МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

(ФГБОУ ВО «ВГТУ»)

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование кафедры)

**ОТЧЕТ**

о прохождении \_\_\_\_\_\_\_ производственной \_\_\_\_\_\_\_\_\_\_\_\_ практики

(учебной, производственной, преддипломной и др.)

с «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ по «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_в 20 \_\_\_/ \_\_\_ учебном году

Место прохождения практики \_\_\_\_\_\_ АО «Концерн Созвездие»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование организации)

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата представления отчёта на кафедру (Подпись)

Факультет  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** ФИТКБ**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(наименование факультета)

Наименование (код) специальности или направления подготовки\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Курс \_\_\_\_\_\_\_\_\_\_\_, группа \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тема \_Децентрализованные системы контроля версий, как инструмент обеспечения целостности информации при разработке программно-определяемых телекоммуникационных систем связи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики от предприятия \_\_\_\_\_Болгов А.А.\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, Ф.И.О.) (Подпись)

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_\_г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата аттестации) оценка, полученная при аттестации (при защите отчёта)

Руководитель практики от кафедры \_\_\_\_ Поздышева О.В.\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, Ф.И.О.) (Подпись)

Воронеж 2022

Задание руководителя

Получить практический навык использования децентрализованной системы контроля версий, как инструмента для обеспечения целостности информации при разработке программно-определяемых телекоммуникационных систем связи.

Для этого требуется выполнить ряд задач:

1. Используя различные информационные ресурсы, ознакомиться с децентрализованными системами контроля версий. Выделить основные преимущества и недостатки в отличии от централизованных систем контроля версий.

2. В качестве децентрализованной системы контроля версий выбрать GIT и провести обзор данного инструмента (что это такое, с какой целью используется, где применяется).

3. Провести анализ возможных уязвимостей децентрализованной системы контроля версий GIT.

4. Выделить основные свойства и функции децентрализованной системы контроля версий GIT, которые позволяют обеспечивать целостность и доступность информации.

5. Пройти онлайн курс в сети интернет, для ознакомления с основными возможностями и командами для работы в GIT.

6. Используя полученные навыки, вести работу над отчетом по практике с применением GIT.

Замечания руководителя

• Обратите внимание на сформулированные задачи. Необходимо поправить работу в соответствии с задачами;

• Ваше введение не годится. Смотрите пример отчета, который я ранее вам скидывал;

• Проверьте текст на отсутствие грамматических и пунктуационных ошибок;

• Придумайте какую-нибудь ситуацию, при которой вам пришлось вернуться к старой фиксации и начать работу в новой ветке GIT, чтобы не потерять последнюю фиксацию.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc105949372)

[1. Общие сведения о распределённой системе управления версиями Git. 7](#_Toc105949373)

[1.1 Основные сведения 7](#_Toc105949374)

[1.2 Особенности реализации 7](#_Toc105949375)

[1.3 Архитектура 8](#_Toc105949376)

[1.4 Сетевые возможности и серверные решения 11](#_Toc105949377)

[2. Использование GIT в написании отчета 13](#_Toc105949378)

[2.1 Начало работы над новым проектом с помощью GIT 13](#_Toc105949379)

[2.2 Работа над существующим проектом с помощью GIT 13](#_Toc105949380)

[2.3 Создание новой ветки и работа в нем 15](#_Toc105949381)

[3. Уязвимости в GIT 18](#_Toc105949382)

[3.1 Уязвимость CVE-2022-24765 19](#_Toc105949383)

[3.2 Уязвимость CVE-2022-24765 19](#_Toc105949384)

[ЗАКЛЮЧЕНИЕ 21](#_Toc105949385)

[СПИСОК ЛИТЕРАТУРЫ 22](#_Toc105949386)

# ВВЕДЕНИЕ

Git — это гибкая, распределенная система контроля версий, дающая массу возможностей не только разработчикам программных продуктов, но и писателям для изменения, дополнения и отслеживания изменения «рукописей» и сюжетных линий, и учителям для корректировки и развития курса лекций, и администраторам для ведения документации, и для многих других направлений, требующих управления историей изменений.

Git, на фоне других СКВ, выделяется главным образом из-за того, что он применяется для разработки ядра Linux.

Собственно, именно для этого Git и разрабатывался. Ядро Linux — весьма немаленький проект, как по объему исходного кода, так и по числу участников, поэтому при его разработке большое внимание было уделено производительности и легкости слияния веток.

У каждого разработчика, использующего Git, есть свой локальный репозиторий, позволяющий локально управлять версиями. Затем, сохраненными в локальный репозиторий данными, можно обмениваться с другими пользователями.

Часто при работе с Git создают центральный репозиторий, с которым остальные разработчики синхронизируются. Пример организации системы с центральным репозиторием — это проект разработки ядра Linux.

В этом случае все участники проекта ведут свои локальны разработки и беспрепятственно скачивают обновления из центрального репозитория. Когда необходимые работы отдельными участниками проекта выполнены и отлажены, они, после удостоверения владельцем центрального репозитория в корректности и актуальности проделанной работы, загружают свои изменения в центральный репозиторий.

Наличие локальных репозиторием также значительно повышает надежность хранения данных, так как, если один из репозиториев выйдет из строя, данные могут быть легко восстановлены из других репозиториев.

Работа над версиями проекта в Git может вестись в нескольких ветках, которые затем могут с легкостью полностью или частично объединяться, уничтожаться, откатываться и разрастаться во все новые и новые ветки проекта.

# 1. Общие сведения о распределённой системе управления версиями Git.

## 1.1 Основные сведения

Git — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано [1].

Среди проектов, использующих Git — ядро Linux, Swift, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki, DokuWiki, Qt, ряд дистрибутивов Linux.

Программа является свободной и выпущена под лицензией GNU GPL версии 2. По умолчанию используется TCP порт 9418.

## 1.2 Особенности реализации

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и даёт возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом) [1].

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов, и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создаётся рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit) [1].

## 1.3 Архитектура

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне. Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git своё приложение.

Для каждого объекта в репозитории вычисляется SHA-1-хеш, и именно он становится именем файла, содержащего данный объект в каталоге .git/objects. Для оптимизации работы с файловыми системами, не использующими деревья для каталогов, первый байт хеша становится именем подкаталога, а остальные — именем файла в нём, что снижает количество файлов в одном каталоге (ограничивающий фактор производительности на таких устаревших файловых системах).

Все ссылки на объекты репозитория, включая ссылки на один объект, находящийся внутри другого объекта, являются SHA-1-хешами.

Кроме того, в репозитории существует каталог refs, который позволяет задать читаемые человеком имена для каких-то объектов Git. В командах Git оба вида ссылок — читаемые человеком из refs, и нижележащие SHA-1 — полностью взаимозаменяемы [1].

В классическом обычном сценарии в репозитории git есть три типа объектов — файл, дерево и «коммит» (англ. commit — фиксация). Файл есть какая-то версия какого-то пользовательского файла, дерево — совокупность файлов из разных подкаталогов, «коммит» — дерево и некая дополнительная информация (например, родительские коммиты, а также комментарий).

В репозитории иногда производится сборка мусора, во время которой устаревшие файлы заменяются на «дельты» между ними и актуальными файлами (то есть, актуальная версия файла хранится неинкрементально, инкременты используются только для возврата к предыдущим версиям), после чего данные «дельты» складываются в один большой файл, к которому строится индекс. Это снижает требования по ёмкости хранения.

Репозиторий Git бывает локальный и удалённый. Локальный репозиторий — это подкаталог .git, создаётся (в пустом виде) командой git init и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой git clone [1].

Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием. Удалённый репозиторий можно только синхронизировать с локальным как «вверх» (push), так и «вниз» (pull).

Наличие полностью всего репозитория проекта локально у каждого разработчика даёт Git ряд преимуществ перед SVN. Так, например, все операции, кроме push и pull, можно осуществлять без наличия интернет-соединения.

Очень мощной возможностью git являются ветви, реализованные куда более полно, чем в SVN: по сути, ветвь git есть не более чем именованная ссылка, указывающая на некий коммит в репозитории (используется подкаталог refs). Коммит без создания новой ветви всего лишь передвигает эту ссылку на себя, а коммит с созданием ветви — оставляет старую ссылку на месте, но создаёт новую на новый коммит, и объявляет её текущей. Заменить локальные девелоперские файлы на набор файлов из иной ветви, тем самым перейдя к работе с ней — так же тривиально [1].

Также поддерживаются субрепозитории с синхронизацией текущих ветвей в них.

Команда push передаёт все новые данные (те, которых ещё нет в удалённом репозитории) из локального репозитория в репозиторий удалённый. Для исполнения этой команды необходимо, чтобы удалённый репозиторий не имел новых коммитов в себя от других клиентов, иначе push завершается ошибкой, и придётся делать pull и слияние.

Команда pull — обратна команде push. В случае, если одна и та же ветвь имеет независимую историю в локальной и в удалённой копии, pull немедленно переходит к слиянию.

Слияние в пределах разных файлов осуществляется автоматически (всё это поведение настраивается), а в пределах одного файла — стандартным трёхпанельным сравнением файлов. После слияния нужно объявить конфликты как разрешённые [1].

Результатом всего этого является новое состояние в локальных файлах у того разработчика, что осуществил слияние. Ему нужно немедленно сделать коммит, при этом в данном объекте коммита в репозитории окажется информация о том, что коммит есть результат слияния двух ветвей и имеет два родительских коммита.

Кроме слияния, Git поддерживает ещё операцию перемещения (англ. rebase). Эта операция есть получение набора всех изменений в ветви А, с последующим их «накатом» на ветвь B. В результате ветвь B продвигается до состояния AB. В отличие от слияния, в истории ветви AB не останется никаких промежуточных коммитов ветви A (только история ветви B и запись о самом rebase, это упрощает интеграцию крупных и очень крупных проектов).

Также Git имеет временный локальный индекс файлов. Это — промежуточное хранилище между собственно файлами и очередным коммитом (коммит делается только из этого индекса). С помощью этого индекса осуществляется добавление новых файлов (git add добавляет их в индекс, они попадут в следующий коммит), а также коммит не всех изменённых файлов (коммит делается только тем файлам, которым был сделан git add). После git add можно редактировать файл далее, получатся три копии одного и того же файла — последняя, в индексе (та, что была на момент git add), и в последнем коммите.

Имя ветви по умолчанию: master. Имя удалённого репозитория по умолчанию, создаваемое git clone во время типичной операции «взять имеющийся проект с сервера себе на машину»: origin.

Таким образом, в локальном репозитории всегда есть ветвь master, которая есть последний локальный коммит, и ветвь origin/master, которая есть последнее состояние удалённого репозитория на момент завершения исполнения последней команды pull или push [1].

Команда fetch (частичный pull) — берёт с удалённого сервера все изменения в origin/master, и переписывает их в локальный репозиторий, продвигая метку origin/master.

Если после этого master и origin/master разошлись в стороны, то необходимо сделать слияние, установив master на результат слияния (команда pull есть fetch+merge). Далее возможно сделать push, отправив результат слияния на сервер и установив на него origin/master.

## 1.4 Сетевые возможности и серверные решения

Git использует сеть только для операций обмена с удалёнными репозиториями.

Возможно использование следующих протоколов:

* git-протокол (схема URI — git:) — открытый протокол [12], требующий наличия на сервере запущенного git-демона[13] (поставляется вместе с Git), протокол не имеет средств аутентификации пользователей;
* SSH (ssh:) — использует аутентификацию пользователей с помощью пар ключей, а также встроенный в Unix-систему «основной» SSH-сервер (sshd), со стороны сервера требуется создание учётных записей с git в качестве оболочки;
* HTTP и HTTPS (http:, https:) — использует инструмент curl (для Windows — поставляется вместе с git), и его возможности HTTP-аутентификации, как и его поддержку SSL и сертификатов.

В последнем случае требуется работа на серверной стороне веб-приложения, исполняющего роль прослойки между командами Git на сервере и HTTP-сервером (среди таковых WebGitNet, разработанный на ASP.NET MVC 4). Кроме поддержки серверной стороны команд push и pull, такие веб-приложения могут также давать доступ только на чтение к репозиторию через веб-браузер [1].

# 2. Использование GIT в написании отчета

## 2.1 Начало работы над новым проектом с помощью GIT

Нужно перейти в папку, в которой хотим работать. После убеждаемся с помощью pwd, что мы находимся в нужном месте, далее с помощью git init и указанием имени мы создаем новый проект (рисунок 2.1).

Репозиторий GIT — это папка, в которую GIT отслеживает изменения. На компьютере может быть любое количество репозиториев, каждое из которых хранится в собственной папке. Каждый репозиторий GIT в системе является независимым, поэтому изменения, сохраненные в одном репозитории GIT, не влияют на содержимое другого [2, 3].

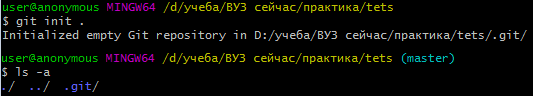


Рисунок 2.1 – Создание новой репозитории

Внутри нашей папки можем видеть с помощью ls -a, что создается скрытая подпапка .git для контроля и управления версиями.

## 2.2 Работа над существующим проектом с помощью GIT

Теперь я создал 4 файла в репозитории созданным для написания отчета (рисунок 2.2). Репозиторий GIT содержит каждую версию каждого файла, сохраненного в репозитории. Данная особенность отличает GIT от других систем управления версиями, которые хранят только различия между файлами. GIT хранит версии файлов в скрытой папке .git вместе с другими сведениями, которые необходимо использовать для управления кодом.

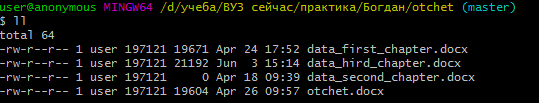


Рисунок 2.2 – Созданные файлы

У меня в отчете 3 главы, и по мере написания каждой из них я делал коммиты, для того чтобы сохранять изменения в репезатирии проекта. С помощью команды git status можно посмотреть какие файлы изменены и пока что не добавлены в индекс (рисунок 2.3).

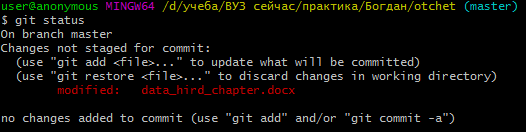


Рисунок 2.3 – Просмотр изменений

Далее идет команда git add для добавления измененных файлов в индекс (рисунок 2.4). Индекс в Git — это специальная промежуточная область, в которой хранятся изменения файлов на пути от рабочей директории до репозитория. При выполнении коммита в него попадают только те изменения, которые были добавлены в индекс [2].

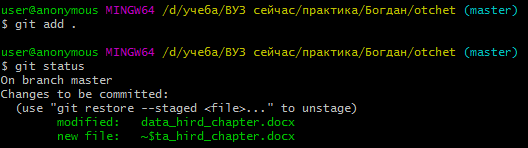


Рисунок 2.4 – Добавление изменений в индекс

И в итоге, делается коммит файлов, находящихся в индексе. А с помощью команды git log можно посмотреть историю коммитов и получить по ним краткую информацию (рисунок 2.5). Git commit - это команда для записи индексированных изменений в репозиторий GIT [2].

Прежде чем создавать очередной коммит, необходимо проиндексировать файлы в рабочей области с помощью команды git-add, это я сделал на предыдущем шаге. Новый коммит будет включать текущие состояния индексированных файлов плюс последние сохраненные состояния неиндексированных (но отслеживаемых) файлов.

Каждому коммиту соответствует код, создаваемый GIT по алгоритму Secure Hash Alrorithm 1. Он зависит от содержимого коммита, автора и времени создания. Таким образом, коммит с тем же содержимым, созданный в другое время, имеет другой sha1. GIT использует sha1 для того, чтобы различать коммиты (и другие объекты) между собой [3].

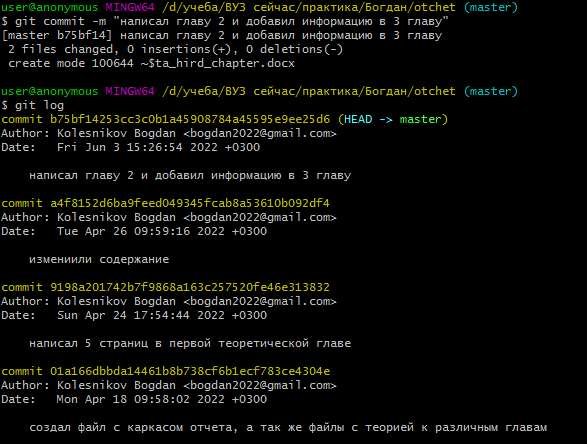


Рисунок 2.5 – Коммит и просмотр лога коммитов

## 2.3 Создание новой ветки и работа в нем

Почти каждая система контроля версий (СКВ) в какой-то форме поддерживает ветвление. Используя ветвление, Вы отклоняетесь от основной линии разработки и продолжаете работу независимо от неё, не вмешиваясь в основную линию. Во многих СКВ создание веток — это очень затратный процесс, часто требующий создания новой копии каталога с исходным кодом, что может занять много времени для большого проекта.

Некоторые люди, говоря о модели ветвления Git, называют ее «киллер-фича», что выгодно выделяет Git на фоне остальных СКВ. Что в ней такого особенного? Ветвление Git очень легковесно: операция создания ветки выполняется почти мгновенно, переключение между ветками туда-сюда, обычно, также быстро. В отличие от многих других СКВ, Git поощряет процесс работы, при котором ветвление и слияние выполняется часто, даже по несколько раз в день. Понимание и владение этой функциональностью дает вам уникальный и мощный инструмент, который может полностью изменить привычный процесс разработки.

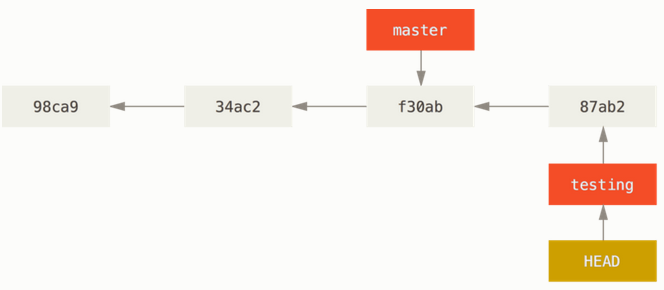


Рисунок 2.6 – Пример работы в разных ветках

Что же на самом деле происходит при создании ветки? Всего лишь создаётся новый указатель для дальнейшего перемещения. Далее я создам новую ветку для доработок моего отчета (рисунок 2.7). В результате создаётся новый указатель на текущий коммит.

Для переключения на существующую ветку выполните команду git checkout. В результате указатель HEAD переместится на ветку testing.

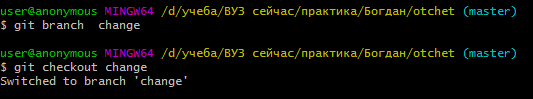


Рисунок 2.7 – Создание новой ветки

Сделаю коммит на новой ветке. Будет наблюдаться следующая ситуация: указатель на ветку change переместился вперёд, а master указывает на тот же коммит, где вы были до переключения веток командой git checkout. Давайте переключимся назад на ветку master (Рисунок 2.8). Эта команда сделала две вещи: переместила указатель HEAD назад на ветку master и вернула файлы в рабочем каталоге в то состояние, на снимок которого указывает master. Это также означает, что все вносимые с этого момента изменения будут относиться к старой версии проекта. Другими словами, вы откатили все изменения ветки change и можете продолжать в другом направлении.

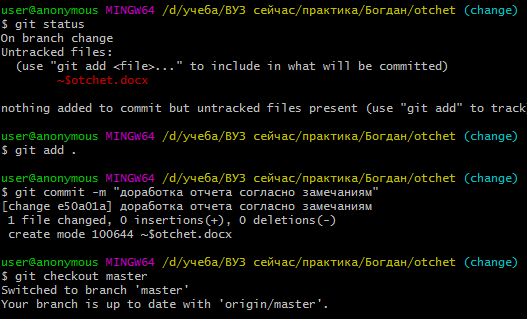


Рисунок 2.8 – Коммит и переход на основную ветку

После проверки необходимо слить наши ветки, то есть переместить указатель master на последний коммит (рисунок 2.9).

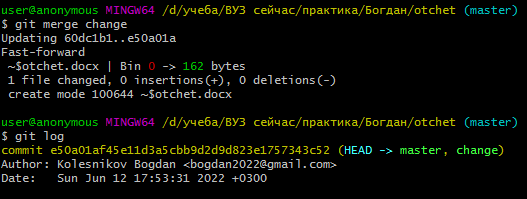


Рисунок 2.9 – Слияние веток

# 3. Уязвимости в GIT

CVE (англ. Common Vulnerabilities and Exposures) — база данных общеизвестных уязвимостей информационной безопасности. Каждой уязвимости присваивается идентификационный номер вида CVE-год-номер [1], описание и ряд общедоступных ссылок с описанием.

Уязвимость - это слабость в части компьютерного программного обеспечения, которое может быть использовано для доступа к вещам, к которым нельзя получить доступ. Например, программное обеспечение, которое обрабатывает кредитные карты, не должно позволять людям читать номера кредитных карт, которые оно обрабатывает, но хакеры могут использовать уязвимость для кражи номеров кредитных карт. Говорить об одной конкретной уязвимости сложно, потому что существует много частей программного обеспечения, иногда со многими уязвимостями. Идентификаторы CVE дают каждой уязвимости одно имя, поэтому люди могут говорить о конкретных уязвимостях, используя их имена [4, 5].

Выглядит примерно так: CVE ID, Reference и Description. ID записывается с указанием года и порядкового номера, например, "CVE-2017-5754". В поле Reference записываются ссылки на патчи, документы рекомендательного рода или комментарии разработчика. Description отвечает за описание самой уязвимости. CVE — система широкого профиля и не сосредотачивается только на клиентских уязвимостях или исключительно на WEB-протоколе. Изначально она задумывалась как единый стандарт идентификации уязвимостей, который должен охватывать несколько звеньев информационной системы: систему поиска и обнаружения брешей (например, сканер безопасности), антивирусное ПО, а также исследуемое ПО.

CVE предназначены для программного обеспечения, которое было публично выпущено; это может включать бета-версии и другие предварительные версии, если они широко используются. Коммерческое программное обеспечение включено в категорию "общедоступное", однако специально созданное программное обеспечение, которое не распространяется, как правило, не получает CVE. Кроме того, службы (например, веб-провайдер электронной почты) не назначаются CVE для уязвимостей, обнаруженных в службе (например, уязвимость XSS), если проблема не существует в базовом программном продукте, который распространяется публично [5].

## 3.1 Уязвимость CVE-2022-24765

CVE-2022-24765 - на многопользовательских системах с совместно используемыми каталогами выявлена возможность организации атаки, приводящей к запуску команд, определённых другим пользователем. Атакующий может создать каталог ".git" в местах, пересекающихся с другими пользователями (например, в совместно используемых каталогах или каталогах с временными файлами) и разместить в нём файл конфигурации ".git/config" с настройкой обработчиков, вызываемых при выполнении тех или иных команд git (например, для организации выполнения кода можно использовать параметр core.fsmonitor) [4].

Определённые в ".git/config" обработчики будут вызваны с правами другого пользователя, если этот пользователь воспользуется git в каталоге, расположенном уровнем выше, чем созданный атакующим подкаталог ".git". В том числе вызов может быть совершён косвенно, например, при использовании редакторов кода с поддержкой git, таких как VS Code и Atom, или при применении надстроек, запускающих "git status" (например, Git Bash или posh-git). В версии Git 2.35.2 уязвимость блокирована через изменения логики поиска ".git" в нижележащих каталогах (каталог ".git" теперь не учитывается, если он принадлежит другому пользователю) [4].

## 3.2 Уязвимость CVE-2022-24765

CVE-2022-24767 - специфичная для платформы Windows уязвимость, позволяющая организовать выполнение кода с привилегиями SYSTEM при запуске операции удаления (Uninstall) программы Git for Windows. Проблема вызвана тем, что программа удаления запускается во временном каталоге, доступном на запись пользователям системы. Атака осуществляется через размещение заменяющих DLL во временном каталоге, которые будут загружены при запуске uninstaller с правами SYSTEM [4, 5].

Опубликованы корректирующие выпуски распределённой системы управления исходными текстами Git 2.35.2, 2.30.3, 2.31.2, 2.32.1, 2.33.2 и 2.34.2, в которых устранены две уязвимости.

# ЗАКЛЮЧЕНИЕ

В заключение своего отчета хочу сказать, что при прохождении производственной практики особых трудностей не возникало. Было очень интересно знакомиться с работой предприятия. Данная практика поможет мне в дальнейшем обучении и работе. Во время прохождения практики я овладел навыки работы с системой контроля версий GIT, которая широко применяется в разработке программ.

Так же мною были выполнены следующие задачи:

1. Изучил и ознакомиться с различными материалами и информацией (в интернете, книгах и т.д) связанные с GIT, чтобы иметь представление, что это такое, для чего нужно, где применяется и как используется.

2. Прошел онлайн уроки по GIT, для понимания как с ним работать и разобраться с командами и как он устроен (https://learngitbranching.js.org).

3. Выполнил работу над отчетом по практике в GIT, т.е. создал локальный репозиторий и добавил в него свой отчет. При каждом изменении делал фиксации и прочее.

4. Нашел статьи о том, как важно использовать GIT, чтобы обеспечить целостность и доступность информации.

Думаю, что опыт, полученный мной на данной практике, несомненно, пригодится мне и в дальнейшем при построении своей профессиональной деятельности.

# 

# СПИСОК ЛИТЕРАТУРЫ

1. GIT [Электронный ресурс] дата обращения 29.04.2022 URL: https://ru.wikipedia.org/wiki/Git

2. Индекс [Электронный ресурс] дата обращения 29.04.2022 URL: https://ru.hexlet.io/courses/intro\_to\_git/lessons/index/theory\_unit

3. Настройка репозитория [Электронный ресурс] дата обращения 29.04.2022 URL: https://docs.microsoft.com/ru-ru/devops/develop/git/set-up-a-git-repository

4. Две уязвимости в Git [Электронный ресурс] дата обращения 29.04.2022 URL: https://www.opennet.ru/opennews/art.shtml?num=57013

5. Чакон С., Штрауб Б. Git для профессионального программиста. — Питер, 2017. — 496 с. — ISBN 978-5-496-01763-3.