Московский государственный технический университет
имени Н. Э. Баумана

В. И. Кузнецов

Система контроля версий GIT

Методические указания к выполнению
лабораторной работы №6 по дисциплине
«Языки интернет программирования»

Редакция от 1.09.2025

Факультет «Информатика и системы управления»
Кафедра «Компьютерные системы и сети»

Кузнецов В.И.

Система контроля версий GIT: методические указания к выполнению практикума № 6 и лабораторной работы № 6 по дисциплине «Языки интернет- программирования» / В. И. Кузнецов.

В методическом пособии приведены основы использования системы контроля версия GIT. Так же приведены основные методики построения ведения проектов в системах контроля версий

Для студентов МГТУ им. Н.Э. Баумана, обучающихся по направлению «Информатика и вычислительная техника».

Теоретическая часть

Git — это распределённая система контроля версий, предоставляющая набор консольных инструментов для отслеживания и фиксации изменений в файлах (чаще всего — в исходном коде, однако её можно применять к любым данным). Создана Линусом Торвальдсом в ходе разработки ядра Linux и впоследствии получила широкое распространение. С помощью Git можно сравнивать версии, анализировать различия, объединять изменения и при необходимости возвращаться к любому ранее зафиксированному состоянию.

Основное назначение Git — обеспечить управляемую эволюцию проекта во времени. Система позволяет видеть, какие правки вносились на всех этапах разработки, и оперативно откатываться к стабильной версии, если недавние изменения привели к ошибкам, без трудоёмкого ручного поиска прежнего состояния.

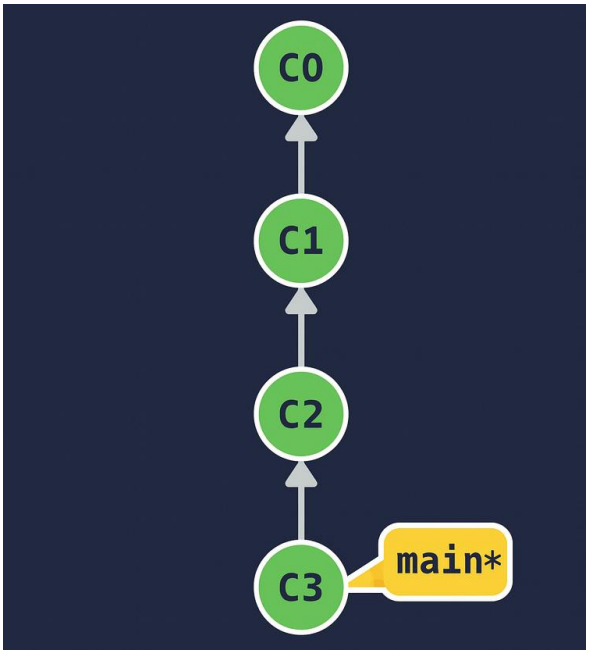
Git особо эффективен при совместной работе нескольких специалистов над одним проектом: он поддерживает параллельную разработку, упрощает согласование правок и снижает риск конфликтов, неизбежных при ручной замене файлов.

Архитектурно Git распределён: он не зависит от единого центрального сервера, поскольку каждая локальная копия содержит полный репозиторий. При этом репозиторий может быть размещён на удалённом сервере, что облегчает командную работу и синхронизацию. Удаленным сервером может служить общедоступные открытые средства: GitLab, GitHub. Могут быть закрытые: BitBucket. Так же можно развернуть свой сервер контроля версий.

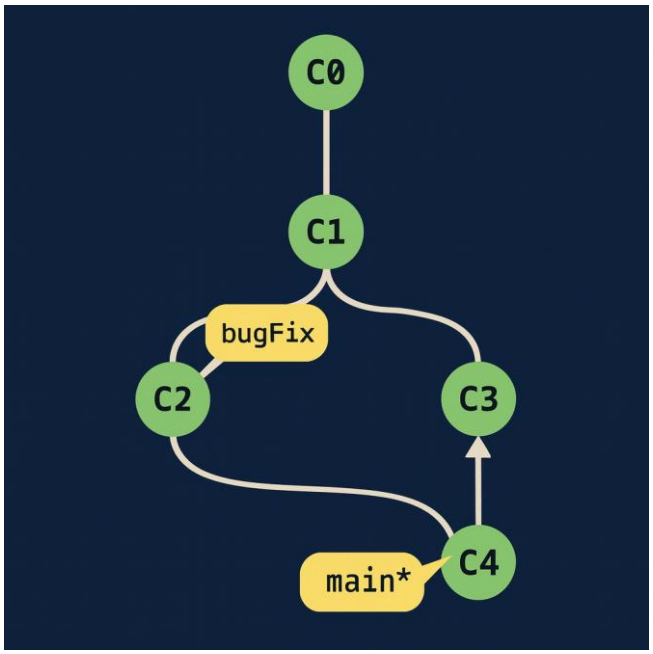
Каждая зафиксированная версия проекта называется коммитом (commit). Коммит представляет собой снимок состояния файлов на определённый момент времени и содержит служебные данные: автора и время фиксации, ссылки на «родительские» коммиты, а также сообщение коммита (краткий заголовок и, при необходимости, развёрнутое описание сделанных изменений).

Каждому коммиту присваивается криптографический хеш — уникальный идентификатор. Этот хеш вычисляется на основе содержимого и метаданных коммита, благодаря чему изменение уже сохранённой версии невозможно незаметно: при любых правках идентификатор изменится. В повседневной работе часто используют сокращённую форму хеша (несколько первых символов), если она остаётся уникальной в пределах репозитория.

Из последовательности связанных между собой коммитов формируется ветка. Технически ветка — это именованный указатель на конкретный коммит; «история ветки» — это цепочка коммитов, достижимая по ссылкам на родителей от её текущего состояния назад во времени. У каждой ветки есть собственное имя (например, main, develop, feature/login).



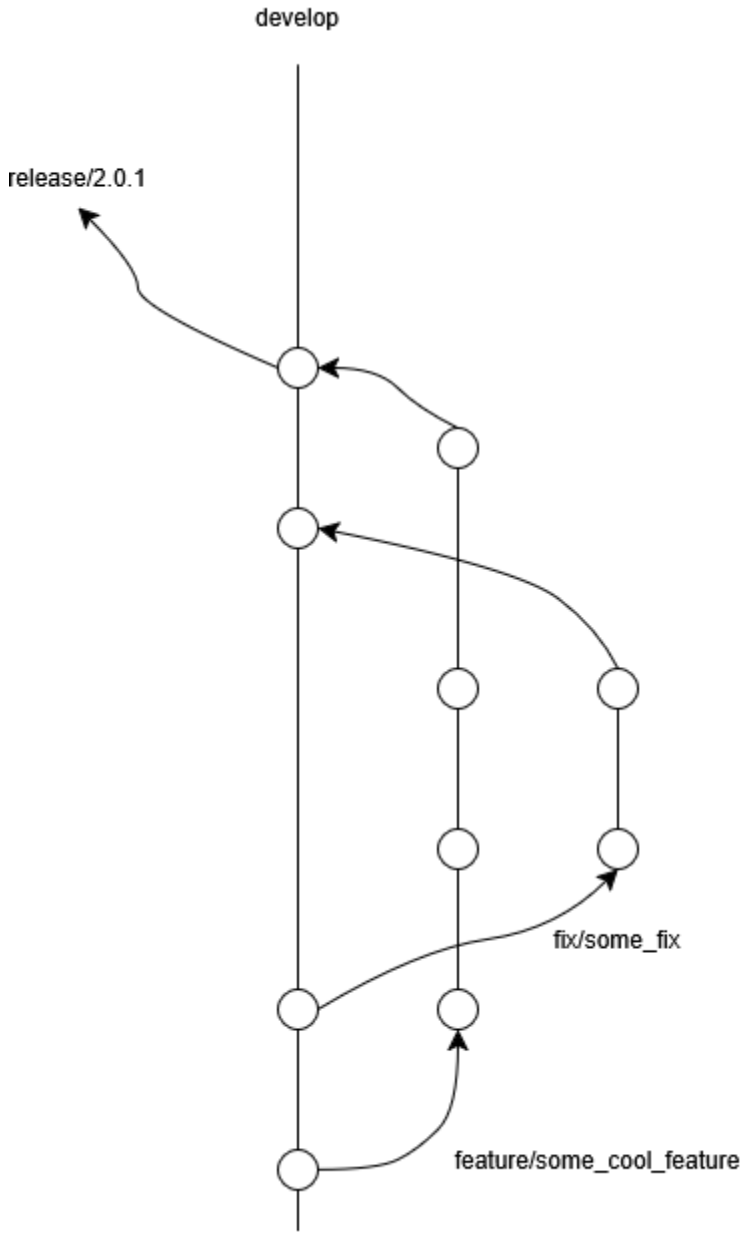
Один репозиторий может содержать множество веток. Их создают от любой существующей точки истории для разработки новой функциональности, исправления ошибок или подготовки релиза. По завершении работ изменения ветки объединяют с другой веткой — как правило, с основной — с помощью слияния (merge) или переноса коммитов (rebase), после чего вспомогательную ветку можно удалить, не теряя истории.



Подход ведения GIT

Подход ведения зависит от подхода к ведению проектов в компании или соглашения команды. Основных подхода к ведению GIT 2.

Общий ветка develop

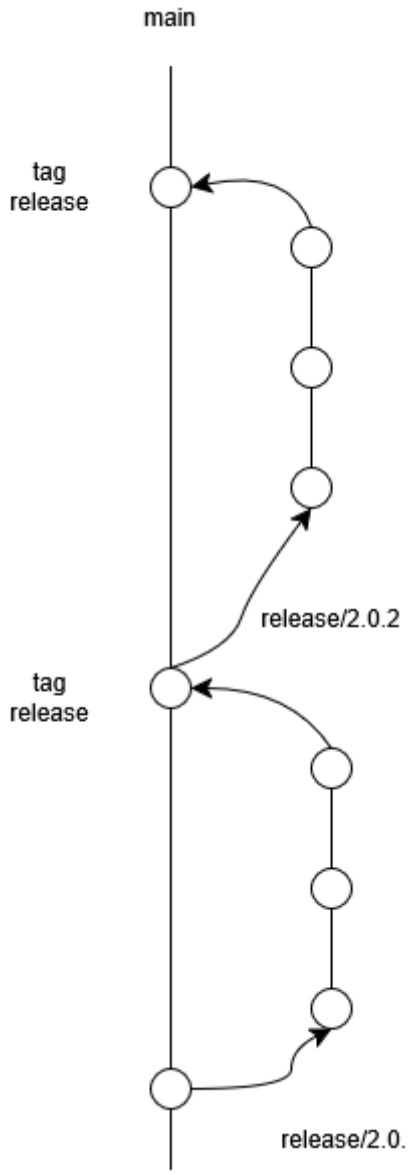


Стратегия ведения через develop заключается в ответвлении от главной ветки develop (реже main) веток задачи и все изменения ведутся именно в них. После завершения цикла разработки по задаче (разботка+тестирования) происходит влитие ветки в develop через pull request. Он позволяет предложить свою ветку кода для проверки (код-ревью) другими разработчиками, обсудить их, и после одобрения — слить (merge) эти изменения в основную ветку проекта.

При таком ведении версий кода возможны конфликты слияния (merge conflict). Они происходят при работе в разных ветках над одним и тем же кодом разными разработчиками и решается своевременным обновлением своей ветки новыми коммитами из главной ветки.

В целом данная методология часто используется при отсутствии четких дат релиза или частом изменении состава спринта. Данная методология в целом быстра по доставке релиза и тестировании и позволяет работать большому количеству разработчиков. Проблематика заключается в подвисающих ветках при отсутствии необходимости включать включения в релиз или обратная ситуация с попаданием не желательного кода в релиз

Ведение через релизные ветки



Основа принципа заключается в отделении отдельной релизной ветки и добавление в неё всех коммитов по задачам. Все разработчики сливают свои

изменения в одну ветку, а конец спринта ознаменуется влитием релизной ветки обратно в мастер и проставлением тега с номером релиза.

К данного метода присутствует проблема с тем, что релиз может быть задержан из-за отсутствия каких-либо доработок одного из разработчиков, а также большей сложности разрешения мердж конфликтов.

Иногда используется гибрид данных методов для исключения их проблематики, но это сильно замедляет процесс разработки и требует соблюдения доктрины всеми разработчиками команды.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Шпаргалка по основным командам

`git add`
Добавляет изменения из рабочей директории в индекс (staging area) для последующего включения в коммит. Поскольку `git commit` формирует снимок именно из индекса, `git add` используется для подготовки содержимого будущего коммита.

`git status`
Отображает состояние рабочей директории и индекса: какие файлы изменены и не проиндексированы, какие подготовлены к коммиту. Дополнительно выводит подсказки по изменению этих состояний.

`git diff`
Вычисляет различия между двумя деревьями Git. Примеры: рабочая директория ↔ индекс (`git diff`), индекс ↔ последний коммит (`git diff --staged`), произвольные коммиты или ветки (`git diff master branchB`).

`git difftool`
Запускает внешнюю программу сравнения для просмотра различий между деревьями, если предпочтителен инструмент, отличный от встроенного просмотровщика `git diff`.

`git commit`
Сохраняет в репозиторий снимок состояния, сформированный из индекса (`git add`), и переводит указатель текущей ветки на созданный коммит.

`git reset`
Используется преимущественно для отмены изменений. Перемещает указатель `HEAD` и при необходимости изменяет состояние индекса. С параметром `--hard` может перезаписать файлы в рабочей директории, что чревато потерей данных; применять с осторожностью.

`git rm`
Удаляет файлы из индекса и рабочей дирекции. Логически противоположна `git add`: подготавливает удаление к включению в следующий коммит.

`git mv`
Удобная обёртка для переименования/перемещения файла: эквивалентна последовательности `mv`, затем `git add` для нового пути и `git rm` для старого.

`git clean`
Удаляет «мусор» из рабочей директории: артефакты сборки, временные файлы, оставшиеся после разрешения конфликтов, и т. п.

Шпаргалка по ветвлению и слиянию

`git branch`
Управляет ветками: перечисляет существующие, создаёт новые, удаляет и переименовывает.

`git checkout`
Переключает текущую ветку и выгружает её содержимое в рабочую директорию.

`git merge`
Объединяет указанную(ые) ветку(и) с текущей и переводит указатель текущей ветки на результирующий коммит.

`git mergetool`
Запускает внешнюю утилиту для разрешения конфликтов слияния.

`git log`
Выводит историю коммитов от более новых к более старым. По умолчанию показывает историю текущей ветки; при необходимости можно просматривать историю других (в том числе нескольких) веток и сравнивать их на уровне коммитов.

`git stash`
Временно сохраняет незакоммиченные изменения, очищая рабочую директорию без фиксации незавершённой работы в отдельной ветке.

`git tag`
Создаёт постоянную метку на определённой точке истории проекта, как правило, для обозначения релизов.

Совместная работа и обновление репозитория

Большинство команд Git работают локально; сетевое подключение требуется лишь для взаимодействия с удалёнными репозиториями.

`git fetch`
Подключается к удалённому репозиторию, забирает отсутствующие локально изменения и сохраняет их в удалённые ветки без изменения текущей рабочей ветки.

`git pull`
Комбинирует `git fetch` и последующее объединение с текущей веткой (обычно `merge`): получает изменения с сервера и пытается слить их с активной веткой.

`git push`
Отправляет локальные коммиты, отсутствующие в удалённом репозитории, на сервер. Требует прав на запись и аутентификацию.

`git remote`
Управляет списком удалённых репозиториев: добавляет, изменяет и удаляет записи. Позволяет сопоставлять длинные URL кратким именам (например, `origin`) для удобства последующих операций.

ЗАДАНИЕ

Создать аккаунт в системе контроля версий в GITHUB и выполнить следующие задания.

1. Создать пустой репозиторий в системе контроля версий.
2. Добавить коммит с любым содержимым в ветку main(master). Содержимым может служить код к одной из лабораторных работ.
3. Выделить ветку и добавить в неё 2 коммита с изменением изначального содержимого и добавления нового.
4. Через вкладку Settings раздел Collaborators пользователя Emetless.
5. Создать пул реквест с названием вашей группы и фио для вашей ветки с новыми коммитами в основную ветку и добавить в ревьюверы пользователя Emetless.

Каждый шаг должен быть зафиксирован с помощью скриншотов и описан. Для защиты необходим открытый пулл-реквест с указанным ревьювером.

Контрольные вопросы

- Что такое Git-репозиторий и чем локальный репозиторий отличается от удалённого?
- Объясните разницу между рабочей директорией, индексом (staging area) и коммитом. Каков типичный цикл команд для фиксации изменений?
- Какие команды используют, чтобы: a) увидеть текущие изменения, b) посмотреть историю коммитов? В чём разница между git status и git log?
- Как создать новую ветку и сразу переключиться на неё? Напишите одну или две возможные команды.
- Чем отличаются git fetch, git pull и git push? В каких ситуациях применяется каждая команда?