if ( copyIsArray && !Array.isArray( src ) ) {

clone = [];

} else if ( !copyIsArray && !jQuery.isPlainObject( src ) ) {

clone = {};

} else {

clone = src;

}

copyIsArray = false;

target[ name ] = jQuery.extend( deep, clone, copy );

} else if ( copy !== undefined ) {

target[ name ] = copy;

}

}

}

}

return target;

};

jQuery.extend( {

expando: "jQuery" + ( version + Math.random() ).replace( /\D/g, "" ),

isReady: true,

error: function( msg ) {

throw new Error( msg );

},

noop: function() {},

isPlainObject: function( obj ) {

var proto, Ctor;

if ( !obj || toString.call( obj ) !== "[object Object]" ) {

return false;

}

proto = getProto( obj );

if ( !proto ) {

return true;

}

Ctor = hasOwn.call( proto, "constructor" ) && proto.constructor;

return typeof Ctor === "function" && fnToString.call( Ctor ) === ObjectFunctionString;

},

isEmptyObject: function( obj ) {

var name;

for ( name in obj ) {

return false;

}

return true;

},

globalEval: function( code, options, doc ) {

DOMEval( code, { nonce: options && options.nonce }, doc );

},

each: function( obj, callback ) {

var length, i = 0;

if ( isArrayLike( obj ) ) {

length = obj.length;

for ( ; i < length; i++ ) {

if ( callback.call( obj[ i ], i, obj[ i ] ) === false ) {

break;

}

}

} else {

for ( i in obj ) {

if ( callback.call( obj[ i ], i, obj[ i ] ) === false ) {

break;

}

}

}

return obj;

},

makeArray: function( arr, results ) {

var ret = results || [];

if ( arr != null ) {

if ( isArrayLike( Object( arr ) ) ) {

jQuery.merge( ret,

typeof arr === "string" ?

[ arr ] : arr

);

} else {

push.call( ret, arr );

}

}