СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Е. Л. Турнецкая

Репозиторий проекта

Конспект лекций

СПбГЭТУ «ЛЭТИ», 2024 г.

2.1. РЕПОЗИТОРИЙ ПРОЕКТА

При работе над программным проектом каждый из команды разработчиков должен иметь доступ ко всем файлам. Поэтому возникает необходимость организации виртуального хранилища программных компонент в системе контроля версий.

Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение.

Подходы к сохранению изменений в файлах проекта

При сохранении изменений файлов проекта существует два подхода.

Первый подход – сделать *снимок или слепок файла* (snapshot). После модификации файла сделать новый слепок и сохранить оба снимка. При необходимости можно вернуться к ранее сохраненным снимкам. При таком подходе требуется много места для одновременного хранения нескольких версий файлов

Второй подход предусматривает *сохранения разницы* – дельты изменений между новой версией и старой версией файла и сохраняет только эту разницу.

На основе этих двух подходов разработано большое количество систем управления версиями. Работы с системами контроля версий рассмотрим на примере распределенной системы управления версиями Git.

Система управления версиями Git

Git – специальная программа, позволяющая контролировать изменения файлов проекта как для одиночной, так и совместной разработки кода.

Систему Git создал в 2005 году Линус Торвальдс, когда вместе с командой разрабатывал ядро операционной системы Linux.

К ее задачам относят:

1. Возврат к любой версии кода
2. Просмотр истории изменений и восстановление любых данных
3. Совместная работа разработчиков

Программные средства данной системы позволяют сохранить изменения в файл или набор файлов в процессе их модификации и при необходимости вернуться к конкретной версии файла. Если над проектом работают несколько человек, то каждому из них обеспечивают доступ для совместной работы над файлом. Каждое внесенное изменение фиксируют, поэтому возникает многоверсионность разрабатываемого программного продукта.

Базовые определения

Познакомимся с базовыми определениями:

• repository – некоторое хранилище файлов, ссылок на изменения в файлах

• commit – отслеживание изменений, сохраняет разницу в изменениях

• working directory – рабочий каталог на вашем компьютере

• staging area, index – область подготовленных файлов или рабочая область

• branch – ветка, состоит из набора коммитов, обычно ссылается на последний коммит

• merge – слияние, слияние веток в одну

• pull – взять проект с сервера, получить изменения из удаленного репозитория

• push – отправить изменения на сервер

Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз при фиксации текущей версии проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент (рис.1). Если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл.

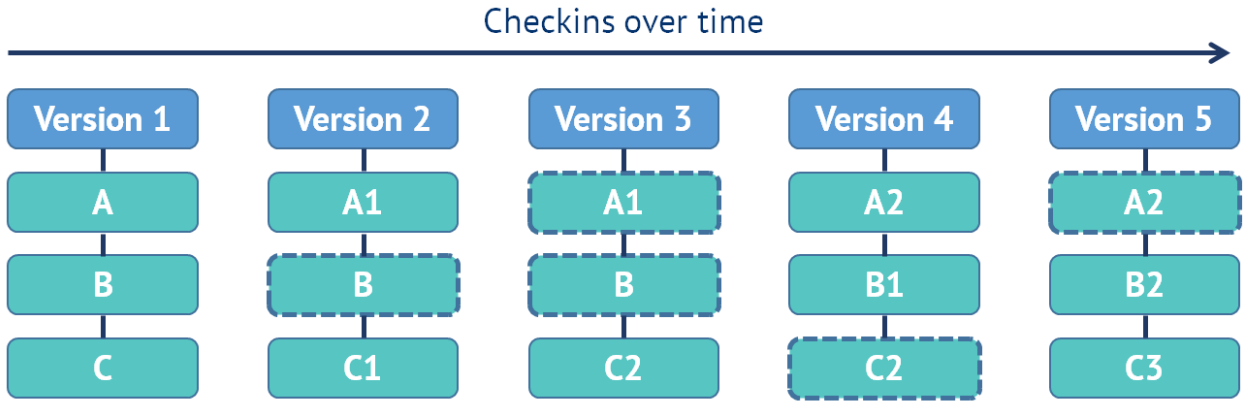

Рисунок 1 – Подход к хранению файлов и изменения в Git

В репозитории хранят все файлы проекта. Для того, чтобы начать с ними работать, их необходимо скопировать в рабочий каталог. С точки зрения Git каждый файл может находится только в одном из двух состояний: отслеживаемый и неотслеживаемый (рис.2).

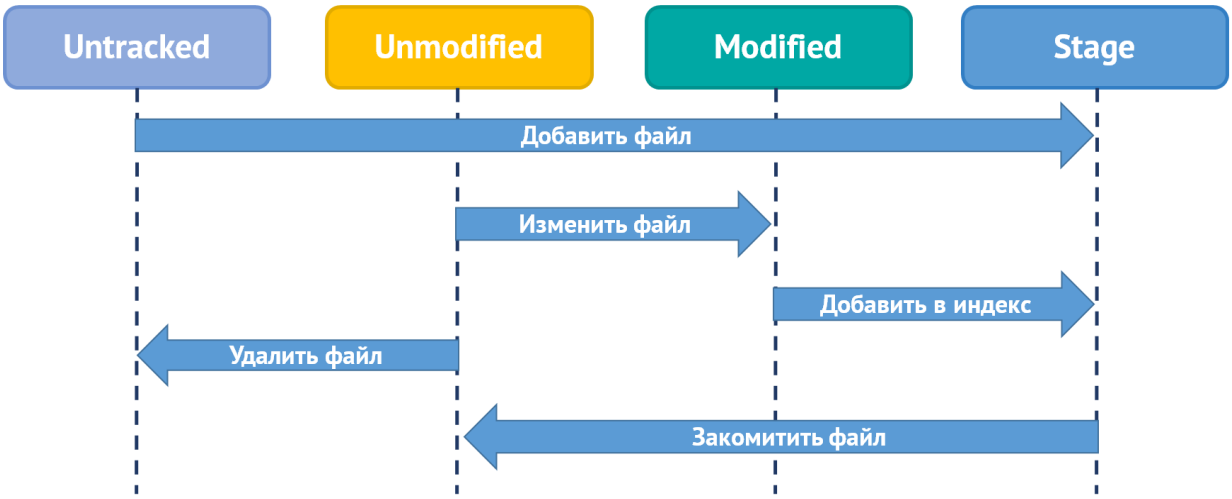

Рисунок 2 – Состояния файлов Git

1. Отслеживаемый. Об этих файлах Git знает и отслеживает изменения в них. Отслеживаемые файлы в свою очередь могут находится в следующих состояниях:

– Неизмененный (unmodified). То есть с момента последнего коммита в файле не было никаких изменений.

– Измененный (modyfield). То есть с последнего коммита в файле были произведены какие-то изменения.

– Подготовленный к коммиту (stage). Это значит, что вы внесли изменения в этот файл и затем проиндексировали их, и эти изменения будут добавлены в следующий коммит.

2. Неотслеживаемый (untracked). О неотслеживаемых файлах Git не знает, поэтому изменения в них не будут добавлены в коммит. Это любые файлы в вашем рабочем каталоге, которые не входили в последний коммит и не подготовлены к текущему коммиту.

В репозитории хранят все файлы проекта. Для того, чтобы начать сними работать их необходимо скопировать в рабочий каталог. После внесенных изменений файлы перемещают в область подготовленных файлов. И только затем перемещают в репозиторий Git, делают коммит.

Репозиторий Git условно можно разделить на три составляющие (рис.3)

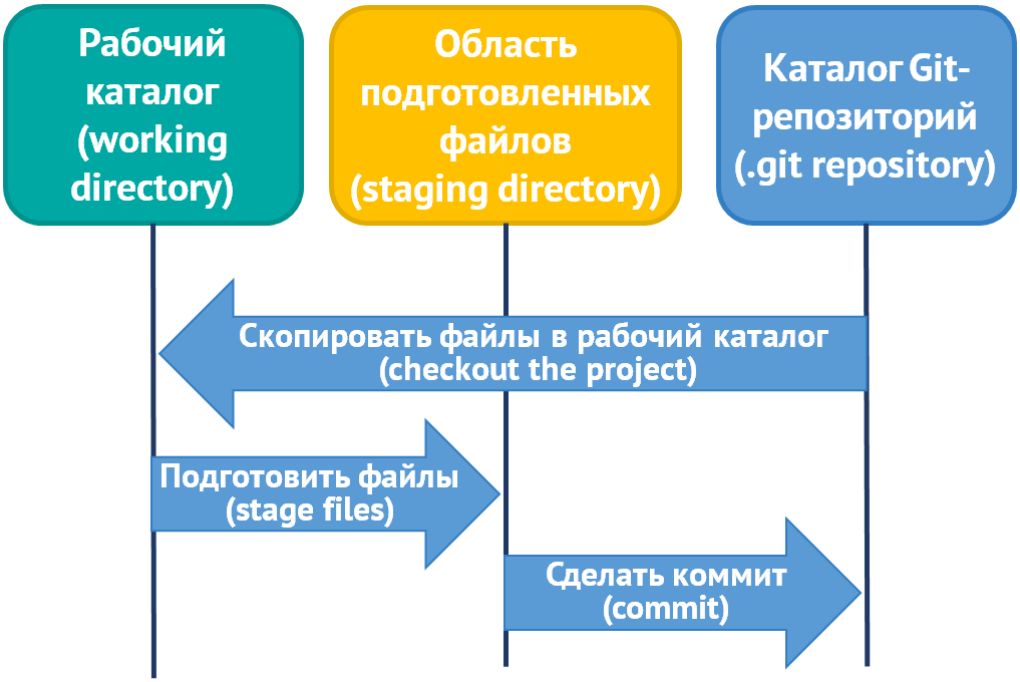

Рисунок 3 – Перемещение файлов проекта

Рабочая директория – Working directory. Это файловая структура, с которой непосредственно работает пользователь в конкретный момент времени. Технически же – это копия определенной версии вашего проекта, которую вы извлекли из базы Git и в которую пытаетесь внести свои изменения.

Индекс или Область подготовленных файлов – Index / Staging area. Это область, где хранятся имена файлов и изменения в них, которые должны войти в следующий коммит. Технически индекс – это просто файл.

Директория Git – Git Directory. Папка, в которой Git хранит все версии вашего проекта и также свои служебные файлы. Данная папка носит название .git и располагается в корневой директории вашего проекта.

При работе над проектом создают ветви (рис.4). Главная ветвь проекта называют main или master. Другие ветви проекта могут называть deveop, Future или придумывают любые псевдонимы.

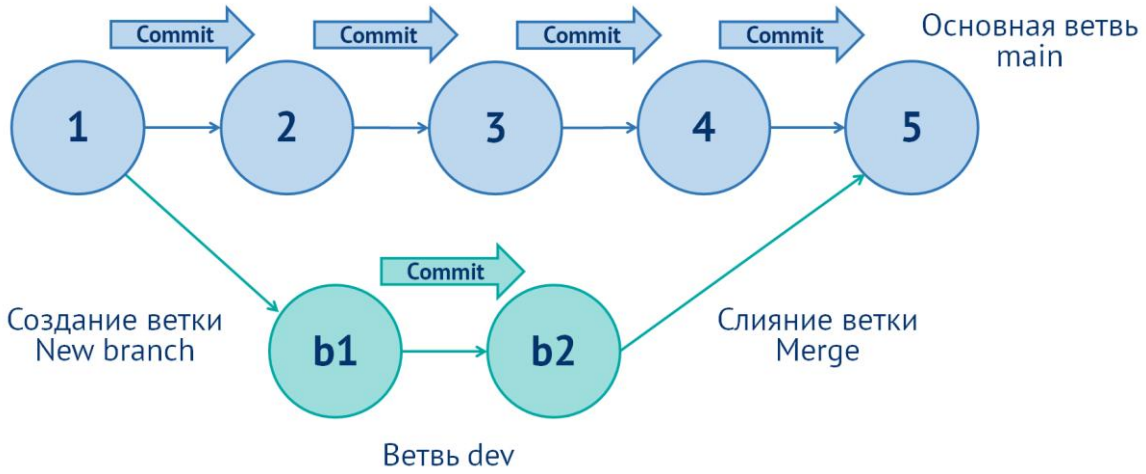

Рисунок 4 – Ветви проекта

При фиксации изменений внутри одной ветви проводят коммит, выполнив команду git commit. Коммитить можно один файл или сразу несколько. Система сама найдёт, что изменилось в каждом файле, и добавит эти изменения в проект. Но все эти правки внесутся в репозиторий за один раз, потому что при коммите обрабатываются сразу все добавленные в список файлы.

Единственное требование к коммитам – указывать, что именно вы поменяли в проекте, человеческим языком. Хорошим тоном и правильным подходом считается писать, что именно вы изменили: «Добавил цвет и стили основной кнопки», «Убрали метод вызова старого API». Это описание будут читать другие разработчики. Есть правило о коммитах (https://www.conventionalcommits.org/ru/v1.0.0-beta.2/).

Семь правил хорошего описания коммита:

Правило 1: оставляйте пустую строку между заголовком и описанием.

Правило 2: ограничивайте длину заголовка 50 символами.

Правило 3: пишите заголовок с прописной (заглавной) буквы.

Правило 4: не ставьте точку в конце заголовка описания.

Правило 5: используйте повелительное наклонение в заголовке.

Правило 6: ограничивайте длину строки в теле описания 72 символами.

Правило 7: в теле описания отвечайте на вопросы «что?» и «почему?», а не «как?».

Новые версии могут разрабатывать в разных ветвях. Для их создания выполняют команду git branch. После выполнения своей части работ разработчику необходимо объединить свою ветвь с главной. Для добавления нового кода, новых файлов проекта в репозиторий используют команду git merge.

При работе с системой контроля версий следует помнить, что все изменения происходят в рабочей зоне на компьютере у пользователя. Потом сделанные изменения необходимо зафиксировать.

Разработчик сначала забирает файлы проекта к себе командой push request, а затем передает их обратно на сервер командой pull request. Т.е. делает запрос на слияние. Если команда его одобряет, то выполняется слияние ветвей.

Рассмотрим некоторые команды без атрибутов и ключей:

• git init – создает новый репозиторий;
• git status – отображает список измененных, добавленных и удаленных файлов;
• git add – добавляет указанные файлы в индекс;
• git commit – фиксирует добавленные в индекс изменения;
• git branch – создать ветку с помощью команды;
• git checkout – создать ветки и сразу переключиться на них;
• git branch – просмотр всех веток.

Подробно с командами гит можно ознакомиться в большой книге ProGit (https://git-scm.com/book/ru).

Например, в ней будет такое описание команды git add (добавить):

Команда git add добавляет содержимое рабочего каталога в индекс (staging area) для последующего коммита. По умолчанию git commit использует лишь этот индекс, так что вы можете использовать git add для сборки слепка вашего следующего коммита.

Все команды выполняются в консоли. Git – это инструмент для командной строки, у него нет графического интерфейса. Но у него довольно много приложений (рис.5) с графическим интерфейсом, например GitHub, GitLab (https://git-scm.com/downloads/guis/).

Рисунок 5 – Информация о графических клиентах Git

Онлайн-сервис GitHub

GitHub – это онлайн-сервис, который основан на технологии Git.

Назначение GitHub:

– сохранять репозитории в интернете,

– автоматически синхронизировать их с репозиториями у разработчиков,

– следить за обновлениями кода,

– редактировать код прямо в репозитории,

– копировать себе чужие репозитории.

В нем встроены трекер задач, система уведомлений, форум, переписка между пользователями и комментарии.

GitHub обладает интуитивно понятным, дружелюбным интерфейсом (рис.6).

Начало работы с GitHub:

1. Зарегистрироваться на GitHub;
2. Создать свой репозиторий;
3. Загрузить код на GitHub;
4. Добавить описание репозитория и файл README.md (можно позже).

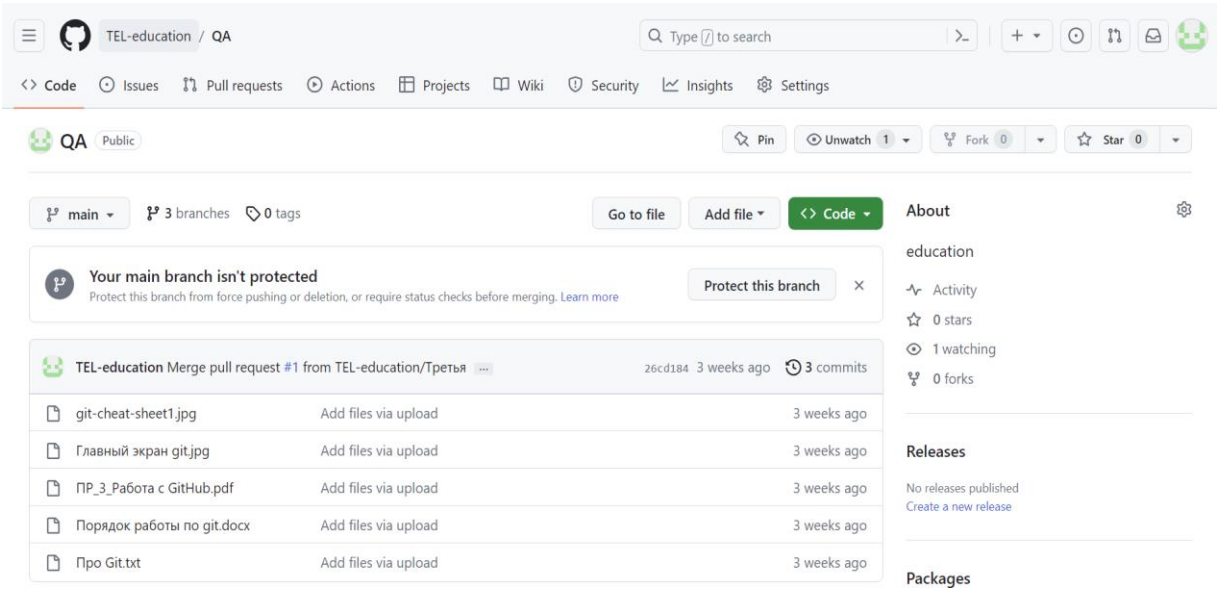

Рисунок 6 – Интерфейс репозитория на GitHub

Пользователь определяет порядок действий, а GitHub генерирует программный код команд и выполняет их (рис.7).

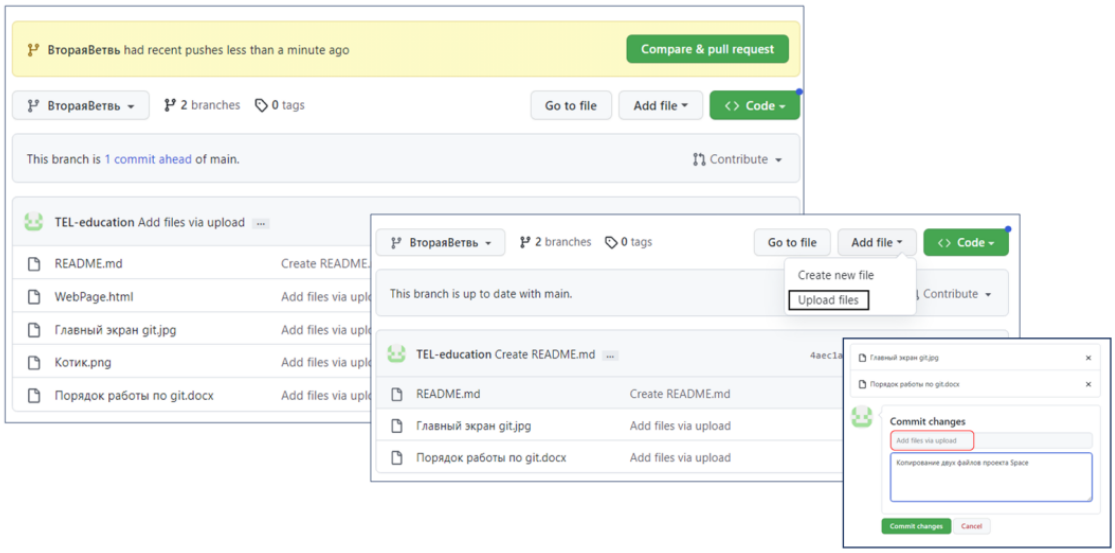

Рисунок 7 – Автогенерация кода команд

Например, так будут выглядеть изменения, внесенные в программный код после слияния (рис.8).

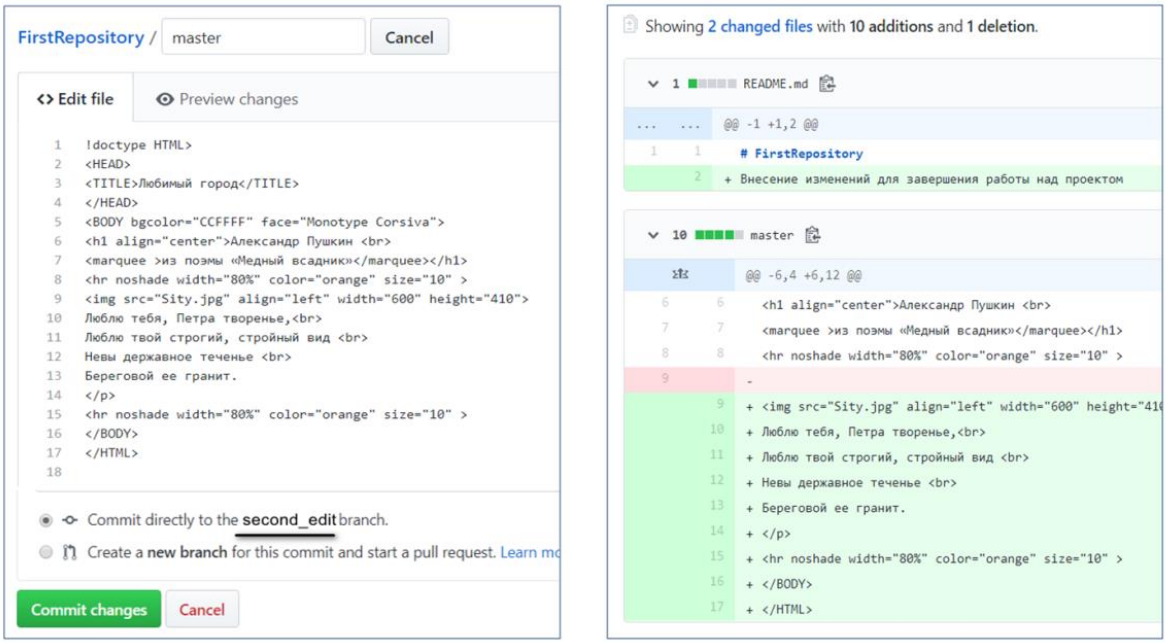

Рисунок 8 – Визуализация изменения в коде

Работа в системе контроля версий дает возможность сохранять промежуточные варианты проекта, авторов и время изменения. В случае неудачного варианта изменений всегда можно откатиться назад и остановиться на стабильном коммите.

В профиле на GitHub можно разместить контакты, список навыков, код проектов и ссылки на важные ресурсы. Если нет своего сайта, страница сможет его заменить.

Список источников

1. Орлов С.А. Программная инженерия. Технологии разработки программного обеспечения: учебник. — СПб: Питер, 2020. С.640.
2. Руководство по основам Git URL: https://guides.hexlet.io/ru/git-guide/
3. Методологии и технологии проектирования информационных систем: учебно-методическое пособие / А. В. Аграновский, В. С. Павлов, Е. Л. Турнецкая; С.-Петерб. гос. ун-т аэрокосм. приборостроения. - Санкт-Петербург: Изд-во ГУАП, 2021. -111 с.
4. Турнецкая, Е. Л. Программная инженерия. Интеграционный подход к разработке / Е. Л. Турнецкая, А. В. Аграновский. — Санкт-Петербург : Лань, 2023. — 216 с. — ISBN 978-5-507-46898-0. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/352307 (дата обращения: 09.09.2023). — Режим доступа: для авториз. пользователей.
5. Платформа IT-технологий URL: https://smartiqa.ru/