

МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»



Направление подготовки/специальность
09.03.01 Информатика и вычислительная техника

направленность (профиль)/специализация
«Технологии разработки программного обеспечения»

Выпускная квалификационная работа

Разработка методики работы с системой контроля версий
разрабатываемого продукта

Обучающегося 4 курса
очной формы обучения
Елистратова Ивана Владимировича

Руководитель выпускной квалификационной
работы:

Ученая степень *(при наличии)*, ученое звание
(при наличии), должность
Ф. И. О. *(указывается в именительном
падеже)*

Рецензент:

Ученая степень *(при наличии)*, ученое звание
(при наличии), должность
Ф. И. О. *(указывается в именительном
падеже)*

Санкт-Петербург
2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

ГЛАВА 1. Архитектура, принципы реализации и работы современных приложений

1 Функции, задачи и принципы реализации современных приложений в цифровом обществе и инфраструктуре

1.2 Progressive web application (PWA) технология разработки приложений

1.3 Multi-page application (MPA) технология разработки приложений

1.4 Single-page application (SPA) технология разработки приложений

1.5 Проблемы и трудности разработки Single-page application приложений

2 Архитектура, технологии, шаблоны, принципы и проблемы реализации приложений на основе Single-page application (SPA) технологий в условиях цифровизации общества

2.1 Система организации работы при разработке Single-page application приложений

2.2 Принципы, функции и архитектура систем контроля версий Структура и функции систем контроля версий при разработке Single-page application приложений

2.3 Принципы, функции и архитектура систем контроля версий

2.4 Централизованные системы контроля версий

2.5 Распределённые системы контроля версий

ВЫВОД ПО ГЛАВЕ 1

ГЛАВА 2. Разработка методики работы с системой контроля версий для разработки Single-page application приложений

1 Методика работы с СКВ для новостного Single-page application(SPA) приложения

1.1 Методика работы с СКВ для Single-page application(SPA) приложения через консоль.

2. Особенности и принципы использования Single-page application приложений

ВЫВОД ПО ГЛАВЕ 2

ЗАКЛЮЧЕНИЕ

СПИСОК ЛИТЕРАТУРЫ

Введение

Мы живем в очень интересное время, когда на наших глазах происходят масштабные изменения технологий, когда разрабатываются проекты, которые еще до недавнего времени казались невозможными, мы живем во времени начала четвертой промышленной революции. Отличительной особенностью этой революции будет то, что появление новых технологий будет происходить с огромной скоростью и сопровождаться сильнейшей конкуренцией.

Клаус Шваб считает, что : ”Четвёртая промышленная революция окажет кардинальное влияние на всю структуру мировой экономики, и если мы хотим быть среди ее лидеров, мы должны понимать, в каком направлении будет происходить технологическое развитие в ближайшие годы, и какие прорывные инновации ожидают нас в будущем”. Мы можем наблюдать кардинальные изменения во всех отраслях жизни, появление новых бизнес-моделей, переоценку традиционных ценностей, потребления и многого другого. Социальная сфера также не останется незатронутой, уже сейчас мы можем наблюдать изменения в коммуникациях между людьми, самовыражении, получении информации и остальном.

Аналогичные преобразования происходят на уровне правительств и государственных учреждений, в частности, в сфере образования, здравоохранения и транспорта. Кроме того, новые способы использования технологий для изменения нашего поведения, а также существующие системы производства и потребления открывают возможности для восстановления и сохранения окружающей среды.

Тенденции развития Российской Федерации также направлены на развитие информационных технологий. Исходя из документов “Распоряжение Правительства Российской Федерации от 01.09.2021 № 2419-р” и “Указ Президента Российской Федерации от 07.05.2018 г. № 204” правительство нацелено на сотрудничество между Правительством Российской Федерации и Всемирным экономическим форумом по вопросу создания и функционирования Аффилированного центра четвертой промышленной революции. Исходя из поставленных в указе целей и задач следует, что основные акценты сделаны на внедрение отечественного программного обеспечения, подготовки высококвалифицированных кадров, обеспечение информационной безопасности. На сегодняшний день выполнение поставленных задач

осложняется условиями коронавирусной пандемии, в этот период были применены различные технологии дистанционного обучения, работы и разработки.

ГЛАВА 1. Архитектура, принципы реализации и работы современных приложений.

1. Функции, задачи и принципы реализации современных приложений в цифровом обществе и инфраструктуре.

В данном параграфе проведен анализ основных принципов разработки приложений. С этой целью были изучены различные источники, были просмотрены лекции ИТ-специалистов в сфере разработки приложений.

Анализ литературы[какой] позволил сделать вывод о том, что зачастую разработка ведется не одним человеком, а командой разработчиков, каждый из них выполняет свою задачу, у каждого должен быть доступ к проекту, при этом все варианты изменений сохраняются отдельно и на сервере, а не на локальном компьютере. Если же несколько изменений затрагивают один и тот же фрагмент документа, то система предложит выбрать нужный вариант.

Заказчикам же нужно, чтобы приложение всё время функционировало и приносило доход, при использовании систем контроля версий разработчик вносит свои изменения в отдельную ветку и эти изменения принимаются в силу только при условии что приложение продолжает нормально функционировать.

В целом работа над созданием программного продукта идет по следующим этапам:

- аналитика;
- техническое задание;
- проектирование и дизайн;
- разработка;
- тестирование и стабилизация;
- запуск приложения;
- поддержка;

Этап 1. Аналитика

В первую очередь заказчик обозначает основную идею приложения, ставит задачи, которые должен решать сервис, по этим исходным производится сбор аналитики, анализ рынка, рассмотрение уже существующих решений, анализ деятельности конкурентов и поведение покупателей. Итогом первого этапа должны быть готовые референсы по функционалу и дизайну.

Этап 2. Техническое задание

На данном этапе необходимо составить техническое задание, то есть подробное описание функциональности и дизайна приложения. Также необходимо иметь представление о пользовательском опыте потенциальных клиентов. Нужно иметь четкое представление о том как должно выглядеть приложение, что оно должно уметь и как работать. Результатом второго этапа должно быть четкое техническое задание, в котором будет отражен перечень функций приложения, требования к интерфейсу, безопасности, производительности и другие нефункциональные требования, а также смету проекта.

Благодаря грамотно составленному техническому заданию команда разработчиков четко понимает какое приложение должно получиться в итоге и имеет возможность поэтапно реализовывать проект.

Этап 3. Проектирование и дизайн

На этом этапе проектируется и создается дизайн, который будет работать на изначальную концепции приложения, а именно:

- Дизайн концепция
- Набор компонентов интерфейса
- Дизайн макеты
- Интерактивные прототипы

Этап 4. Разработка

Задача данного этапа - реализация продукта. По заранее выбранной методологии программирования начинается создание и сборка проекта. Основная задача разработки - чтобы все элементы исправно выполняли свои функции и сочетались между собой. Речь идет о кнопках, иконках, разделах меню, изображениях и текстовых данных. Разработчики превращают эти элементы в функционирующую интерактивную модель.

Этап 5. Тестирование и стабилизация

Этот этап необходим для проверки функциональности софта, логичности его работы, удобства использования. Ошибки, выявленные на этом этапе разработки, быстро исправляются, чтобы пользователь не

потерял интерес к проекту и не ушел к конкуренту. За счет тестирования повышается качество работы приложения.

Тестирование бывает автоматизированным или ручным. Первый вид подразумевает настройку определенного списка кейсов, которые будут автоматически тестировать приложение. Второй вид предполагает применение исследовательских методов мониторинга. То есть проверяется соответствие сервиса ожиданиям и требованиям пользователя.

Этап 6. Запуск приложения

Прежде чем приложение станет доступным для пользователей, оно проходит модерацию. Длительность проверки зависит от многих факторов: категории приложения, его истории, разработчика и загрузки команды модерации. Проект разворачивается на финальном сервере. Для обеспечения бесперебойной работы приложения, настраивают систему мониторинга.

Этап 7. Поддержка

Продукт должен жить и меняться вместе со средой, отвечать на угрозы, приспосабливаться к изменениям устройств и операционных систем, реагировать на отклики пользователей о его удобстве и функциональности. Именно для этого необходима техподдержка приложения разработчиками и бизнес-аналитиками.

Веб-приложение - это сайт с элементами интерактива. Они позволяют пользователям взаимодействовать: нажимать кнопки, заполнять формы, запрашивать прайс, совершать покупки. Почтовые клиенты, соцсети, поисковики, интернет-магазины, программы для управления проектами - это всё примеры таких приложений.

Один из наиболее важных шагов, которые нужно предпринять, когда дело доходит до разработки приложения, - это выбрать правильный стек технологий. Этот вопрос значительно влияет на успех проекта, как в краткосрочной, так и долгосрочной перспективе.

Стек технологий для разработки веб-приложений влияет на стоимость проекта, сроки выпуска продукта на рынок, безопасность и масштабируемость приложения. Грамотно подобранный стек позволяет получить стабильный, безопасный и удобный в обслуживании продукт.

Рассмотрим, что такое стек и какие технологии используются для создания веб-приложений.

С точки зрения архитектуры веб-приложения состоят из двух частей: клиентской и серверной. Клиентская часть также называется фронтэнд. По сути это то, что пользователи видят на экране устройства. Серверная часть, или бэкенд — программно-аппаратная часть сервиса. Это набор средств, которые реализуют логику приложения. Выделяют три вида веб-приложений, которые определяют подход к разработке.



Взаимодействие клиентской и серверной частей веб-приложения

1.2 Progressive web application (PWA) технология разработки приложений

Progressive web application (PWA) - технология в web-разработке, которая визуально и функционально трансформирует сайт в приложение.

Чтобы интегрировать технологию PWA необходимо, чтобы сайт отвечал следующим требованиям:

- Service Worker;
- архитектура application shell;
- Web App Manifest;
- Push Notifications;
- SSL-сертификат для передачи данных по протоколу HTTPS.

Service Worker — это JavaScript-файл, который запускается в фоновом режиме как автономный сервис. Он не связан с DOM или web-страницами, работает на другом потоке и получает доступ к DOM с

помощью API `postMessage`. С точки зрения пользователя Service Worker позволяет выполнять такие действия, как, например, отправка push-уведомлений и предварительная загрузка материалов для просмотра в автономном режиме офлайн.

Application shell — это виртуальная оболочка. Подобно оболочке нативного приложения, она загружается при его запуске, а далее динамическая информация загружается на неё из сети.

Web App Manifest предоставляет информацию о приложении в текстовом JSON-файле. Необходимо, чтобы web-приложение было загружено и визуально отображалось для пользователя аналогично нативному приложению.

Push Notifications — технология для отправки push-уведомлений.

Помимо этого, PWA требует, чтобы все ресурсы сайта передавались по протоколу HTTPS.

PWA совмещает в себе свойства нативного приложения и функции браузера, что имеет свои преимущества:

- PWA поддерживается наиболее популярными ОС: Windows, iOS, Android;
- обновления добавляются разработчиками удаленно. Пользователи видят изменения и улучшения, но им не требуется скачивать эти обновления самостоятельно;
- PWA индексируется Google и другими поисковыми системами;
- благодаря сценарию Service Worker, который запускается браузером в фоновом режиме, и стратегии кэширования обеспечивается возможность работы офлайн;
- front отделен от back'а. Меньше времени и ресурсов тратится на разработку и переработку дизайна и логики взаимодействия PWA с клиентом;
- PWA можно установить без «Play Маркета» и App Store, а также вопреки запрету устанавливать приложения из неизвестных источников. Лояльно относятся к PWA и антивирусные программы. Одновременно с этим передача данных происходит по протоколу HTTPS, поэтому PWA безопасно;

Технология PWA не универсальна и имеет ряд недостатков:

- не все устройства и не все операционные системы поддерживают полный набор возможностей PWA;
- невозможно наладить активное участие пользователей iOS (например приложение может хранить локальные данные и файлы размером только до 50 Мбайт, нет доступа к In-App Payments (встроенные платежи) и многим другим сервисам Apple, нет интеграции с Siri), поддержка iOS начинается с версии 11.3;
- работа офлайн ограничена;
- работу PWA может ограничивать неполный доступ к аппаратным компонентам;
- нет достаточной гибкости в отношении «специального» контента для пользователей (например программы лояльности);
- при использовании PWA увеличивается расход заряда батареи мобильного устройства.

1.3 Multi-page application (MPA) технология разработки приложений

Multi-page application (MPA) - это многостраничные приложения, работающие, как привычные нам веб-сайты. Они отправляют запрос на сервер и полностью обновляют страницу, когда с ней совершается какое-либо действие. Подобная архитектура приложения значительно влияет на скорость и производительность, поскольку большая часть данных подгружается повторно при каждом переходе.

Принцип работы MPA приложения



МРА подходит крупным компаниям, предлагающим широкий спектр услуг и продуктов: интернет-магазины, сайты компаний, каталоги и торговые площадки.

Преимущества МРА:

- Простая SEO-оптимизация. МРА часто используют сайты, для которых важно попадать в топы поисковых систем. Каждая из страниц имеет уникальный URL и стабильна, что позволяет поисковым ботам адекватно ее просканировать.
- Масштабируемость. В МРА приложение можно вложить столько информации, сколько потребуется, без ограничений по страницам и функциям.
- Проверенная классика. МРА работают по тем же принципам, что и знакомые пользователю веб-сайты с классической навигацией.

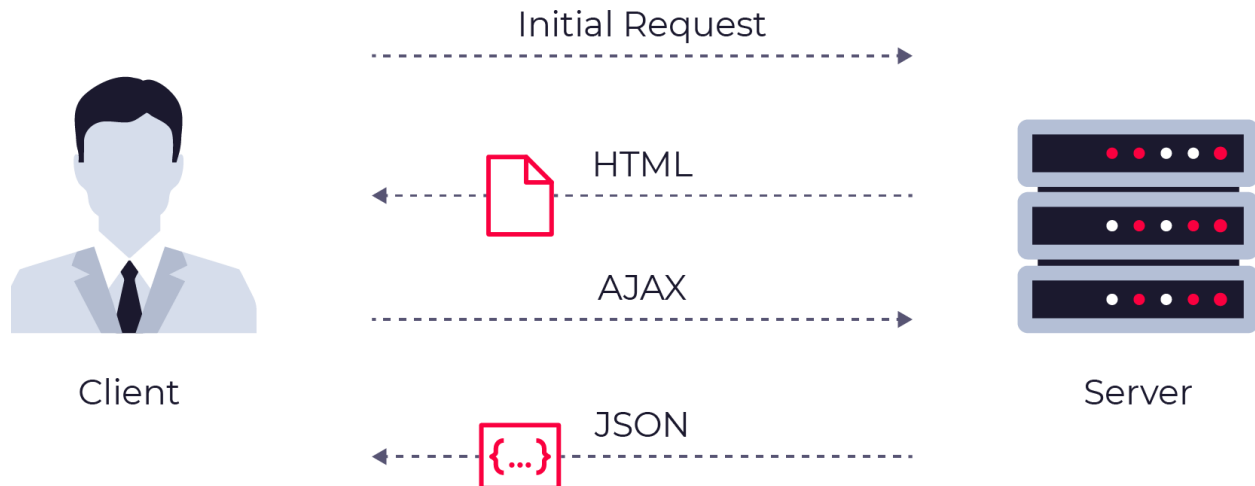
Недостатки МРА:

- Низкая скорость разработки. МРА приложения требуют использования фреймворков на обеих сторонах: клиента и сервера. Также из-за тесной взаимосвязи frontend и backend усложняется работа разработчиков. Всё это в совокупности увеличивает бюджет и сроки разработки.
- Затраты на создание мобильного приложения. МРА приложения плохо конвертируются в мобильные. Для этого в значительной части случаев потребуется разработка backend с нуля.

1.4 Single-page application (SPA) технология разработки приложений

Single-page application (SPA) - это одностраничное приложение, содержащее HTML-страницу, которая динамически обновляется в ответ на действия пользователя. Архитектура приложения устроена так, что при первоначальном запуске посетитель видит основной контент сайта в браузере, а новые данные загружаются на ходу по мере необходимости, например, при прокрутке или клике на иконку.

SPA Lifecycle



Одностраничные приложения во многом похожи на десктопные: при переходе на новую страницу обновляется только часть контента, позволяя не загружать одни и те же элементы сайта множество раз. Подобный эффект для SPA обеспечивают современные JavaScript фреймворки. Для небольших проектов подойдёт jQuery, однако для создания серьезного SPA потребуются такие frontend-библиотеки, как Angular.

SPA приложения применяются в случаях, когда загруженной на сайт информации немного и её требуется эффектно продемонстрировать пользователю. Примерами SPA являются соцсети и сайты с большим количеством пользователей. В вопросе скорости подгрузки у SPA сайтов нет равных, что и является их главным преимуществом. Так как львиная доля ресурсов загружается в одну сессию, при дальнейших действиях SPA быстро обновляет необходимую часть данных, значительно экономя время.

Преимущества SPA:

- **Легкость создания.** Для разработки SPA приложения уже готовы библиотеки и фреймворки, работа над frontend и backend может вестись параллельно. Кроме того, на основе готового кода в дальнейшем можно построить мобильное приложение;

- Гибкость пользовательского интерфейса. Для одной страницы проще разработать увлекательный и интерактивный дизайн интерфейса;
- Простое кэширование данных. Важные данные загружаются в один запрос, а далее пользователь одностраничного приложения может работать частично оффлайн, подключаясь к интернету, чтобы сохранить прогресс.

Недостатки SPA:

- Некоторые пользователи отключают JavaScript в браузерах, чтобы ускорить загрузку сайта и избавиться от нежелательной рекламы. У таких пользователей приложение просто не заработает. Кроме того, существует проблема утечки памяти JavaScript, которая уменьшает скорость загрузки и открывает ваши данные для действий злоумышленников.
- Плохо поддаются SEO оптимизации. URL страниц практически не меняется, а данные подгружаются динамически, когда для оптимизации важна устойчивость и уникальные URL для каждой страницы. Поисковым ботам сложно просканировать SPA приложение, и на данный момент грамотно индексировать такие сайты умеет только Google;
- Сильно нагружают браузер. Такая проблема возникает из-за относительно “тяжелых” клиентских фреймворков;
- Могут недешево стоить. Разумеется, итоговая стоимость разработки разнится, но цена сложных сайтов довольно высока.

1.5 Проблемы и трудности разработки Single-page application приложений.

Суммируя всё вышесказанное, можно выделить следующие ключевые отличия у SPA и MPA приложений:

SPA	MPA
Простая разработка	Низкая скорость
Гибкий интерфейс	Традиционный интерфейс

Простое кэширование данных	Высокая масштабируемость
Сложности в SEO оптимизации	Простая SEO оптимизация
Сильная нагрузка на браузер	Средняя нагрузка на браузер
Цена зависит от сложности сайта	Высокая стоимость разработки

Традиционный многостраничный сайт отличается от одностраничного тем, что при загрузке новой страницы на нем происходит ее генерирование под запрос серверной программой, что требует соответствующего времени. Мало того, обновление данных всей web-страницы создаёт серьёзную нагрузку на сервер.

Single-Page Applications, как правило, используют “AJAX”. Он представляет собой определённый подход к построению интерактивных интерфейсов, в которых предусмотрен обмен данными между браузером и сервером в фоновом режиме. Приложение отправляет запрос на сервер и получает информацию от него, за счёт чего и осуществляется выведение определенных частей страницы в браузере.

Выбирая тип веб-приложения для бизнеса, в первую очередь, стоит ориентироваться на его цели:

- одностраничные приложения выгодны для создания быстрого сайта и броского, многофункционального API интерфейса, внутри которого требуется разместить небольшой объем данных. Такой подход идеален для социальных сетей и SaaS-бизнеса;
- многостраничные приложения хороши для подачи большого количества информации и развертки широкого функционала в пределах веб-приложения.

У каждого подхода к построению веб-сайта есть свои достоинства и недостатки. Традиционные веб-приложения MPA просты и надежны, легко считываются поисковыми ботами и привычны пользователю, но разрабатывать их дороже и дольше. PWA и SPA приложения работают быстрее и проще в разработке, однако SEO-оптимизацию для них настраивать гораздо сложнее.

(Достоинства и недостатки SPA по сравнению с MPA, PWA и пр. проблемы и трудности)

2. Архитектура, технологии, шаблоны, принципы и проблемы реализации приложений на основе Single-page application (SPA) технологий в условиях цифровизации общества.

2.1 Система организации работы при разработке Single-page application приложений.

Сайты Single-Page Application работают по следующему принципу: при открытии пользователем web-страницы браузер загружает весь код приложения, выводя на экран лишь ту его часть, которую пользователь запросил. При переходе на другую страницу на экран выводятся ранее загруженные данные и при необходимости происходит динамическая подгрузка с сервера нужного содержимого, которая не требует обновления страницы. Таким образом, эти сайты работают достаточно быстро и создают минимальную нагрузку на сервер. С другой же – при загрузке первой страницы может потребоваться немного больше времени.

В целом, web-приложение SPA – ведь очень полезная. Согласно статистике, около половины пользователей Интернета не готовы ждать загрузки web-страницы более двух секунд. Около 40% уйдут с неё в том случае, если она будет загружаться более трёх секунд. Это значит, что задержка всего лишь в одну секунду может стать причиной снижения конверсии почти на 10%. По этой причине администраторы всячески стремятся сокращать время загрузки страниц, нередко используя для решения этой задачи SPA. Их «магия» заключается в том, чтобы после первого открытия web-сайта автоматически загружать все данные. Это значит, что браузер отображает выбранное пользователем состояние страницы, изменяя лишь её содержимое, но не её саму.

В случае с одностраничным сайтом загрузка первой страницы требует больше времени, чем с многостраничным. Однако, загрузка всех остальных страниц происходит в разы быстрее. У одностраничных back- и front-end полностью разделены, при этом первый может быть задействован повторно. Что касается степени зависимости от JavaScript у одностраничных сайтов она есть, что может вызвать сложности с первой загрузкой.

В первую очередь для разработки SPA необходимо использовать соответствующие базы данных, благодаря которым можно выбрать

оптимальный для того или иного случая вариант. Чаще всего это “MySQL”, “PostgreSQL” или “MongoDB”. Такой инструмент, как “AJAX”, без которого не обходится запуск ни одного SPA. Он представляет собой определенный подход, при который обмен данными браузера с сервером происходит в фоновом режиме.

Backend-технологии, обеспечивающие реализацию логики сайта. Оптимальным вариантом может стать фреймворк на программной платформе “Node.JS” или языке “PHP”.

Сложность работы SEO со SPA-сайтами в том, что они нарушают принципы индексации, что критично для SEO. А именно:

- на SPA-сайтах только один URL — нет перезагрузки страниц;
- одностраничные приложения не содержат контента в пригодном для сканирования виде;
- поисковые системы, кроме Google, не способны обрабатывать JavaScript, на котором написан сайт;
- SEO-рекомендации от Яндекс и Google по работе с SPA-сайтами противоречат друг другу.

Для создания SPA используются библиотеки и фреймворки на языке “JavaScript”, разработанные специально для решения этой задачи. В зависимости от ситуации их выбирают индивидуально среди “React”, “Vue” и “Angular”, или же используют JS в чистом виде.

Фреймворки — это программные продукты, которые упрощают создание и поддержку технически сложных нагруженных проектов. Фреймворк, как правило, содержит только базовые программные модули, а все специфичные для проекта компоненты реализуются разработчиком на их основе. Тем самым достигается не только высокая скорость разработки, но и большая производительность и надежность решений.

С точки зрения бизнеса разработка на фреймворке почти всегда экономически эффективнее и качественнее по результату, нежели написание проекта на чистом языке программирования без использования каких-либо платформ. Разработка без использования платформы может быть правильным решением только в двух случаях — либо проект совсем простой и не требующий дальнейшего развития, либо очень нагруженный и требует очень низкоуровневой оптимизации (например, веб-сервисы с десятками тысяч обращений в секунду). Во

всех других случаях разработка на программной платформе быстрее и качественнее.

Одним из главных преимуществ в использовании фреймворков является то, что фреймворк определяет унифицированную структуру для построенных на его базе приложений. Поэтому приложения на фреймворках значительно проще сопровождать и дорабатывать, так как стандартизированная структура организации компонентов понятна всем разработчикам на этой платформе и не требуется долго разбираться в архитектуре, чтобы понять принцип работы приложения или найти место реализации того или иного функционала. Большинство фреймворков для разработки веб-приложений использует парадигму MVC (модель-представление-контроллер) — то есть очень во многих фреймворках идентичный подход к организации компонентов приложения и это еще больше упрощает понимание архитектуры приложения даже на незнакомом разработчику фреймворке.

((в параграфе нужно провести анализ предметной области на основе источников литературы, РИНЦ, СКОПУС, в том числе рассмотреть модели программных особенностей работы браузеров, серверов, скриптов, фреймворков, создания виртуальных объектных моделей документов, особенностей обновления контента, особенностей создания «гибких» интерфейсов, сложности в SEO оптимизации, особенности решения проблем нагрузок на браузеры, особенностей использования frontend-библиотек типа Angular и пр. в условиях реализации Single-page application (SPA)

2.2 Структура и функции систем контроля версий при разработке Single-page application приложений.

Порядок использования системы управления версиями в каждом конкретном случае определяется техническими регламентами и правилами, принятыми в конкретной фирме или организации, разрабатывающей проект. Тем не менее, общие принципы правильного использования VCS немногочисленны и едины для любых разработок и систем управления версиями.

- Любые рабочие, тестовые или демонстрационные версии проекта собираются только из репозитория системы.
«Персональные» сборки, включающие ещё незафиксированные изменения, могут делать только разработчики для целей промежуточного тестирования.

Таким образом, гарантируется, что репозиторий содержит всё необходимое для создания рабочей версии проекта.

- Текущая версия главной ветви всегда корректна. Не допускается фиксация в главной ветви неполных или не прошедших хотя бы предварительное тестирование изменений. В любой момент сборка проекта, проведенная из текущей версии, должна быть успешной.
- Любое значимое изменение должно оформляться как отдельная ветвь. Промежуточные результаты работы разработчика фиксируются в эту ветвь. После завершения работы над изменением ветвь объединяется со стволом. Исключения допускаются только для мелких изменений, работа над которыми ведётся одним разработчиком в течение не более чем одного рабочего дня.
- Версии проекта помечаются тегами. Выделенная и помеченная тегом версия более никогда не изменяется.

2.3 Принципы, функции и архитектура систем контроля версий.

Каждая система управления версиями имеет свои специфические особенности в наборе команд, порядке работы пользователей и администрировании. Тем не менее, общий порядок работы для большинства VCS совершенно стереотипен. Здесь предполагается, что проект, каким бы он ни был, уже существует и на сервере размещен его репозиторий, к которому разработчик получает доступ.

Начало работы с проектом. Первым действием, которое должен выполнить разработчик, является извлечение рабочей копии проекта или той его части, с которой предстоит работать. Это действие выполняется с помощью команды извлечения версии. Разработчик задает версию, которая должна быть скопирована, по умолчанию обычно копируется последняя (или выбранная администратором в качестве основной) версия.

По команде извлечения устанавливается соединение с сервером, и проект в виде дерева каталогов и файлов копируется на компьютер разработчика. Обычной практикой является дублирование рабочей копии: помимо основного каталога с проектом на локальный диск дополнительно записывается ещё одна его копия. Работая с проектом,

разработчик изменяет только файлы основной рабочей копии. Вторая локальная копия хранится в качестве эталона, позволяя в любой момент без обращения к серверу определить, какие изменения внесены в конкретный файл или проект в целом и от какой версии была отделена рабочая копия. Как правило, любая попытка ручного изменения этой копии приводит к ошибкам в работе программного обеспечения VCS.

Ежедневный цикл работы. При некоторых вариациях, определяемых особенностями системы и деталями принятого технологического процесса, обычный цикл работы разработчика в течение рабочего дня выглядит следующим образом.

Обновление рабочей копии. По мере внесения изменений в основную версию проекта рабочая копия на компьютере разработчика стареет: расхождение ее с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений. Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версии, для чего разработчик выполняет операцию обновления рабочей копии настолько часто.

Модификация проекта. Разработчик модифицирует проект, изменяя входящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу VCS.

Фиксация изменений. Завершив очередной этап работы над заданием, разработчик фиксирует свои изменения, передавая их на сервер. VCS может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей копии. При наличии в системе поддержки отложенных изменений изменения могут быть переданы на сервер без фиксации. Если утвержденная политика работы в VCS это позволяет, то фиксация изменений может проводиться не ежедневно, а только по завершении работы над заданием. В этом случае до завершения работы все связанные с заданием изменения сохраняются только в локальной рабочей копии разработчика.

Ветвления. Делать мелкие исправления в проекте можно путем непосредственной правки рабочей копии и последующей фиксации изменений прямо в главной ветви на сервере. Однако при выполнении объёмных работ такой порядок становится неудобным: отсутствие фиксации промежуточных изменений на сервере не позволяет работать над чем-либо в групповом режиме, кроме того, повышается риск потери

изменений при локальных авариях и теряется возможность анализа и возврата к предыдущим вариантам кода в пределах данной работы. Поэтому для таких изменений обычной практикой является создание ветвей, то есть «отпочковывание» от ствола в какой-то версии нового варианта проекта или его части, разработка в котором ведётся параллельно с изменениями в основной версии. Ветвь создаётся специальной командой. Рабочая копия ветви может быть создана заново обычным образом, либо путем переключения имеющейся рабочей копии на заданную ветвь.

Базовый рабочий цикл при использовании ветвей остаётся точно таким же, как и в общем случае: разработчик периодически обновляет рабочую копию и фиксирует в ней свою ежедневную работу. Иногда ветвь разработки так и остаётся самостоятельной, но чаще всего, когда работа, для которой создана ветвь, выполнена, ветвь интегрируется в основную ветвь. Это может делаться командой слияния, либо путём создания патча, содержащего внесенные в ходе разработки ветви изменения и применения этого патча к текущей основной версии проекта.

Слияние версий. Три вида операций, выполняемых в системе управления версиями, могут приводить к необходимости объединения изменений. Это:

- Обновление рабочей копии.
- Фиксация изменений.
- Слияние ветвей.

Во всех случаях ситуация принципиально одинакова и имеет следующие характерные черты:

- Ранее была сделана копия дерева файлов и каталогов репозитория или его части.
- Впоследствии и в оригинальное дерево, и в копию были независимо внесены некоторые изменения.
- Требуется объединить изменения в оригинале и копии таким образом, чтобы не нарушить логическую связность проекта и не потерять данные.

Совершенно очевидно, что при невыполнении второго условия объединение элементарно — достаточно скопировать изменённую часть туда, где изменений не было. В противном случае слияние изменений превращается в нетривиальную задачу, во многих случаях требующую

вмешательства разработчика. В целом механизм автоматического слияния изменений работает, основываясь на следующих принципах:

- Изменения могут состоять в модификации содержимого файла, создании нового файла или каталога, удалении или переименовании ранее существовавшего файла или каталога в проекте.
- Если два изменения относятся к разным и не связанным между собой файлам и/или каталогам, они всегда могут быть объединены автоматически. Их объединение состоит в том, что изменения, сделанные в каждой версии проекта, копируются в объединенную версию.
- Создание, удаление и переименование файлов в каталогах проекта могут быть объединены автоматически, если только они не конфликтуют между собой. В этом случае изменения, сделанные в каждой версии проекта, копируются в объединенную версию.

Конфликтующими обычно являются:

- Удаление и изменение одного и того же файла или каталога.
- Удаление и переименование одного и того же файла или каталога.
- Создание в разных версиях файла с одним и тем же именем и разным содержимым.
- Изменения в пределах одного текстового файла, сделанные в разных версиях, могут быть объединены, если они находятся в разных местах этого файла и не пересекаются. В этом случае в объединенную версию вносятся все сделанные изменения.
- Изменения в пределах одного файла, если он не является текстовым, всегда являются конфликтующими и не могут быть объединены автоматически.

Во всех случаях базовой версией для слияния является версия, в которой было произведено разделение сливаемых версий. Если это операция фиксации изменений, то базовой версией будет версия последнего обновления перед фиксацией, если обновление — то версия предыдущего обновления, если слияние ветвей — то версия, в которой была создана соответствующая ветвь. Соответственно,

сопоставляемыми наборами изменений будут наборы изменений, сделанных от базовой до текущей версии во всех объединяемых вариантах.

Абсолютное большинство современных систем управления версиями ориентировано, в первую очередь, на проекты разработки программного обеспечения, в которых основным видом содержимого файла является текст. Соответственно, механизмы автоматического слияния изменений ориентируются на обработку текстовых файлов, то есть файлов, содержащих текст, состоящий из строк буквенно-цифровых символов, пробелов и табуляций, разделенных символами перевода строки.

При определении допустимости слияния изменений в пределах одного и того же текстового файла работает типовой механизм построчного сравнения текстов, который сравнивает объединяемые версии с базовой и строит список изменений, то есть добавленных, удаленных и измененных наборов строк. Минимальной единицей данных для этого алгоритма является строка, даже самое малое отличие делает строки различными. С учётом того, что символы-разделители, в большинстве случаев, не несут смысловой нагрузки, механизм слияния может игнорировать эти символы при сравнении строк.

Те найденные наборы изменённых строк, которые не пересекаются между собой, считаются совместимыми и их слияние делается автоматически. Если в сливаемых файлах находятся изменения, затрагивающие одну и ту же строку файла, это приводит к конфликту. Такие файлы могут быть объединены только вручную. Любые файлы, кроме текстовых, с точки зрения VCS являются бинарными и не допускают автоматического слияния.

Конфликты и их разрешение. Ситуацию, когда при слиянии нескольких версий сделанные в них изменения пересекаются между собой, называют конфликтом. При конфликте изменений система управления версиями не может автоматически создать объединенный проект и вынуждена обращаться к разработчику. Как уже говорилось выше, конфликты могут возникать на этапах фиксации изменений, обновления или слияния ветвей. Во всех случаях при обнаружении конфликта соответствующая операция прекращается до его разрешения.

Для разрешения конфликта система, в общем случае, предлагает разработчику три варианта конфликтующих файлов: базовый, локальный и серверный. Конфликтующие изменения либо показываются разработчику в специальном программном модуле объединения изменений, либо просто помечаются специальной разметкой прямо в тексте объединённого файла.

Конфликты в файловой системе разрешаются проще: там может конфликтовать только удаление файла с одной из прочих операций, а порядок файлов в каталоге не имеет значения, так что разработчику остается лишь выбрать, какую операцию нужно сохранить в сливаемой версии.

Блокировки. Механизм блокировки позволяет одному из разработчиков захватить в монопольное использование файл или группу файлов для внесения в них изменений. На то время, пока файл заблокирован, он остаётся доступным всем остальным разработчикам только на чтение, и любая попытка внести в него изменения отвергается сервером. Технически блокировка может быть организована по-разному. Типичным для современных систем является следующий механизм.

- Файлы, для работы с которыми требуется блокировка, помечаются специальным флагом «блокируемый». Такая пометка может ставиться автоматически при добавлении файла в проект, обычно для этого предварительно создается список масок имен файлов, которые при добавлении должны становиться блокируемыми.
- Если файл помечен как блокируемый, то при извлечении рабочей копии с сервера он получает в локальной файловой системе атрибут «только для чтения», что препятствует его случайному редактированию.
- Разработчик, желающий изменить файл, вызывает специальную команду блокировки с указанием имени этого файла.

В результате работы этой команды происходит следующее:

- сервер проверяет, не заблокирован ли уже файл другим разработчиком; если это так, то команда блокировки завершается с ошибкой «файл заблокирован другим пользователем» и разработчик, вызвавший ее, должен

ожидать, пока другой пользователь не снимет свою блокировку;

- файл на сервере помечается как «заблокированный», с сохранением идентификатора заблокировавшего его разработчика и времени блокировки;
- если блокировка на сервере прошла удачно, на локальной файловой системе с файла рабочей копии снимается атрибут «только для чтения», что позволяет начать его редактировать.
- Разработчик работает с заблокированным файлом. Если в процессе работы выясняется, что файл изменять не нужно, он может вызвать команду снятия блокировки. Все изменения файла будут отменены, локальный файл вернётся в состояние «только для чтения», с файла на сервере будет снят атрибут «заблокирован» и другие разработчики получат возможность изменять этот файл.
- По завершении работы с блокируемым файлом разработчик фиксирует изменения. Обычно блокировка при этом снимается автоматически, хотя в некоторых системах блокировку требуется снимать вручную после фиксации, либо указывать в команде фиксации изменений соответствующий параметр. Так или иначе, при этом файл после изменений теряет флаг «заблокирован» и может быть изменен другими разработчиками.

Массовое использование блокировок, когда все или большинство файлов в проекте являются блокируемыми и для любых изменений необходимо заблокировать соответствующий набор файлов, называется ещё стратегией «блокированного извлечения». Ранние системы управления версиями поддерживали исключительно эту стратегию, предотвращая таким способом появление конфликтов на корню. В современных VCS предпочтительным является использование неблокирующих извлечений, блокировки же считаются скорее неизбежным злом, которое нужно по возможности ограничивать. Недостатки использования блокировок очевидны:

- Блокировки просто мешают продуктивной работе, поскольку вынуждают ожидать освобождения заблокированных файлов, хотя в большинстве случаев даже совместные изменения одних и тех же файлов, которые делаются в ходе разных по смыслу работ, не пересекаются и объединяются при слиянии автоматически.
- Частота возникновения конфликтов и сложность их разрешения в большинстве случаев не настолько велики, чтобы создать серьезные затруднения. Возникновение же серьезного конфликта изменений чаще всего сигнализирует либо о существенном расхождении во мнениях разных разработчиков относительно дизайна одного и того же фрагмента, либо о неправильной организации работы.
- Блокировки создают административные проблемы. Типичный пример: разработчик может забыть снять блокировку с занятых им файлов, уходя в отпуск. Для разрешения подобных проблем приходится применять административные меры, в том числе включать в систему технические средства для сброса неверных блокировок, но и при их наличии на приведение системы в порядок расходуется время.

С другой стороны, в некоторых случаях использование блокировок вполне оправданно. Очевидным примером является организация работы с бинарными файлами, для которых нет инструментальных средств слияния изменений либо такое слияние принципиально невозможно (как, например, для файлов изображений). Если автоматическое слияние невозможно, то при обычном порядке работы любое параллельное изменение подобных файлов будет приводить к конфликту. В данном случае гораздо удобнее сделать такой файл блокируемым, чтобы гарантировать, что любые изменения в него будут вноситься только последовательно.

Версии проекта, теги. Система управления версиями обеспечивает хранение всех существовавших вариантов файлов и, как следствие, всех вариантов проекта в целом, имевших место с момента начала его разработки. Но само понятие «версии» в разных системах может трактоваться двояко.

Одни системы поддерживают версиюность файлов. Это означает, что любой файл, появляющийся в проекте, получает собственный номер версии. При каждой фиксации разработчиком изменений, затрагивающих файл, соответствующая часть фиксируемых изменений применяется к файлу и файл получает новый, обычно следующий по порядку, номер версии. Поскольку фиксации обычно затрагивают только часть файлов в репозитории, номера версий файлов, имеющиеся на один и тот же момент времени, со временем расходятся, и проект в целом, фактически, никакого «номера версии» не имеет, поскольку состоит из множества файлов с различными номерами версий. Подобным образом работает, например, система управления версиями CVS.

Для других систем понятие «версия» относится не к отдельному файлу, а к репозиторию целиком. Вновь созданный пустой репозиторий имеет версию 1 или 0, любая фиксация изменений приводит к увеличению этого номера. Таким способом трактует номера версий, например, система Subversion. Номера версии отдельного файла здесь, фактически, не существует, условно можно считать таковым текущий номер версии репозитория. Иногда, говоря о «версии файла» в таких системах, имеют в виду ту версию репозитория, в которой файл был последний раз изменен.

Для практических целей обычно имеет значение не отдельный файл, а весь проект целиком. В системах, поддерживающих версиюность отдельных файлов, для идентификации определенной версии проекта можно использовать дату и время — тогда версия проекта будет состоять из тех версий входящих в него файлов, которые имелись в репозитории на указанный момент времени. Если поддерживается версияность репозитория в целом, номером версии проекта может выступать номер версии репозитория. Однако оба варианта не слишком удобны, так как ни дата, ни номер версии репозитория обычно не несут информации о значимых изменениях в проекте, о том, насколько долго и интенсивно над ним работали. Для более удобной пометки версий проекта системы управления версиями поддерживают понятие тегов.

Тег (tag) — это символическая метка, которая может быть связана с определенной версией файла или каталога в репозитории. С помощью соответствующей команды всем или части файлов проекта, отвечающим определенным условиям может быть присвоена заданная метка. Таким

образом можно идентифицировать версию проекта, зафиксировав таким образом его состояние на некоторый желаемый момент. Как правило, система тегов достаточно гибкая и позволяет пометить одним тегом и не одновременные версии файлов и каталогов. Это позволяет собрать «версию проекта» любым произвольным образом. С точки зрения пользователя системы пометка тегами может выглядеть по-разному. В некоторых системах она изображается именно как пометка. В других системах тег представляет собой просто отдельный каталог на файловом дереве репозитория, куда из ствола и ветвей проекта с помощью команды копирования делаются копии нужных версий файлов. Так что визуально тег — это просто вынесенная в отдельный каталог копия определённых версий файлов репозитория. По соглашению в дерево каталогов, соответствующее тегу, запрещена фиксация изменений.

(Типовая (Базовая методика с системой как правило включает рассмотрение следующих этапов : 1 Начало работы с проектом, 2 Ежедневный цикл работы, 3 Ветвления, 4 Слияние версий, 5 Конфликты и их разрешение, 6 Блокировки, 7 Версии проекта, теги, 8 Базовые принципы разработки ПО, например в VCS, 9. Особенности работы в распределённых системах управления версиями)

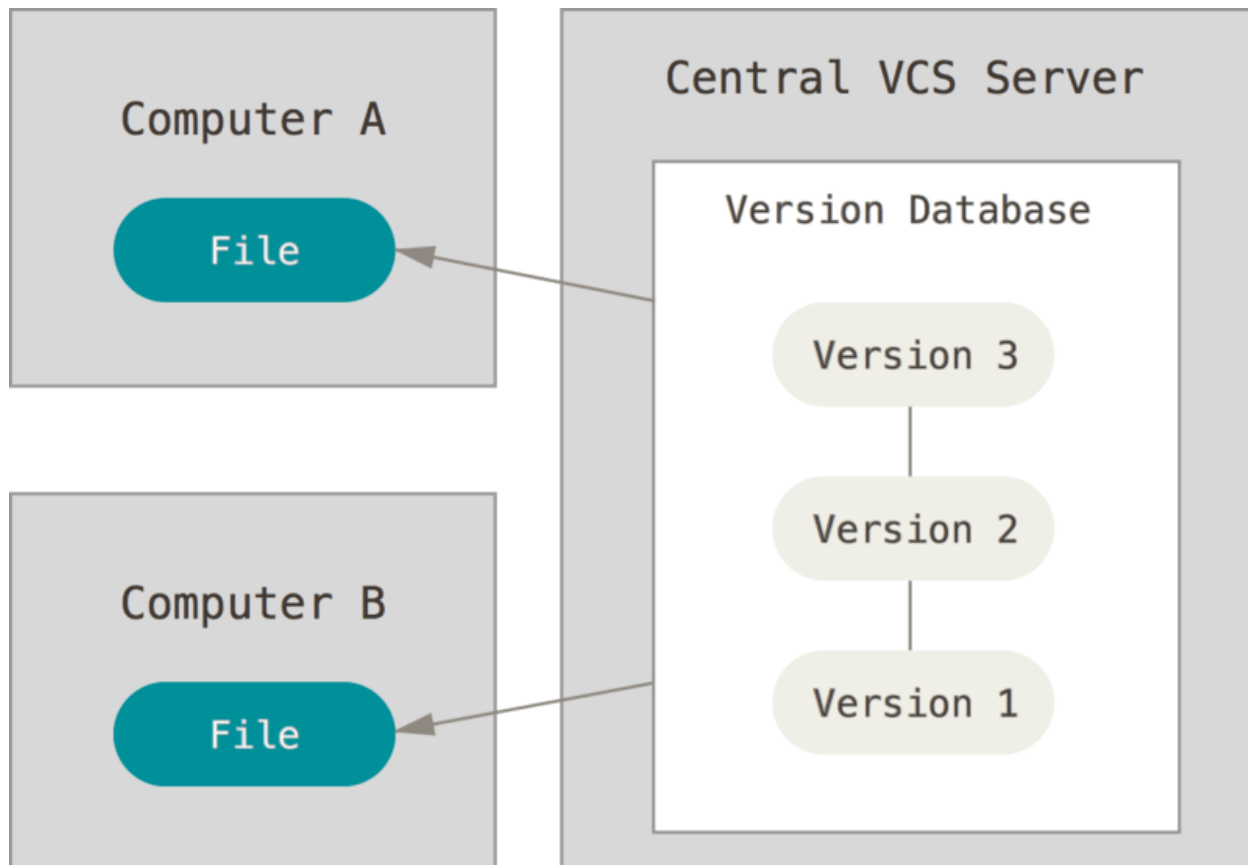
2.4 Централизованные системы контроля версий.

Для организации такой системы контроля версий используется единственный сервер, который содержит все версии файлов. Клиенты, обращаясь к этому серверу, получают из этого централизованного хранилища. Применение централизованных систем контроля версий на протяжении многих лет являлась стандартом. К ним относятся CVS, Subversion, Perforce.

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определенной 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 серьезные 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.



2.5 Распределённые системы контроля версий.

Такие системы используют распределенную модель вместо традиционной клиент-серверной. Они, в общем случае, не нуждаются в централизованном хранилище: вся история изменения документов хранится на каждом компьютере, в локальном хранилище, и при необходимости отдельные фрагменты истории локального хранилища синхронизируются с аналогичным хранилищем на другом компьютере. В некоторых таких системах локальное хранилище располагается непосредственно в каталогах рабочей копии.

Когда пользователь такой системы выполняет обычные действия, такие как извлечение определенной версии документа, создание новой версии и тому подобное, он работает со своей локальной копией хранилища. По мере внесения изменений, хранилища, принадлежащие разным разработчикам, начинают различаться, и возникает необходимость в их синхронизации. Такая синхронизация может осуществляться с помощью обмена патчами или так называемыми наборами изменений между пользователями.

Описанная модель логически близка созданию отдельной ветки для каждого разработчика в классической системе управления версиями. Отличие состоит в том, что до момента синхронизации другие разработчики этой ветки не видят. Пока разработчик изменяет только свою ветвь, его работа не влияет на других участников проекта и наоборот. По завершении обособленной части работы, внесенные в ветви изменения сливаются с основной ветвью. Как при слиянии ветвей, так и при синхронизации разных хранилищ возможны конфликты версий. На этот случай во всех системах предусмотрены те или иные методы обнаружения и разрешения конфликтов слияния.

С точки зрения пользователя распределенная система отличается необходимостью создавать локальный репозиторий и наличием в командном языке двух дополнительных команд: команды получения репозитория от удаленного компьютера и передачи своего репозитория на удаленный компьютер. Первая команда выполняет слияние изменений удаленного и локального репозитория с помещением результата в локальный репозиторий; вторая — наоборот, выполняет слияние изменений двух репозитория с помещением результата в удаленный репозиторий. Как правило, команды слияния в распределенных системах позволяют выбрать, какие наборы изменений будут передаваться в другой репозиторий или извлекаться из него, исправлять конфликты слияния непосредственно в ходе операции или после её неудачного завершения, повторять или возобновлять неоконченное слияние. Обычно передача своих изменений в чужой репозиторий завершается удачно только при условии отсутствия конфликтов. Если конфликты возникают, пользователь должен сначала слить версии в своём репозитории, и лишь затем передавать их другим.

Основные преимущества распределенных систем — их гибкость и значительно большая автономия отдельного рабочего места. Каждый

компьютер разработчика является, фактически, самостоятельным и полнофункциональным сервером, из таких компьютеров можно построить произвольную по структуре и уровню сложности систему, задав желаемый порядок синхронизации. При этом каждый разработчик может вести работу независимо, так, как ему удобно, изменяя и сохраняя промежуточные версии документов, пользуясь всеми возможностями системы даже в отсутствие сетевого соединения с сервером. Связь с сервером или другими разработчиками требуется исключительно для проведения синхронизации, при этом обмен наборами изменений может осуществляться по различным схемам.

К недостаткам распределенных систем можно отнести увеличение требуемого объема дисковой памяти: на каждом компьютере приходится хранить полную историю версий, тогда как в централизованной системе на компьютере разработчика обычно хранится лишь рабочая копия, то есть срез репозитория на какой-то момент времени и внесенные изменения. Менее очевидным, но неприятным недостатком является то, что в распределенной системе практически невозможно реализовать некоторые виды функциональности, предоставляемые централизованными системами. Это:

- Блокировка файла или группы файлов. Это вынуждает применять специальные административные меры, если приходится работать с бинарными файлами, непригодными для автоматического слияния.
- Слежение за определенным файлом или группой файлов.
- Единая сквозная нумерация версий системы и файлов, в которой номер версии монотонно возрастает. В распределённых системах приходится обходиться локальными обозначениями версий и применять теги, назначение которых определяется соглашением между разработчиками или корпоративными стандартами фирмы.
- Локальная работа пользователя с отдельной, небольшой по объему выборкой из значительного по размеру и внутренней сложности хранилища на удалённом сервере.

Можно выделить следующие типичные ситуации, в которых использование распределенной системы дает заметные преимущества:

- Периодическая синхронизация нескольких компьютеров под управлением одного разработчика. Использование

распределенной системы избавляет от необходимости выделять один из компьютеров в качестве сервера, а синхронизация выполняется по необходимости, обычно при «пересадке» разработчика с одного устройства на другое.

- Совместная работа над проектом небольшой территориально распределенной группы разработчиков без выделения общих ресурсов. Как и в предыдущем случае, реализуется схема работы без главного сервера, а актуальность репозитория поддерживается периодическими синхронизациями по схеме «каждый с каждым».
- Крупный распределенный проект, участники которого могут долгое время работать каждый над своей частью, при этом не имеют постоянного подключения к сети. Такой проект может использовать централизованный сервер, с которым синхронизируются копии всех его участников. Возможны и более сложные варианты — например, с созданием групп для работы по отдельным направлениям внутри более крупного проекта. При этом могут быть выделены отдельные «групповые» серверы для синхронизации работы групп, тогда процесс окончательного слияния изменений становится древовидным: сначала отдельные разработчики синхронизируют изменения на групповых серверах, затем обновлённые репозитории групп синхронизируются с главным сервером. Возможна работа и без «групповых» серверов, тогда разработчики одной группы синхронизируют изменения между собой, после чего любой из них передаёт изменения на центральный сервер.

В традиционной разработке проектов, когда группа разработчиков относительно невелика и целиком находится на одной территории, в пределах единой локальной компьютерной сети, с постоянно доступными серверами, централизованная система может оказаться лучшим выбором из-за своей более жесткой структуры и наличия функциональности, отсутствующей в распределенных системах. Возможность фиксировать изменения без их слияния в центральную ветвь в таких условиях легко реализуется путем выделения незавершенных работ в отдельные ветви разработки.

ВЫВОД ПО ГЛАВЕ 1.

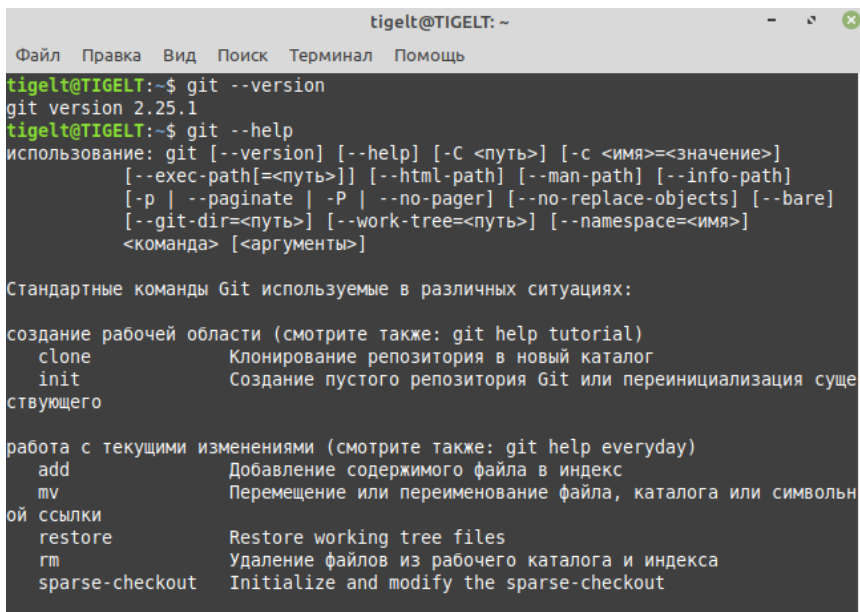
ГЛАВА 2. Разработка методики работы с системой контроля версий для разработки Single-page application приложений.

1 Методика работы с СКВ для Single-page application(SPA) приложения.

1.1 Методика работы с СКВ для Single-page application(SPA) приложения через консоль.

Установка Git. На официальном сайте гита необходимо установить гит на свой локальный компьютер. Ход установки...

Первая команда, которую хотелось бы рассмотреть, это команда (`git --help`). Эта команда показывает базовые возможности гита, после этой команды в консоль выводится большое количество информации в виде названия команд и описания к ним.



```

tigelt@TIGELT: ~
Файл  Правка  Вид  Поиск  Терминал  Помощь
tigelt@TIGELT:~$ git --version
git version 2.25.1
tigelt@TIGELT:~$ git --help
использование: git [--version] [--help] [-C <путь>] [-c <имя>=<значение>]
                [--exec-path[=<путь>]] [--html-path] [--man-path] [--info-path]
                [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
                [--git-dir=<путь>] [--work-tree=<путь>] [--namespace=<имя>]
                <команда> [<аргументы>]

Стандартные команды Git используемые в различных ситуациях:

создание рабочей области (смотрите также: git help tutorial)
  clone      Клонирование репозитория в новый каталог
  init       Создание пустого репозитория Git или переинициализация существующего

работа с текущими изменениями (смотрите также: git help everyday)
  add        Добавление содержимого файла в индекс
  mv         Перемещение или переименование файла, каталога или символической ссылки
  restore    Restore working tree files
  rm         Удаление файлов из рабочего каталога и индекса
  sparse-checkout  Initialize and modify the sparse-checkout
  
```

Чтобы начать работать с гитом необходимо сделать первоначальную настройку, для этого нужно сделать три вещи: указать имя пользователя, почту и текстовый редактор по умолчанию. Введенные нами данные будут отображаться в коммитах, текстовый редактор можно будет выбрать при первом коммите, так что оставим это на потом. Для того чтобы работать с конфигурацией у гита есть команда которая так и называется (`git config`), также имеется три уровня конфигурации:

1. Покрывает всех пользователей системы(`git config - - system`)
2. Настройки которые будут применяться к конкретному пользователю системы(`git config - - global`)

3. Уровень который будет применяться к конкретному репозиторию с которым мы работаем на данный момент (`git config --local`)
Мы будем работать с третьим вариантом, то есть настройки будут применяться ко всем репозиториям конкретно для данного пользователя в системе.

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git config --global user.name "Ivan Elistratov"
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git config --global user.email "tigelt2014@gmail.com"
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Для того чтобы начать работу с гитом необходимо проинициализировать репозиторий, для этого в консоли мы переходим в папку проекта и выполняем команду (`git init`), после этого мы получаем сообщение о том, что был проинициализирован репозиторий и указывается путь до папки с нашим проектом.

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь
fetch      Загрузка объектов и ссылок из другого репозитория
pull       Извлечение изменений и объединение с другим репозиторием или локальной веткой
push       Обновление внешних ссылок и связанных объектов

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
tigelt@TIGELT:~$ ls
D          node_modules          Видео          Изображения    'Рабочий стол'
django    nodesource_setup.sh   Документы      Музыка          Шаблоны
LR        package-lock.json     Загрузки      Общедоступные

tigelt@TIGELT:~$ cd D
tigelt@TIGELT:~/D$ ls
animePower  demoreact  nodaCurse  prog  react-theory  vue
tigelt@TIGELT:~/D$ cd demoreact/
tigelt@TIGELT:~/D/demoreact$ ls
demoreactapp  help.png
tigelt@TIGELT:~/D/demoreact$ cd demoreactapp/
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git init
Инициализирован пустой репозиторий Git в /home/tigelt/D/demoreact/demoreactapp/.git/
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

На данный момент в папке с нашим проектом не произошло никаких изменений, однако создалась скрытая системная папка, которая называется (`.git`). Внутри неё лежат папки относящиеся к самому гиту, в дальнейшем там будет храниться различная техническая информация по работе гита с нашим проектом, но мы не будем с ней работать, но нужно знать что она есть.

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp/.git
Файл  Правка  Вид  Поиск  Терминал  Помощь

push                                Обновление внешних ссылок и связанных объектов

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
tigelt@TIGELT:~$ ls
D          node_modules          Видео          Изображения    'Рабочий стол'
django    nodesource setup.sh    Документы      Музыка          Шаблоны
LR        package-lock.json      Загрузки      Общедоступные
tigelt@TIGELT:~$ cd D
tigelt@TIGELT:~/D$ ls
animePower  demoreact  nodaCurse  prog  react-theory  vue
tigelt@TIGELT:~/D$ cd demoreact/
tigelt@TIGELT:~/D/demoreact$ ls
demoreactapp  help.png
tigelt@TIGELT:~/D/demoreact$ cd demoreactapp/
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git init
Инициализирован пустой репозиторий Git в /home/tigelt/D/demoreact/demoreactapp/.
git/
tigelt@TIGELT:~/D/demoreact/demoreactapp$ cd .git
tigelt@TIGELT:~/D/demoreact/demoreactapp/.git$ ls
branches  config  description  HEAD  hooks  info  objects  refs
tigelt@TIGELT:~/D/demoreact/demoreactapp/.git$

```

На данном этапе уже можно создать основу приложения используя библиотеку ReactJS. В папке с нашим проектом появятся папки и файлы, которые собственно и являются самим приложением.

Мы уже совершили некоторые изменения с нашим репозиторием, для того чтобы понять в каком состоянии находится гит мы можем использовать базовую команду (`git status`), после выполнения этой команды мы получаем информацию о том, что сейчас мы находимся на мастер ветке, на данный момент нет коммитов, но есть неотслеживаемые файлы, как раз таки файлы нашего приложения, за которыми не следит

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь

tigelt@TIGELT:~/D/demoreact/demoreactapp$ ls
node_modules  package.json  package-lock.json  public  README.md  src
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master

Еще нет коммитов

Неотслеживаемые файлы:
(используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
.
.gitignore
README.md
package-lock.json
package.json
public/
src/

ничего не добавлено в коммит, но есть неотслеживаемые файлы (используйте «git add», чтобы отслеживать их)
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

ГИТ.

Чтобы гит начал следить за этими файлами необходимо прописать команду (`git add .`). Мы можем добавлять не все файлы сразу, а выборочно, для этого нужно прописать команду (`git add <имя файла>`). Если теперь посмотреть статус гита, то можно увидеть что появились определенные изменения за которыми гит следит и мы добавили новые

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git add .
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master

Еще нет коммитов

Изменения, которые будут включены в коммит:
(используйте «git rm --cached <файл>...», чтобы убрать из индекса)
    новый файл:   .gitignore
    новый файл:   README.md
    новый файл:   package-lock.json
    новый файл:   package.json
    новый файл:   public/favicon.ico
    новый файл:   public/index.html
    новый файл:   public/logo192.png
    новый файл:   public/logo512.png
    новый файл:   public/manifest.json
    новый файл:   public/robots.txt
    новый файл:   src/App.css
    новый файл:   src/App.js
    новый файл:   src/App.test.js
    новый файл:   src/index.css
    новый файл:   src/index.js
    новый файл:   src/logo.svg
    новый файл:   src/reportWebVitals.js
    новый файл:   src/setupTests.js

tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

файлы.

Если нам нужно не отслеживать файлы которые мы добавили нужно написать команду (`git rm --cached <имя файла>`)

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git rm --cached README.md
rm 'README.md'
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master

Еще нет коммитов

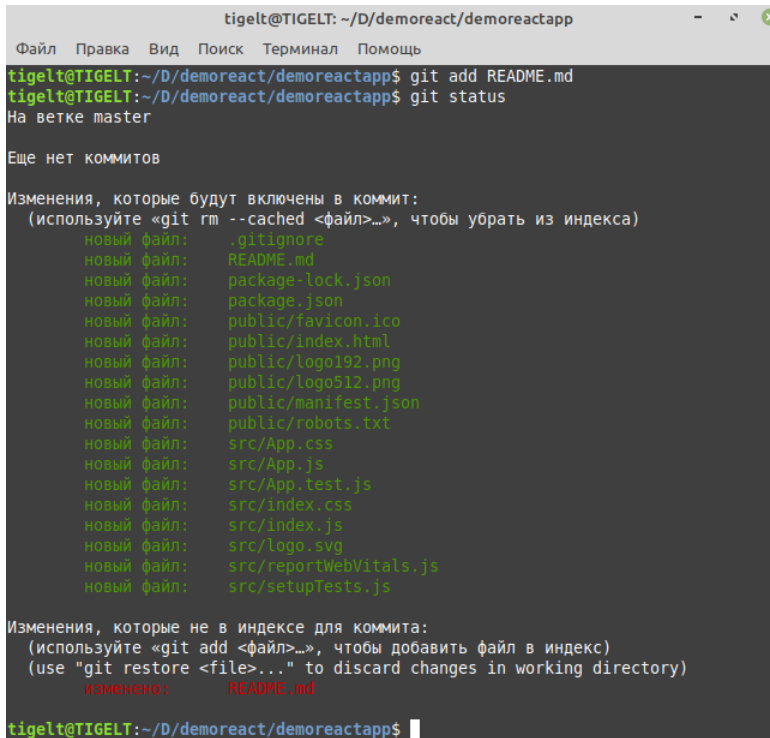
Изменения, которые будут включены в коммит:
(используйте «git rm --cached <файл>...», чтобы убрать из индекса)
    новый файл:   .gitignore
    новый файл:   package-lock.json
    новый файл:   package.json
    новый файл:   public/favicon.ico
    новый файл:   public/index.html
    новый файл:   public/logo192.png
    новый файл:   public/logo512.png
    новый файл:   public/manifest.json
    новый файл:   public/robots.txt
    новый файл:   src/App.css
    новый файл:   src/App.js
    новый файл:   src/App.test.js
    новый файл:   src/index.css
    новый файл:   src/index.js
    новый файл:   src/logo.svg
    новый файл:   src/reportWebVitals.js
    новый файл:   src/setupTests.js

Неотслеживаемые файлы:
(используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
    README.md

tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

В ситуации когда гит отслеживает файл, но в процессе работы мы изменяем его, то гит обратит наше внимание на то, что файл добавлен, но был изменен и его необходимо добавить повторно, это сообщение мы можем увидеть командой (`git status`). Для примера вернем файл `README.md` в отслеживаемые файлы а после изменим его.



```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git add README.md
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master

Еще нет коммитов

Изменения, которые будут включены в коммит:
(используйте «git rm --cached <файл>...», чтобы убрать из индекса)
    новый файл:   .gitignore
    новый файл:   README.md
    новый файл:   package-lock.json
    новый файл:   package.json
    новый файл:   public/favicon.ico
    новый файл:   public/index.html
    новый файл:   public/logo192.png
    новый файл:   public/logo512.png
    новый файл:   public/manifest.json
    новый файл:   public/robots.txt
    новый файл:   src/App.css
    новый файл:   src/App.js
    новый файл:   src/App.test.js
    новый файл:   src/index.css
    новый файл:   src/index.js
    новый файл:   src/logo.svg
    новый файл:   src/reportWebVitals.js
    новый файл:   src/setupTests.js

Изменения, которые не в индексе для коммита:
(используйте «git add <файл>...», чтобы добавить файл в индекс)
(use "git restore <file>..." to discard changes in working directory)
    изменено:   README.md

tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Для того чтобы зафиксировать финальные изменения нужно использовать команду (`git commit -m<сообщение>`). Команда (`-m`) переводится как `message`, название или краткое описание коммита. В консоли появится сообщение в котором сказано, что мы находимся на главной ветке, название коммита и информация о том какие файлы мы закоммитили. Из важного мы можем наблюдать уникальный хэш этого коммита, у каждого коммита он индивидуален и с помощью него мы всегда можем вернуться к какому-то конкретному коммиту.

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git commit -m "First commit"
[master (корневой коммит) 3796725] First commit
18 files changed, 28331 insertions(+)
create mode 100644 .gitignore
create mode 100644 README.md
create mode 100644 package-lock.json
create mode 100644 package.json
create mode 100644 public/favicon.ico
create mode 100644 public/index.html
create mode 100644 public/logo192.png
create mode 100644 public/logo512.png
create mode 100644 public/manifest.json
create mode 100644 public/robots.txt
create mode 100644 src/App.css
create mode 100644 src/App.js
create mode 100644 src/App.test.js
create mode 100644 src/index.css
create mode 100644 src/index.js
create mode 100644 src/logo.svg
create mode 100644 src/reportWebVitals.js
create mode 100644 src/setupTests.js
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Чтобы отслеживать историю изменений мы можем использовать команду (`git log`). На данный момент можем наблюдать что у нас есть один коммит, видим его хэш сумму, автора и короткое сообщение о чем ЭТОТ КОММИТ.

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git log
commit 3796725a80ef98495ad1be195e1614f3d3db9e25 (HEAD -> master)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sat May 21 18:50:22 2022 +0300

    First commit
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Рассмотри многогранную команду (`git show`), в зависимости от того, что ей передать в качестве аргумента она будет менять свое поведение, в целом она показывает какой то объект, показывает его в каких то подробностях. Например если в качестве аргумента ей передать хэш коммита, то она более детально отобразит что это был за коммит.

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git show 3796725a80ef98495ad1be195e1614f3d3db9e25
commit 3796725a80ef98495ad1be195e1614f3d3db9e25 (HEAD -> master)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sat May 21 18:50:22 2022 +0300

    First commit

diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..4d29575
--- /dev/null
+++ b/.gitignore
@@ -0,0 +1,23 @@
+# See https://help.github.com/articles/ignoring-files/ for more about ignoring
files.
+
+# dependencies
+/node_modules
+/.pnp
+/.pnp.js
+
+# testing
+/coverage
+
+# production
+/build
+
+# misc
+.DS_Store
+.env.local
+.env.development.local
+.env.test.local
+.env.production.local

```

Перед тем как продолжить работу с гитом, стоит рассмотреть ситуацию в которой в проекте есть какие то файлы которые не должны попадать в систему контроля версий, для игнорирования таких файлов в корне проекта создается файл с именем (.gitignore). В этом файле мы перечисляем список папок и файлов которые мы не хотим чтобы гит отслеживал. Однако сам файл гитигнор стоит добавить в гит.

Рассмотрим еще одну интересную команду (git restore <имя файла>). Эта команда откатывает изменения в файле на состояние последнего коммита. Для примера изменяем файл README.md, смотрим что гит видит изменения, а затем возвращаем его на состояние последнего коммита и видим что изменений в файле никаких больше нет.

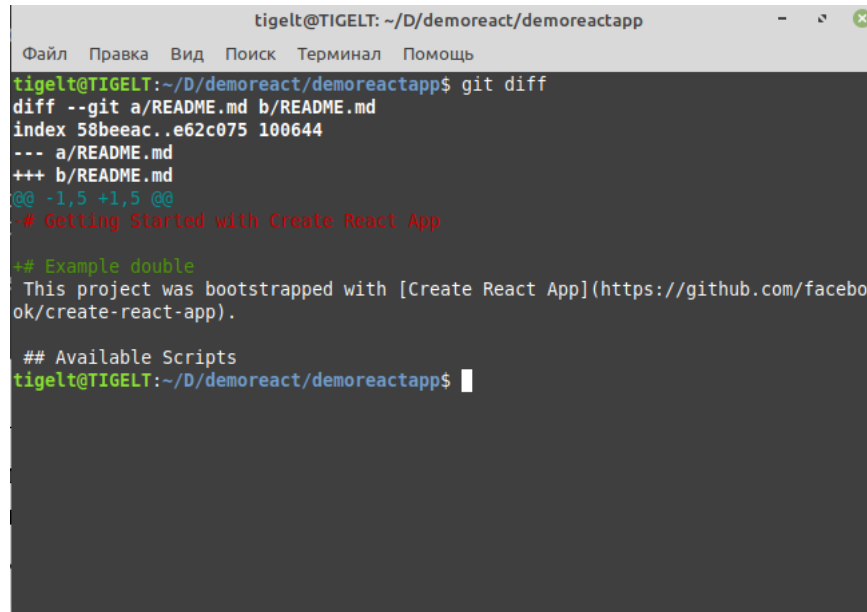
```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>...», чтобы добавить файл в индекс)
  (use "git restore <file>..." to discard changes in working directory)
        изменено:   README.md

нет изменений добавленных для коммита
(используйте «git add» и/или «git commit -a»)
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git restore README.md
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
ничего коммитить, нет изменений в рабочем каталоге
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Еще одна полезная команда называется (git diff), она показывает какие изменения были внесены в файла с момента последнего коммита. Например, вернемся к тому же файлу README.md, изменим его и применяем команду. Получаем сообщение что были совершены изменения, показано где и что именно было добавлено, а что было



```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git diff
diff --git a/README.md b/README.md
index 58beeac..e62c075 100644
--- a/README.md
+++ b/README.md
@@ -1,5 +1,5 @@
-# Getting Started with Create React App
+## Example double
+This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-app).
+
+## Available Scripts
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

удалено.

Стоит обратить внимание на то, что при использовании этой команды гит покажет изменения в файле, который еще не был добавлен в отслеживаемые, чтобы посмотреть какие изменения были сделаны в файлах, которые мы отслеживаем, стоит применить команду (git diff --staged)

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>...», чтобы добавить файл в индекс)
  (use "git restore <file>..." to discard changes in working directory)
        изменено:   README.md

нет изменений добавленных для коммита
(используйте «git add» и/или «git commit -a»)
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git diff
diff --git a/README.md b/README.md
index 58beeac..e62c075 100644
--- a/README.md
+++ b/README.md
@@ -1,5 +1,5 @@
-# Getting Started with Create React App

+# Example double
This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-app).

## Available Scripts
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git add .
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
Изменения, которые будут включены в коммит:
  (use "git restore --staged <file>..." to unstage)
        изменено:   README.md

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git diff
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git diff --staged
diff --git a/README.md b/README.md
index 58beeac..e62c075 100644
--- a/README.md
+++ b/README.md
@@ -1,5 +1,5 @@
-# Getting Started with Create React App

+# Example double
This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-app).

## Available Scripts
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Чтобы узнать в какой ветке мы находимся на данный момент стоит использовать команду (`git branch`). Чтобы не мешать другим разработчикам мы должны создать свою ветку и работать в ней, для этого существует команда (`git branch <имя ветки>`). Если же мы случайно создали ветку или она нам больше не нужна используется команда (`git branch -D <имя ветки>`)


```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch -a
* master
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch bagfix
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch
bagfix
* master
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch baggg
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch
bagfix
baggg
* master
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch -D baggg
Ветка baggg удалена (была 49eff0e).
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch
bagfix
* master
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Для того чтобы работать в новой ветке недостаточно просто создать её, после создания необходимо переключиться на новую ветку командой (`git checkout <имя ветки>`). При работе с гитом довольно распространенная ситуация когда мы часто создаем ветки или переключаемся на другие, но есть команда которая позволяет сделать это гораздо быстрее, это команда (`git checkout -b <имя ветки>`). Это команда означает что мы создаем новую ветку и сразу же на неё переключаемся.

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь
Ветка baggg удалена (была 49eff0e).
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git branch
bagfix
* master
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git checkout -d dev
fatal: git checkout: --detach не принимает путь «dev» как аргумент
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git checkout -b dev
Переключено на новую ветку «dev»
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git log
commit 49eff0eb85991965ef561d02148bab8418dd3ea7 (HEAD -> dev, master, bagfix)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sat May 21 19:38:56 2022 +0300

    Second commit

commit 3796725a80ef98495ad1be195e1614f3d3db9e25
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sat May 21 18:50:22 2022 +0300

    First commit
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status

```

Для того чтоб проиллюстрировать всю прелесть гита создадим в новой ветке файл, добавим и закоммитим его, затем посмотрим статус. Можем заметить что сейчас мы находимся не на главной ветке и у нас есть новые файлы, если же мы переключимся на основную ветку, то увидим что файл, который мы только что создали отсутствует, это и правильно ведь он находится в другой ветке. То есть сейчас у нас есть две разные версии нашего проекта и нам их нужно как то соединить, для этого нужно перейти в основную ветку и применить команду (`git merge`

<имя ветки>). После применения этой команды наблюдаем сообщение, в котором говорится что мы добавили новый файл, видим хэш.

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git add .
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке dev
Изменения, которые будут включены в коммит:
  (use "git restore --staged <file>..." to unstage)
        новый файл:   dev.js

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git commit -m "add dev.js"
[dev 7cf4d36] add dev.js
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 dev.js
tigelt@TIGELT:~/D/demoreact/demoreactapp$ ls
dev.js  node_modules  package.json  package-lock.json  public  README.md  src
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git checkout master
Переключено на ветку «master»
tigelt@TIGELT:~/D/demoreact/demoreactapp$ ls
node_modules  package.json  package-lock.json  public  README.md  src
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git merge dev
Обновление 49eff0e..7cf4d36
Fast-forward
 dev.js | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 dev.js
tigelt@TIGELT:~/D/demoreact/demoreactapp$ ls
dev.js  node_modules  package.json  package-lock.json  public  README.md  src
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Дальше настало время поработать с гитхабом, важно понимать что гит это локальная система, а гитхаб это удаленная система, некоторый интерфейс, который позволяет визуально показывать что локально происходит с проектом и его версиями. Соответственно чтобы воспользоваться гитхабом, необходимо перейти на официальный сайт и завести там аккаунт, после чего создать там репозиторий по инструкции на самом сервисе. Далее необходимо вернуться к нашему локальному гиту и проинициализировать себя, для того чтобы связать гитхаб и наш локальный гит нам понадобится ссылка на наш репозиторий, скопировав ссылку нужно применить команду (`git remote add origin <ссылка>`). Эта команда управляет удаленными серверами. Для того чтобы посмотреть какие у нас есть репозитории нужно выполнить команду (`git remote -v`)

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git remote add origin https://github.com/Tigelt/DiplRep.git
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git remote -v
origin  https://github.com/Tigelt/DiplRep.git (fetch)
origin  https://github.com/Tigelt/DiplRep.git (push)
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Теперь мы можем перенести наш локальный репозиторий на удалённый при помощи команды (`git push origin master`). После применения этой команды мы получаем сообщения, в которых приводится системная информация и предоставляется адрес по которому был залит наш репозиторий. Если мы сейчас обновим наш удаленный репозиторий, то увидим, что наш локальный репозиторий появился в удаленном. В гитхабе мы можем обратить внимание на то, что есть вкладка просмотра всех коммитов, там указаны все коммиты и

изменения которые были сделаны в них, то есть каждый шаг был зафиксирован.

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git remote add origin git@github.com:T
tigelt/DiplRep.git
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git push -u origin master
The authenticity of host 'github.com (140.82.121.4)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeI0ttrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,140.82.121.4' (ECDSA) to the list of know
n hosts.
Перечисление объектов: 28, готово.
Подсчет объектов: 100% (28/28), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (27/27), готово.
Запись объектов: 100% (28/28), 288.46 KiB | 2.94 MiB/s, готово.
Всего 28 (изменения 3), повторно использовано 0 (изменения 0)
remote: Resolving deltas: 100% (3/3), done.
To github.com:Tigelt/DiplRep.git
 * [new branch]      master -> master
Ветка «master» отслеживает внешнюю ветку «master» из «origin».
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Если сейчас мы внесем изменения в локальный проект, например удалим файл `dev.js`, то они не появятся сами по себе в гитхабе, для этого нужно посмотреть статус локального проекта, увидеть список файлов в которых произошли изменения, добавляем эти файлы, проверяем статус, видим что гит отслеживает измененные файлы, после чего нужно сделать коммит, а теперь можно заливать эти изменения на гитхаб командой (`git push`).

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
Ваша ветка обновлена в соответствии с «origin/master».

Изменения, которые не в индексе для коммита:
  (используйте «git add/rm <файл>...», чтобы добавить или удалить файл из индекса)
  (use "git restore <file>..." to discard changes in working directory)
    удалено:      dev.js

нет изменений добавленных для коммита
(используйте «git add» и/или «git commit -a»)
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git add .
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
Ваша ветка обновлена в соответствии с «origin/master».

Изменения, которые будут включены в коммит:
  (use "git restore --staged <file>..." to unstage)
    удалено:      dev.js

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git commit -m "delete devjs"
[master e8ddae2] delete devjs
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 dev.js
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
Ваша ветка опережает «origin/master» на 1 коммит.
  (используйте «git push», чтобы опубликовать ваши локальные коммиты)

ничего коммитить, нет изменений в рабочем каталоге
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git push
Перечисление объектов: 3, готово.
Подсчет объектов: 100% (3/3), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (2/2), готово.
Запись объектов: 100% (2/2), 227 bytes | 227.00 KiB/s, готово.
Всего 2 (изменения 1), повторно использовано 0 (изменения 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:Tigelt/DiplRep.git
 7cf4d36..e8ddae2  master -> master
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

В ветке мастер как правило хранится протестированный стабилизированный рабочий код, иногда у разработчика есть потребность зафиксировать некоторые моменты разработки, например релиз, так вот в гите есть такой механизм, который называется (git tag) метки. По сути мы можем оставить на какомнибудь коммите метку. Есть два типа тегов: аннотированные и легковесные. Легковесные по сути представляют собой указатель, имя которого выбирает сам разработчик. Аннотированные же теги это полноценные объекты в гите, у них есть хэш сумма, автор тега, дата и так далее.

Для того, чтобы создать легковесный тег на текущем коммите достаточно всего лишь написать команду (git tag <имя тега>), а вот для того чтобы создать аннотированный тег нужно написать команду (git tag -a <имя тега> -m "сообщение").

Для того чтобы посмотреть какие вообще у нас есть теги, нужно выполнить команду (git tag). Важно сказать, что когда мы делаем пуш наших изменений теги по умолчанию не отправляются на сервер, для того чтобы отправить их на удаленный сервер нужно сделать это явно, то есть использовать команду (git push origin <имя тега>) или же можно

отправить на сервер вообще все теги какие у нас есть прописав команду (git push --tags)

```

tigelt@TIGELT: ~/D/demoreact/demoreactapp
Файл  Правка  Вид  Поиск  Терминал  Помощь
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git tag v1
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git tag -a R2022.1 -m "Release 2022.1"
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git tag
R2022.1
v1
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git push origin v1
Warning: Permanently added the ECDSA host key for IP address '140.82.121.3' to t
he list of known hosts.
Всего 0 (изменения 0), повторно использовано 0 (изменения 0)
To github.com:Tigelt/DiplRep.git
 * [new tag]          v1 -> v1
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git push --tags
Перечисление объектов: 1, готово.
Подсчет объектов: 100% (1/1), готово.
Запись объектов: 100% (1/1), 171 bytes | 171.00 KiB/s, готово.
Всего 1 (изменения 0), повторно использовано 0 (изменения 0)
To github.com:Tigelt/DiplRep.git
 * [new tag]          R2022.1 -> R2022.1
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

Что ж, на данный момент у нас имеется проект на удаленном репозитории, для того чтобы другой разработчик мог перенести проект к себе на локальный репозиторий ему понадобится ссылка на этот репозиторий. Смоделируем работу двух разработчиков открыв второе окно терминала, затем введем следующую команду (git clone <ссылка>), после чего у нас появляется папка с названием репозитория, в которой те же самые коммиты, тот же самый код.

```

tigelt@TIGELT:~/D/demoreact/demodemoreact$ git clone git@github.com:Tigelt/DiplRep.git
Клонирование в «DiplRep»...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 30 (delta 4), reused 29 (delta 3), pack-reused 0
Получение объектов: 100% (30/30), 288.69 KiB | 771.00 KiB/s, готово.
Определение изменений: 100% (4/4), готово.
tigelt@TIGELT:~/D/demoreact/demodemoreact$ ls
DiplRep
tigelt@TIGELT:~/D/demoreact/demodemoreact$ cd DiplRep
bash: cd: DiplRep: Нет такого файла или каталога
tigelt@TIGELT:~/D/demoreact/demodemoreact$ cd DiplRep
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ ls
package.json  package-lock.json  public  README.md  src
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$

```

Второй разработчик может внести какие то изменения в проект, для примера создадим файл devdev.js, закоммитим его, и запустим изменения на сервер.

```

tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git status
На ветке master
Ваша ветка обновлена в соответствии с «origin/master».

Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
    devdev.js

ничего не добавлено в коммит, но есть неотслеживаемые файлы (используйте «git add», чтобы отслеживать их)
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git add .
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git commit -m "add devdev"
[master 655f134] add devdev
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 devdev.js
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git push
Перечисление объектов: 4, готово.
Подсчет объектов: 100% (4/4), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (2/2), готово.
Запись объектов: 100% (3/3), 267 bytes | 133.00 KiB/s, готово.
Всего 3 (изменения 1), повторно использовано 1 (изменения 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:Tigelt/DiplRep.git
 e8ddae2..655f134 master -> master
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$

```

На данный момент на сервере произошли изменения, второй разработчик сделал коммит и добавил новый файл в проект. Для того, чтобы второй разработчик получил эти правки ему необходимо выполнить команду (`git pull`), эта команда забирает изменения с удаленного сервера и передает их в наш локальный репозиторий.


```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git pull
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Распаковка объектов: 100% (3/3), 247 bytes | 8.00 KiB/s, готово.
Из github.com:Tigelt/DiplRep
e8ddae2..655f134 master -> origin/master
Обновление e8ddae2..655f134
Fast-forward
 devdev.js | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 devdev.js
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git log
commit 655f134c23d0502c3a5a6208b449314eb12e1611 (HEAD -> master, origin/master)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sun May 22 17:03:27 2022 +0300

    add devdev

commit e8ddae27e9555c8dbfd930f73bbe691d39798103 (tag: v1, tag: R2022.1)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sun May 22 15:11:53 2022 +0300

    delete devjs

commit 7cf4d36a59d705fe523ca8e7b4d85f656b17047f (dev)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sat May 21 20:29:25 2022 +0300

    add dev.js

commit 49eff0eb85991965ef561d02148bab8418dd3ea7 (bagfix)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sat May 21 19:38:56 2022 +0300

    Second commit

commit 3796725a80ef98495ad1be195e1614f3d3db9e25
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date: Sat May 21 18:50:22 2022 +0300

    First commit
tigelt@TIGELT:~/D/demoreact/demoreactapp$

```

В данном примере разработчики работали последовательно, у них у обоих на компьютере был одинаковый репозиторий и коммит указывал на ветку мастер, но что будет если у одного будут свои коммиты, а у другого свои, и они решат одновременно внести свои правки на сервер.

Пусть первый разработчик внесет изменения в новосозданный файл devdev.js, а второй сделает изменения в файле README.md. Первый разработчик без проблем зашьет свои изменения на сервер и история коммитов будет выглядеть вот так:

```

tigelt@TIGELT:~/D/demoreact/demoreactapp$ git status
На ветке master
Ваша ветка обновлена в соответствии с «origin/master».

Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>...», чтобы добавить файл в индекс)
  (use "git restore <file>..." to discard changes in working directory)
        изменено:   devdev.js

нет изменений добавленных для коммита
(используйте «git add» и/или «git commit -a»)
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git add .
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git commit -m "console"
[master ff13b30] console
1 file changed, 1 insertion(+)
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git push
Перечисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (2/2), готово.
Запись объектов: 100% (3/3), 275 bytes | 275.00 KiB/s, готово.
Всего 3 (изменения 1), повторно использовано 0 (изменения 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:Tigelt/DiplRep.git
   655f134..ff13b30  master -> master
tigelt@TIGELT:~/D/demoreact/demoreactapp$ git log
commit ff13b3093e0cb33d4eddfeb88f302edf3d2b5f18 (HEAD -> master, origin/master)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date:   Sun May 22 17:14:34 2022 +0300

    console

commit 655f134c23d0502c3a5a6208b449314eb12e1611
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date:   Sun May 22 17:03:27 2022 +0300

    add devdev

commit e8ddae27e9555c8dbfd930f73bbe691d39798103 (tag: v1, tag: R2022.1)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date:   Sun May 22 15:11:53 2022 +0300

    delete devjs

commit 7cf4d36a59d705fe523ca8e7b4d85f656b17047f (dev)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date:   Sat May 21 20:29:25 2022 +0300

    add dev.js

commit 49eff0eb85991965ef561d02148bab8418dd3ea7 (bagfix)
Author: Ivan Elistratov <tigelt2014@gmail.com>
Date:   Sat May 21 19:38:56 2022 +0300

    Second commit

commit 3796725a80ef98495ad1be195e1614f3d3db9e25

```

Теперь пусть второй разработчик попробует залить свои изменения на сервер. При попытке сделать это он получит сообщение об ошибке. Сам гит пишет о том, что наше обновление было отменено, потому что сервер содержит правки, которых у нас нету локально и предлагает использовать команду (git pull). После этого, нам будет предложено написать комментарий к коммиту, то есть гит создает коммит слияния

наших веток и уже только после этого может запустить наши изменения на сервер.

```

tigelt@TIGELT: ~/D/demoreact/demodemoreact/DiplRep
Файл  Правка  Вид  Поиск  Терминал  Помощь

tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git status
На ветке master
Ваша ветка обновлена в соответствии с «origin/master».

Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>...», чтобы добавить файл в индекс)
  (use "git restore <file>..." to discard changes in working directory)
        изменено:   README.md

нет изменений добавленных для коммита
(используйте «git add» и/или «git commit -a»)
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git add .
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git commit -m "prav2README"
[master eb93ae6] prav2README
1 file changed, 1 insertion(+), 1 deletion(-)
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git push
To github.com:Tigelt/DiplRep.git
 ! [rejected]        master -> master (fetch first)
error: не удалось отправить некоторые ссылки в «git@github.com:Tigelt/DiplRep.git»
подсказка: Обновления были отклонены, так как внешний репозиторий содержит
подсказка: изменения, которых у вас нет в вашем локальном репозитории.
подсказка: Обычно, это связано с тем, что кто-то уже отправил изменения в
подсказка: то же место. Перед повторной отправкой ваших изменений, вам нужно
подсказка: заорать и слить изменения из внешнего репозитория себе
подсказка: (например, с помощью «git pull ...»).
подсказка: Для дополнительной информации, смотрите «Note about fast-forwards»
подсказка: в «git push --help».
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Распаковка объектов: 100% (3/3), 255 bytes | 2.00 KiB/s, готово.
Из github.com:Tigelt/DiplRep
655f134..ff13b30 master -> origin/master
Merge made by the 'recursive' strategy.
 devdev.js | 1 +
1 file changed, 1 insertion(+)
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (8/8), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (5/5), готово.
Запись объектов: 100% (5/5), 560 bytes | 560.00 KiB/s, готово.
Всего 5 (изменения 3), повторно использовано 0 (изменения 0)
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:Tigelt/DiplRep.git
 ff13b30..dd1a874 master -> master
tigelt@TIGELT:~/D/demoreact/demodemoreact/DiplRep$

```

2. Особенности и принципы использования Single-page application приложений.

ВЫВОД ПО ГЛАВЕ 2.

Обобщая методику использования системы контроля версий, а в частности гита, стоит отметить не какие то жесткие требования, а некоторые практики, которых стоит придерживаться при работе.

Во первых что касается работы в ветках, не стоит работать в ветке мастер, если нам нужно сделать какой то новый функционал, стоит отвести для этого отдельную ветку, закоммитить туда свои правки, если нужно сделать несколько коммитов, протестировать то что у вас получилось, а потом уже стоит делать сливать нашу ветку с основной.

Во вторых что касается сообщений сопровождающих коммит, их стоит писать императивно, как можно короче, но при этом, чтобы они содержали в себе как можно больше информации, это стоит делать для того, чтобы при просмотре коммитов разработчику было понятно, что конкретно было сделано в данном коммите, не открывая сам коммит и не просматривая сами правки.

В третьих, в истории коммитов стоит придерживаться линейности.

ЗАКЛЮЧЕНИЕ.

СПИСОК ЛИТЕРАТУРЫ.

1. Кизим, С.С. ПРИМЕНЕНИЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ ОРГАНИЗАЦИИ ПРАКТИЧЕСКИХ ЗАНЯТИЙ ПО ДИСЦИПЛИНЕ "ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ" / Инновационные технологии в науке и образовании. 2015. № 1 (1). С. 264. (<https://www.elibrary.ru/item.asp?id=24916590>)
2. СЕЙДАМЕТОВ, С.В. РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ: ПРОБЛЕМЫ НАЧИНАЮЩИХ РАЗРАБОТЧИКОВ / Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2019. № 3 (25). С. 39-44. (<https://www.elibrary.ru/item.asp?id=41829772>)
3. Мешков Д.А., Стариков А.В. ПРИМЕНЕНИЕ СИСТЕМЫ УПРАВЛЕНИЯ И КОНТРОЛЯ ВЕРСИЙ GIT В РАСПРЕДЕЛЕННОЙ СРЕДЕ ПРОЕКТИРОВАНИЯ / Курск, статья в сборнике трудов конференции 2014. - 291-293 с. (<https://www.elibrary.ru/item.asp?id=22849587>)

4. Фролова, Ж.Е. СИСТЕМА КОНТРОЛЯ ВЕРСИЙ GIT В ПОДГОТОВКЕ ИТ СПЕЦИАЛИСТОВ - ОСОЗНАННАЯ НЕОБХОДИМОСТЬ /Сборники конференций НИЦ Социосфера. 2021. № 10. С. 206-208.
(<https://www.elibrary.ru/item.asp?id=44869173>)
5. Сейдаметов С.В. РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ: ПРОБЛЕМЫ НАЧИНАЮЩИХ РАЗРАБОТЧИКОВ / Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2019. № 3 (25). С. 39-44.
(<https://www.elibrary.ru/item.asp?id=41829772>)
6. Рындина А.С. ПРИМЕНЕНИЕ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ GIT ПРИ РАЗРАБОТКЕ ПРОГРАММНОГО ПРОДУКТА / Вестник современных исследований. 2017. № 3 (6). С. 85-87.
(<https://www.elibrary.ru/item.asp?id=28897270>)
7. Будилов В.Н., Романов А.А. ВАЖНОСТЬ ИСПОЛЬЗОВАНИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ / Молодой ученый. 2021. № 24 (366). С. 9-11.
(<https://www.elibrary.ru/item.asp?id=46183469>)
8. Сычев О.А., Терехов Г.В. СОВРЕМЕННЫЕ СИСТЕМЫ РАСПРЕДЕЛЕННОГО КОНТРОЛЯ ВЕРСИЙ / Волгоград, 2018. - 64 с. (<https://www.elibrary.ru/item.asp?id=36392617>)
9. Кузьмин Д.С., РЕАЛИЗАЦИЯ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ СТУДЕНЧЕСКИХ ПРОЕКТОВ В ВЕБ-ПРИЛОЖЕНИИ / В сборнике: Интеллектуальные информационные системы. Труды Всероссийской конференции с международным участием. 2017. С. 34-37. (<https://www.elibrary.ru/item.asp?id=32787232>)
10. Зикратов И.А., Спивак А.И., Разумовский А.В. МЕТОД РАЗГРАНИЧЕНИЯ ДОСТУПА К ОБЪЕКТАМ В СИСТЕМЕ КОНТРОЛЯ ВЕРСИЙ / Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. 2012. № 4 (152). С. 94-96.
(<https://www.elibrary.ru/item.asp?id=17925955>)
11. Пилипенко А.В. АНАЛИЗ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ GIT / В сборнике: Студенческая наука для развития информационного общества. сборник материалов VI

- Всероссийской научно-технической конференции. 2017. С. 205-207. (<https://www.elibrary.ru/item.asp?id=29370182>)
12. Мильчаков А.Е. СИСТЕМА КОНТРОЛЯ ВЕРСИЙ / В сборнике: Информационные системы и технологии в образовании, науке и бизнесе (ИСИТ-2014). Материалы Всероссийской молодежной научно-практической школы. 2014. С. 224-225. (<https://www.elibrary.ru/item.asp?id=22672608>)
13. Копий А.А., Солодко А.А. ПРИМЕНЕНИЕ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ В МОДЕЛИРОВАНИИ БИЗНЕС-ПРОЦЕССОВ / Научный электронный журнал Меридиан. 2020. № 5 (39). С. 33-35. (<https://www.elibrary.ru/item.asp?id=42372978>)
14. Фадеев И.В., Фадеева Т.А. УСТАНОВКА СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ GIT / Экономика и социум. 2016. № 6-2 (25). С. 817-818. (https://www.elibrary.ru/query_results.asp)
15. Картамышев С.В. СИСТЕМЫ УПРАВЛЕНИЯ РЕПОЗИТОРИЯМИ ПРОЕКТОВ НА ОСНОВЕ РАСПРЕДЕЛЕННОЙ СХЕМЫ КОНТРОЛЯ ВЕРСИЙ / В сборнике: Международная научно-техническая конференция молодых ученых БГТУ им. В.Г. Шухова. Белгородский государственный технологический университет им. В.Г. Шухова. 2015. С. 2856-2860. (<https://www.elibrary.ru/item.asp?id=24616317>)
16. Боголепов А.С. АКТУАЛЬНЫЕ ВОПРОСЫ ОБУЧЕНИЯ ПОЛЬЗОВАТЕЛЕЙ РАСПРЕДЕЛЕННОЙ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ GIT / В книге: . Сборник статей и тезисов студенческой открытой конференции. 2020. С. 356-357. (<https://www.elibrary.ru/item.asp?id=42464560>)
17. Рындина А.С. ВЫБОР СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА / В сборнике: Научное сообщество студентов. Междисциплинарные исследования. Электронный сборник статей по материалам XIX студенческой международной научно-практической конференции. 2017. С. 76-81. (<https://www.elibrary.ru/item.asp?id=29048515>)
18. Кустов А.А. ОСОБЕННОСТИ ИСПОЛЬЗОВАНИЯ СИСТЕМ КОНТРОЛЯ ВЕРСИЙ И УПРАВЛЕНИЯ СОВМЕСТНОЙ РАЗРАБОТКИ В ОТДЕЛЕ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ / Научный электронный журнал Меридиан. 2019.

№ 11 (29). С. 192-194.

(<https://www.elibrary.ru/item.asp?id=41435265>)

19. Николаева О.В., Трофимов И.Е. СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ: ЗАЧЕМ ОНИ НУЖНЫ РАЗРАБОТЧИКАМ? / В сборнике: Материалы Всероссийской молодежной конференции "Информационно-телекоммуникационные системы и технологии (ИТСиТ-2012)". 2012. С. 139-140.

(https://www.elibrary.ru/query_results.asp)

20. Альтман Е.А., Александров А.В., Васеева Т.В. СИСТЕМА КОНТРОЛЯ ВЕРСИЙ GIT / Учебно-методическое пособие к выполнению лабораторных работ / Омск, 2021.

(<https://www.elibrary.ru/item.asp?id=45616281>)