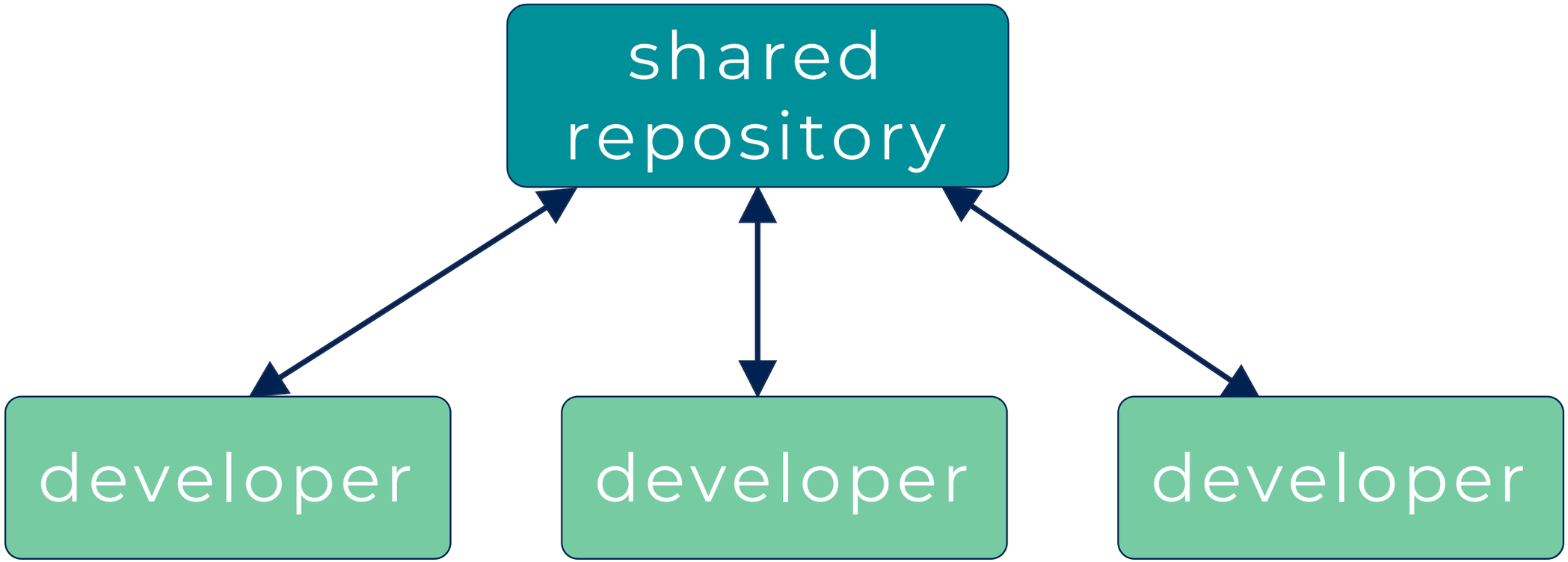
Презентацию подготовили:
студентки группы 504102/50201
Буталова Юлия
Худина Анастасия

ВНУТРЕННЕЕ УСТРОЙСТВО GIT И GITHUB

Архитектура и принципы работы системы контроля версий



СТРУКТУРА ПРЕЗЕНТАЦИИ

01

Введение в системы контроля версий

02

Архитектура и принципы Git

03

Модель хранения данных

04

Ветвление и слияние

05

Работа с удаленными репозиториями

06

Платформа GitHub и collaborative development

07

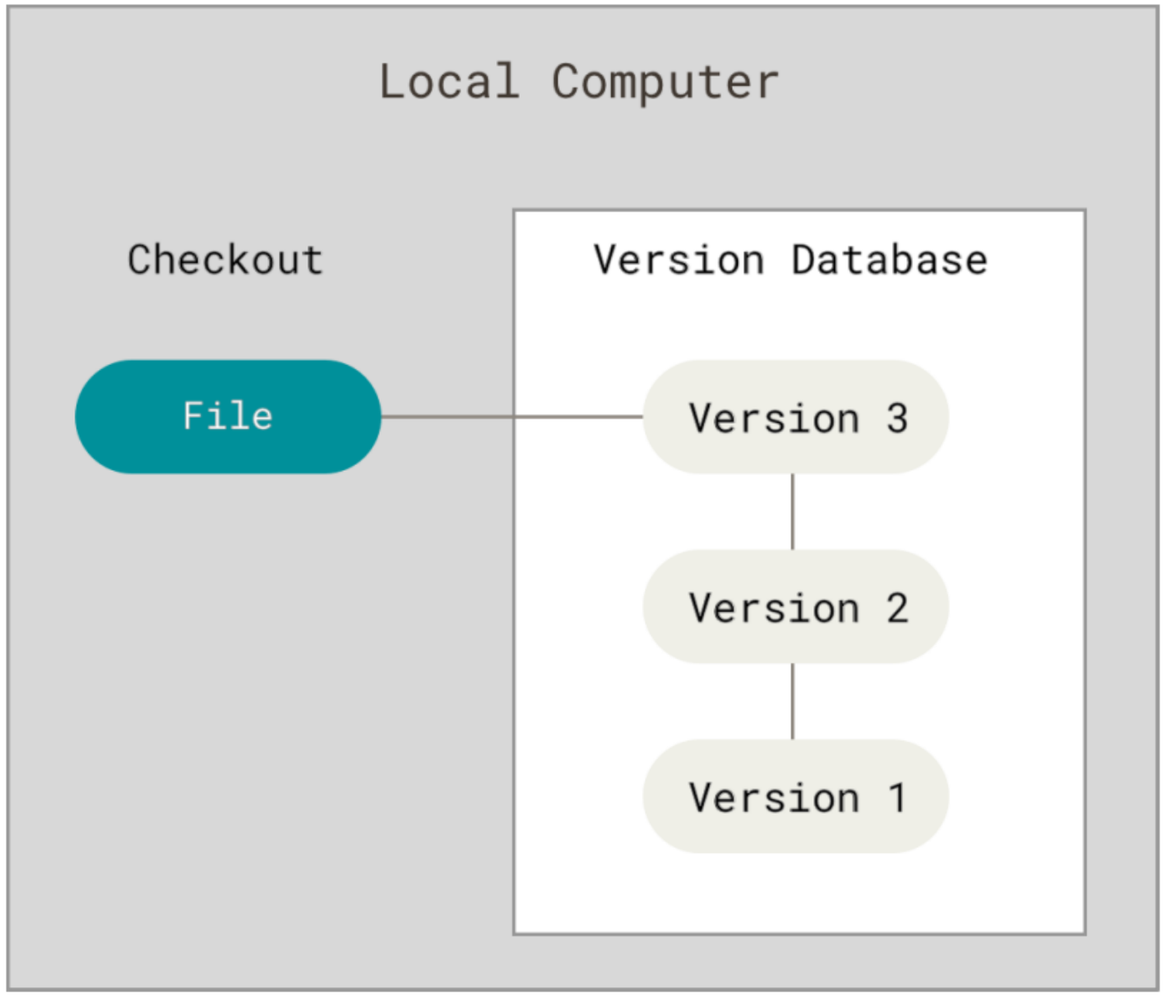
Дополнительные инструменты и практики

3

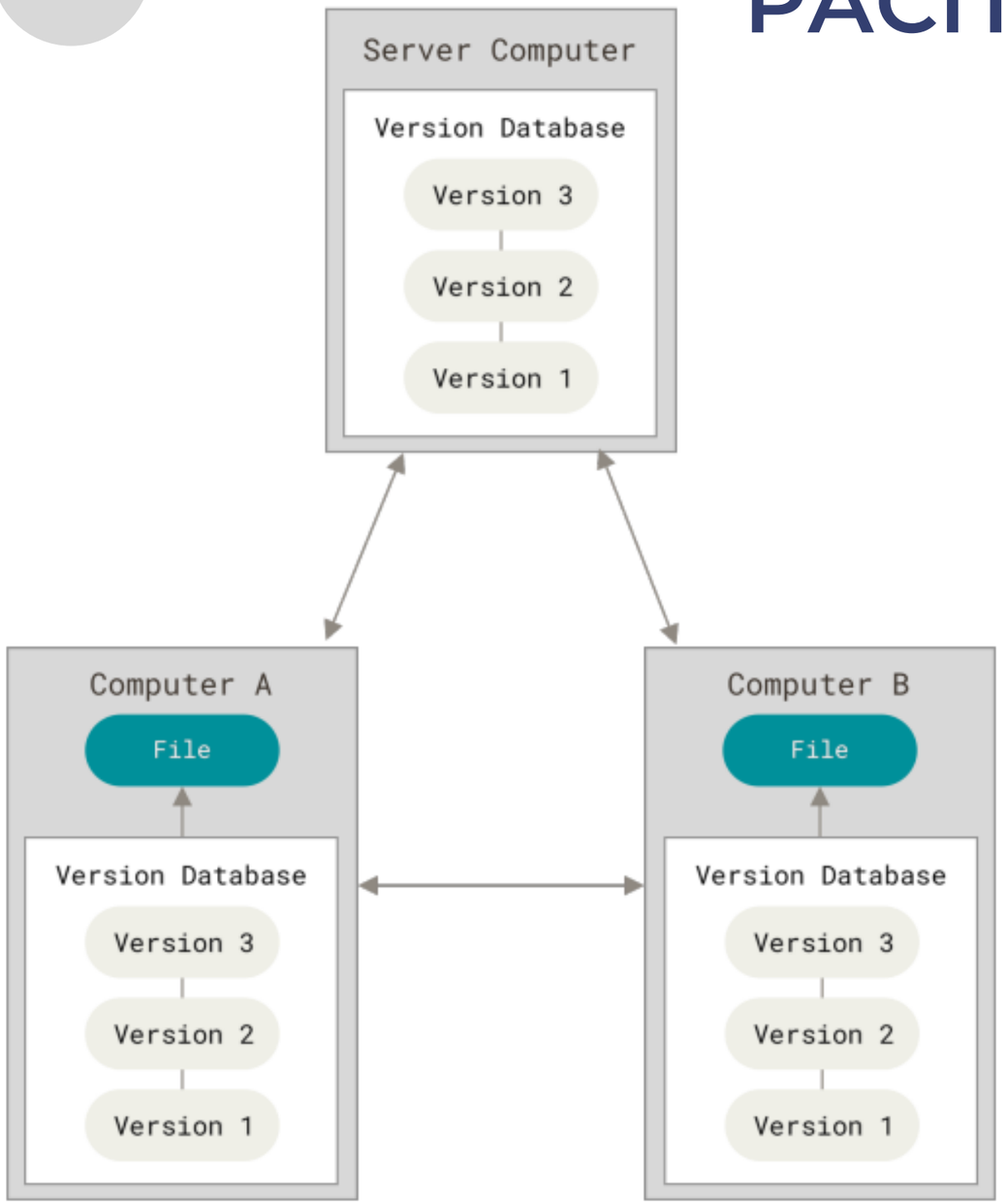
ЭВОЛЮЦИЯ ПОДХОДОВ К КОНТРОЛЮ ВЕРСИЙ

Исторически первой проблемой, решаемой системами контроля версий, была организация хранения истории изменений на локальной машине.

- Локальные системы контроля версий
- Проблема единой точки отказа
- Необходимость координации работы команды



РАСПРЕДЕЛЕННАЯ АРХИТЕКТУРА



Распределенные системы контроля версий, к которым относится Git, принципиально меняют архитектурный подход.

- Полные копии репозитория у каждого разработчика
- Отсутствие единой точки отказа
- Возможность различных workflow

ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ GIT

Скорость операций

Быстрое выполнение всех операций.

Простота архитектуры

Легкая для понимания и использования.

Нелинейная разработка

Эффективная поддержка тысяч параллельных веток.

Полная децентрализация

Каждый репозиторий является полной копией.

5

Скорость операций

Быстрое выполнение всех операций.

Простота архитектуры

Легкая для понимания и использования.

Нелинейная разработка

Эффективная поддержка тысяч параллельных веток.

Полная децентрализация

Каждый репозиторий является полной копией.

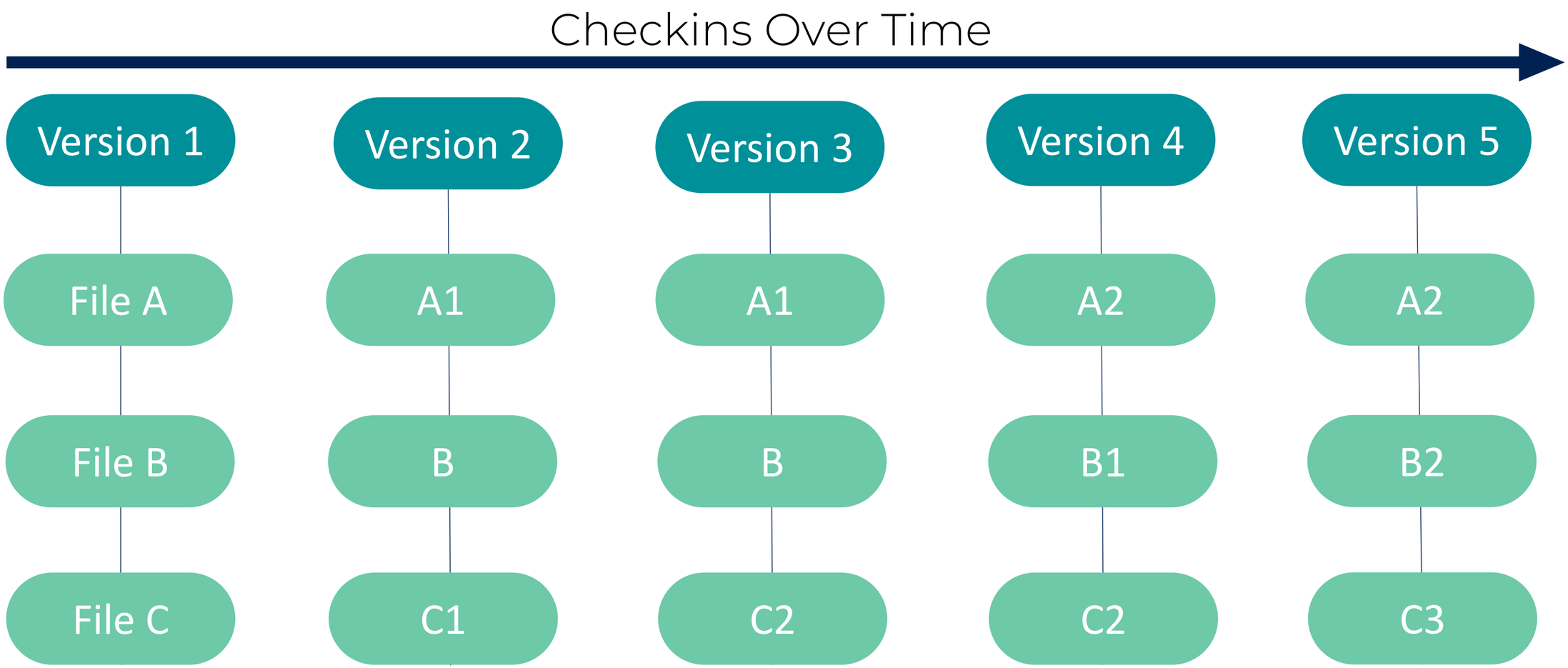
Эффективное управление большими проектами

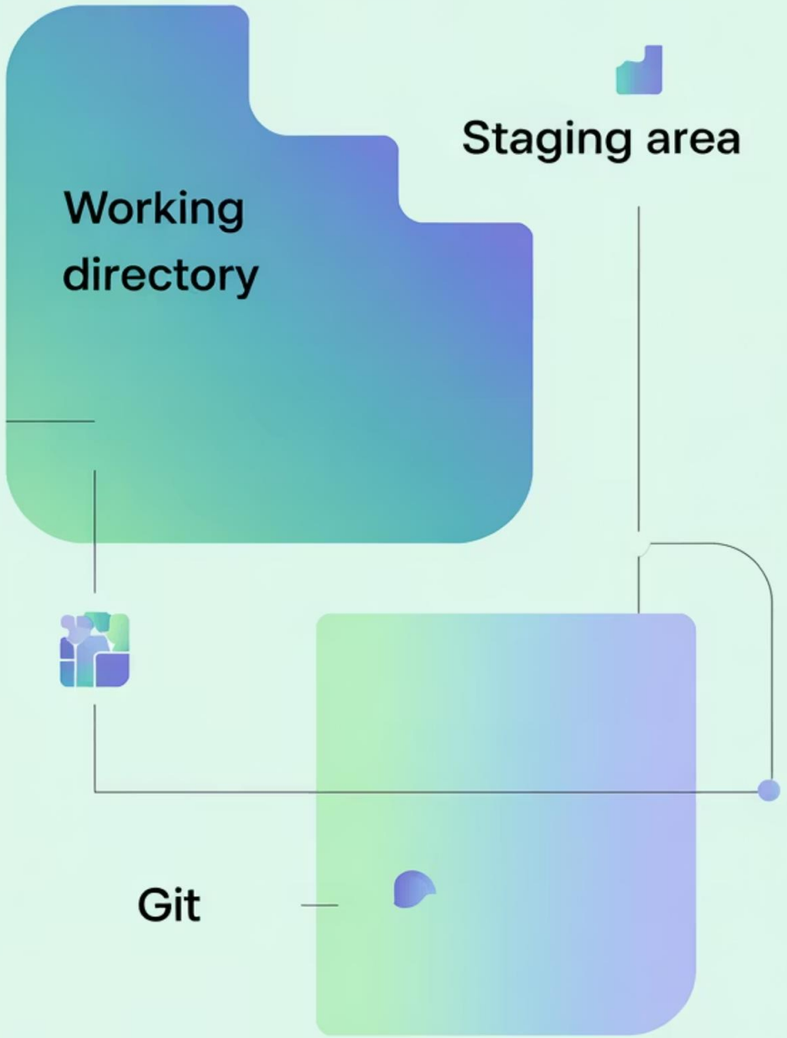
ПОДХОД К ХРАНЕНИЮ ДАННЫХ

Фундаментальным отличием Git является подход к хранению данных.

- Снимки состояния вместо различий
- Поток снимков проекта во времени
- Эффективное использование дискового пространства

ПОДХОД К ХРАНЕНИЮ ДАННЫХ





ЖИЗНЕННЫЙ ЦИКЛ ИЗМЕНЕНИЙ

Working Directory

Рабочая копия файлов.

Staging Area

Область индексирования для следующего коммита.

Git Directory

Репозиторий с метаданными и объектами.

ЖИЗНЕННЫЙ ЦИКЛ ИЗМЕНЕНИЙ

Working Directory

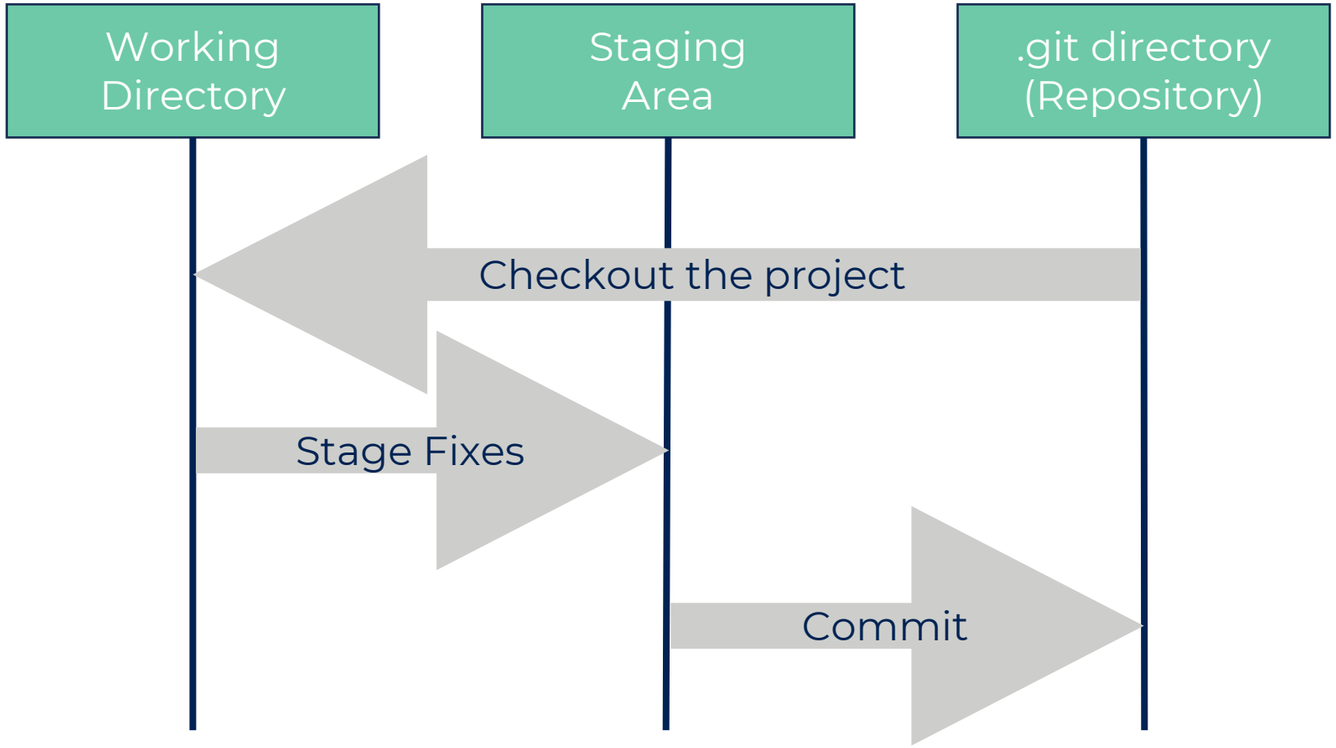
Рабочая копия файлов.

Staging Area

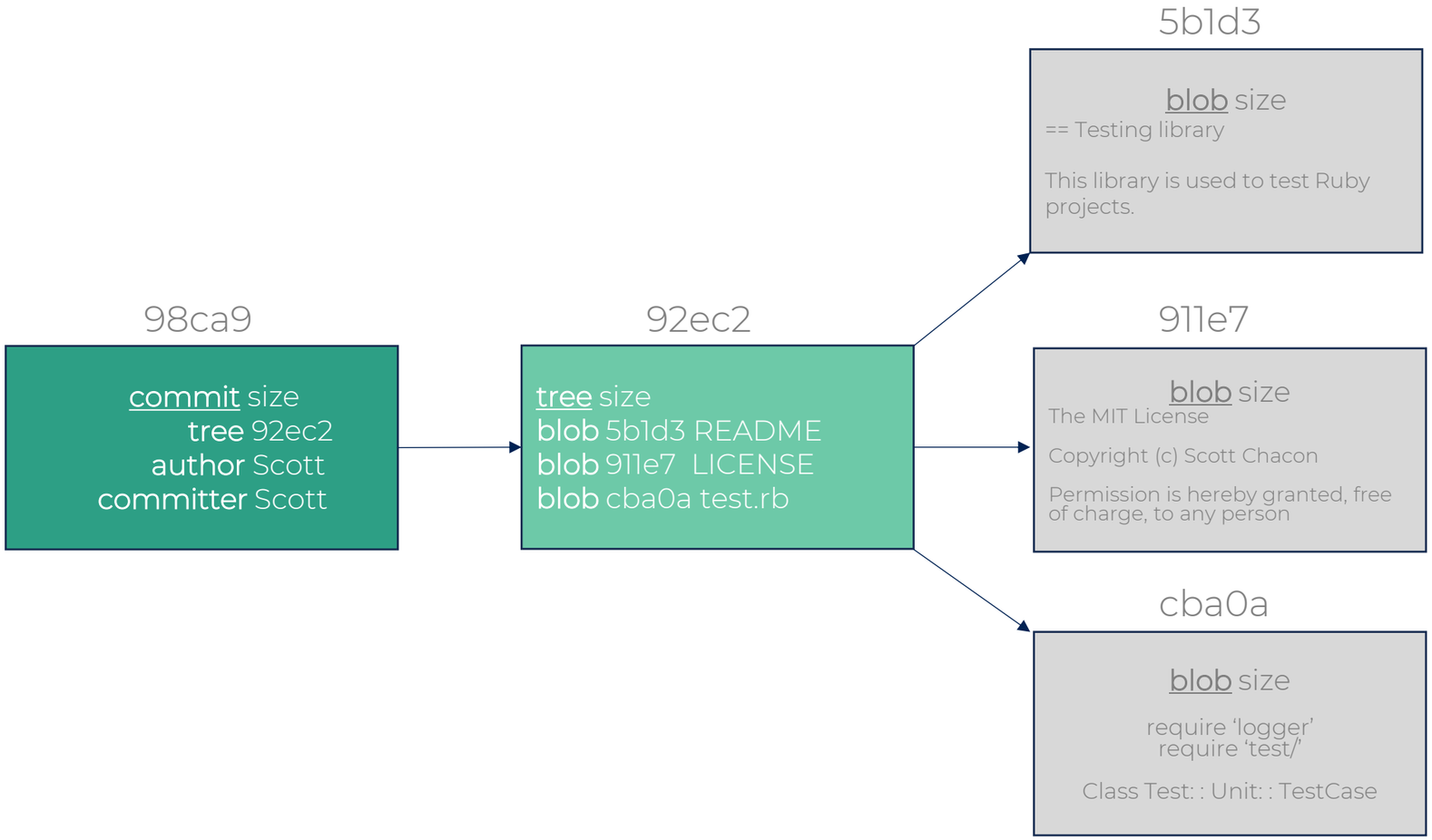
Область индексирования для следующего коммита.

Git Directory

Репозиторий с метаданными и объектами.



СТРУКТУРА ОБЪЕКТОВ GIT



- Blob-объекты для хранения содержимого файлов
- Tree-объекты для представления структуры каталогов
- Commit-объекты для связывания снимков

9

ГАРАНТИИ ЦЕЛОСТНОСТИ ДАННЫХ

SHA-1 хеширование

Каждый объект идентифицируется SHA-1 хешем.

Контентно-адресуемая ФС

Обращение к объектам по их хеш-сумме.

Неизменяемость истории

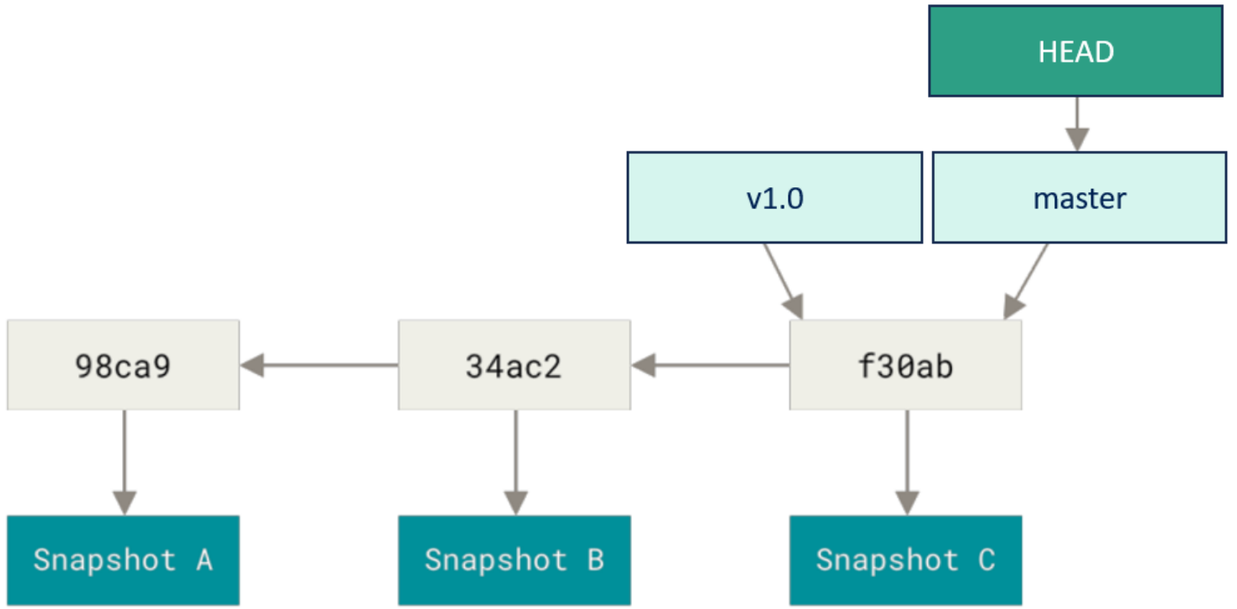
Невозможность незаметной модификации репозитория.

Целостность данных в Git обеспечивается через криптографическое хеширование.

10 ВЕТВЛЕНИЕ В GIT

- Ветки как перемещаемые указатели
- Легковесное создание веток
- Указатель HEAD

Модель ветвления в Git основана на использовании легковесных перемещаемых указателей на коммиты.



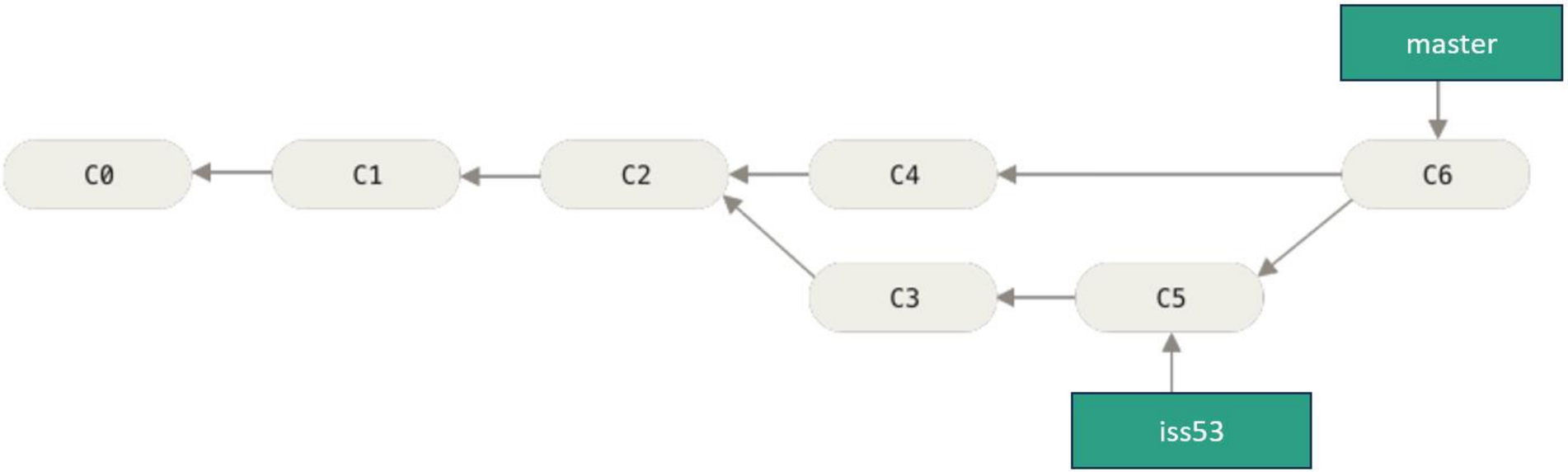
11 АЛГОРИТМЫ СЛИЯНИЯ ВЕТОК

Git поддерживает несколько стратегий слияния веток, каждая из которых предназначена для различных сценариев интеграции изменений.

Fast-forward слияние

Recursive стратегия

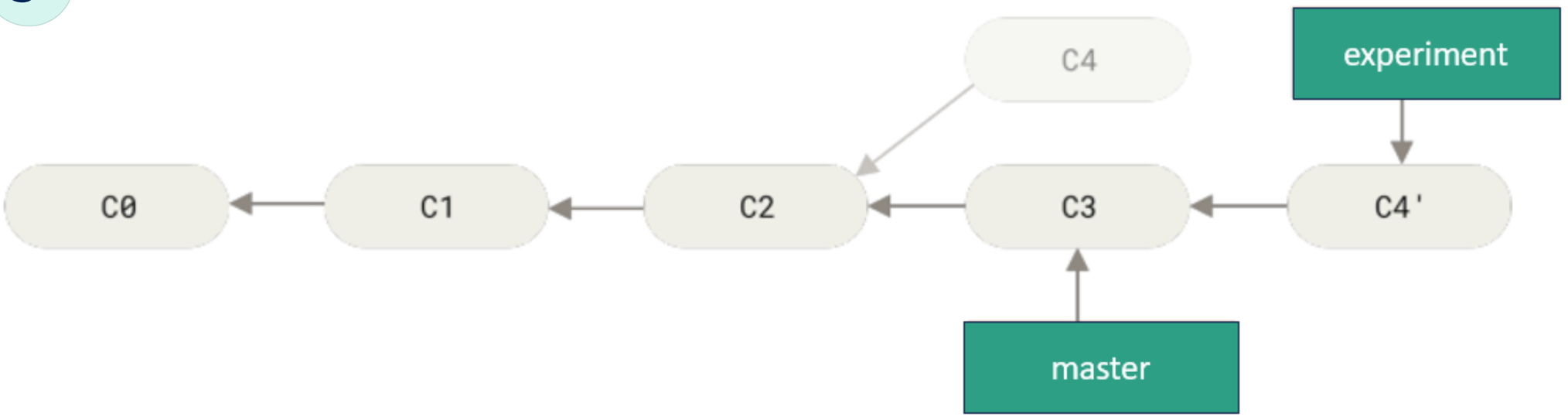
Коммиты слияния



12 АЛЬТЕРНАТИВНЫЙ ПОДХОД К ИНТЕГРАЦИИ ИЗМЕНЕНИЙ: ПЕРЕБАЗИРОВАНИЕ

Перебазирование (rebasing) предлагает мощный, но требующий осторожности способ интеграции изменений из одной ветки в другую.

1. Линеаризация истории
2. Применение коммитов
3. Ограничения использования



13

РАЗРЕШЕНИЕ КОНФЛИКТОВ СЛИЯНИЯ

Автоматическое и ручное разрешение

Маркеры конфликтов

<<<<<<<, =======, >>>>>>>

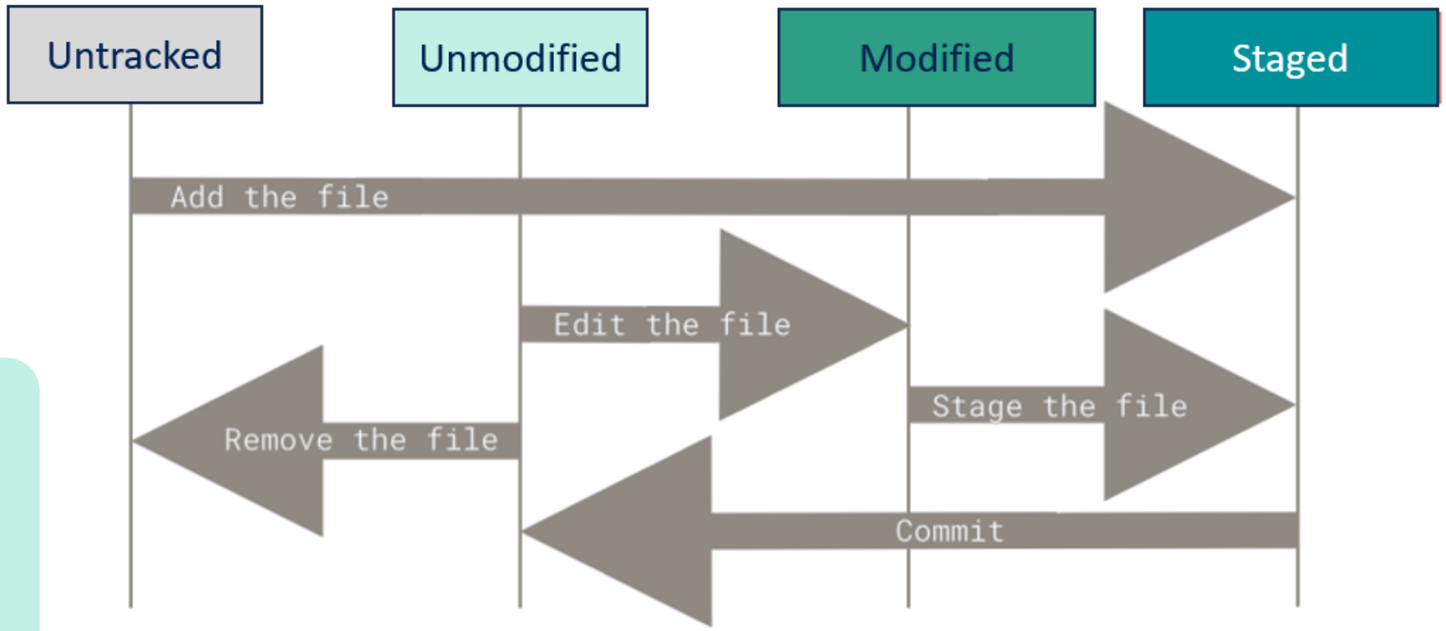
Инструменты для разрешения

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

ИНСТРУМЕНТЫ ОТМЕНЫ ИЗМЕНЕНИЙ В GIT

Git предоставляет гибкие механизмы для отмены изменений на различных этапах рабочего процесса, позволяя эффективно управлять историей проекта.

- Восстановление файлов
- Отмена индексации
- Модификация истории



Важно помнить, что изменение опубликованной истории коммитов может создать значительные проблемы для других разработчиков, работающих с тем же репозиторием.

15

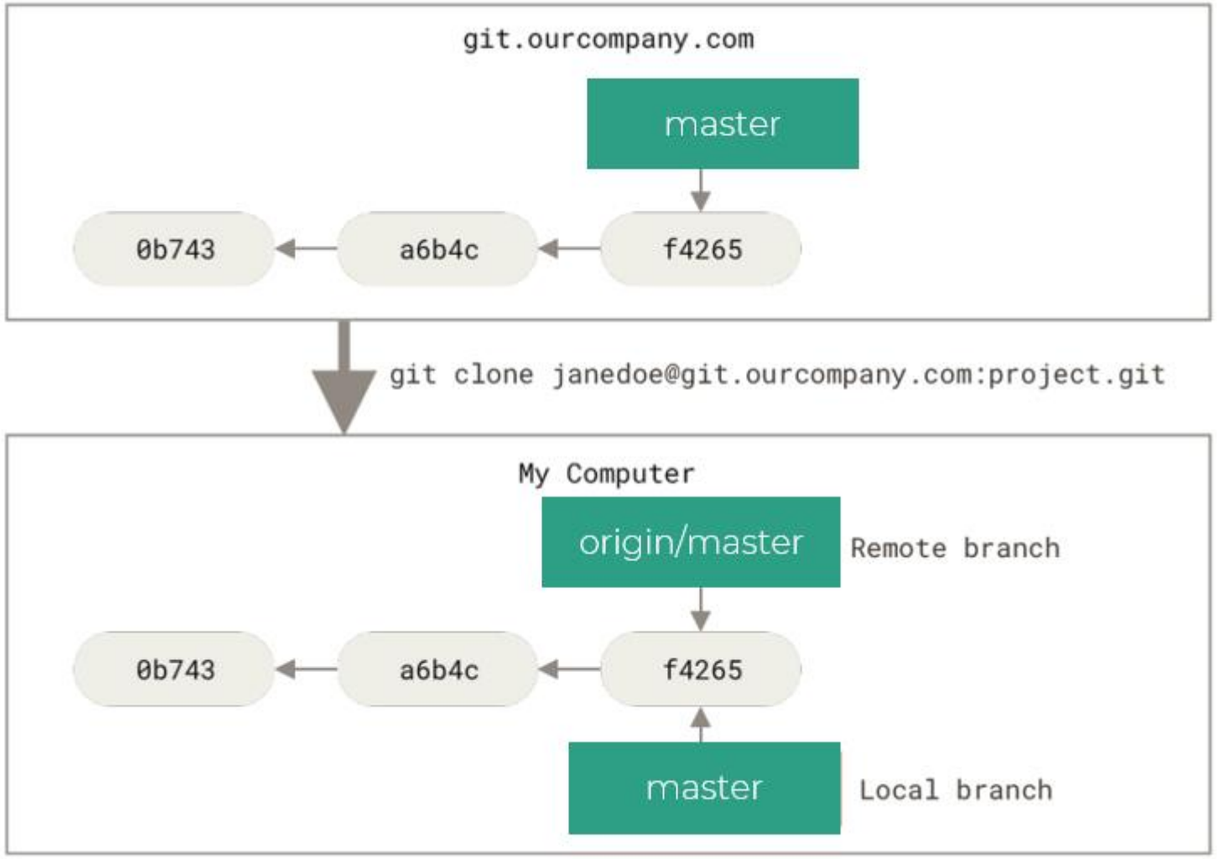
СИСТЕМА ТЕГИРОВАНИЯ В GIT

Теги в Git играют ключевую роль в отметке значимых точек в истории проекта, таких как версии релизов.

1. Аннотированные теги
2. Легковесные теги
3. Семантическое версионирование

СЕТЕВОЕ ВЗАИМОДЕЙСТВИЕ: УДАЛЕННЫЕ РЕПОЗИТОРИИ

Удаленные репозитории являются краеугольным камнем совместной разработки, обеспечивая обмен изменениями между разработчиками.



01

Клонирование

02

Ветки отслеживания

03

Синхронизация изменений

17

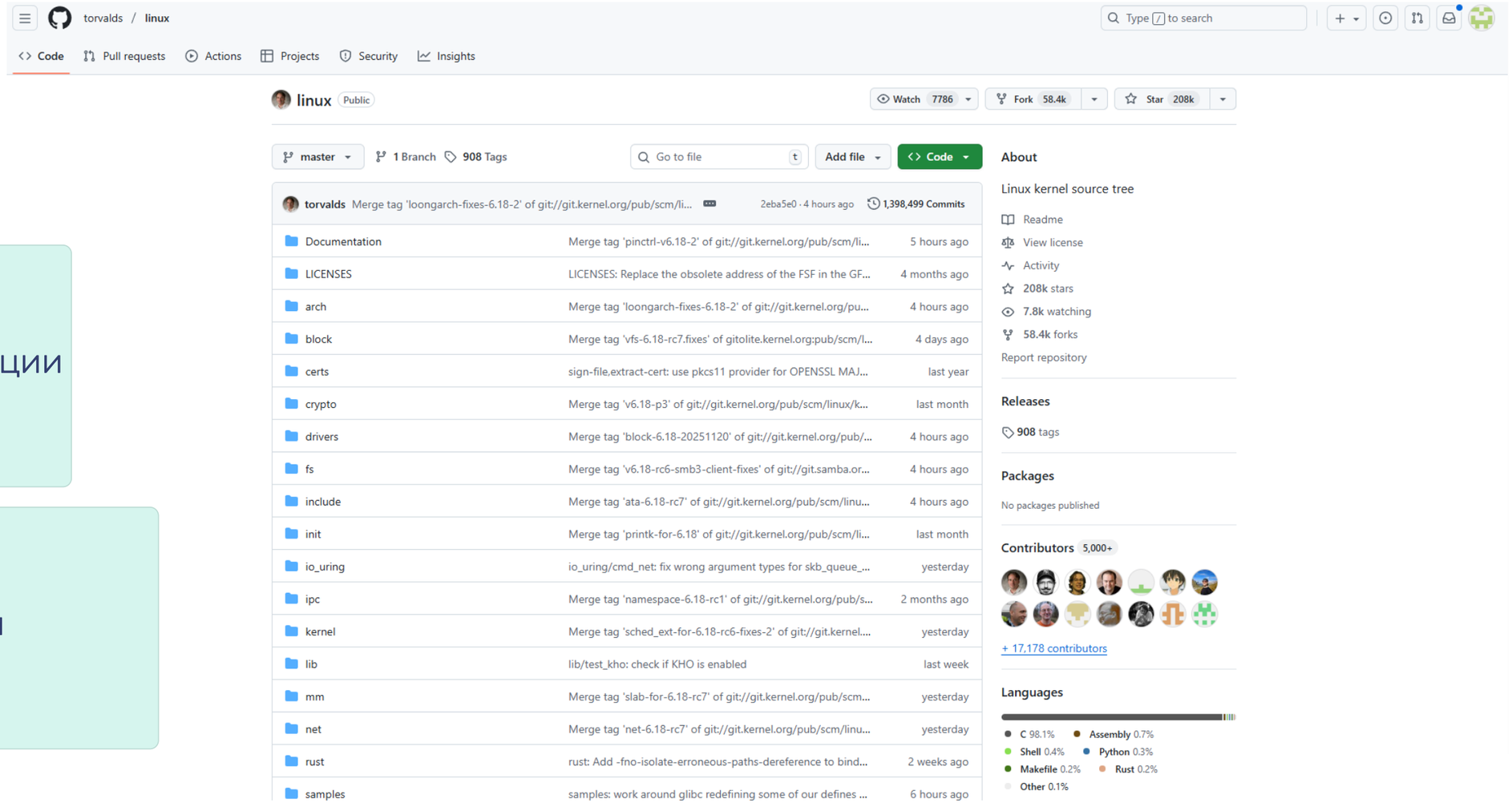
GITHUB КАК ЭКОСИСТЕМА РАЗРАБОТКИ

GitHub вышел за рамки простого хостинга Git-репозиториев, превратившись в мощную платформу для совместной разработки.

Веб-интерфейс

Инструменты коллаборации

Интеграция с сервисами



18

МОДЕЛЬ FORK И PULL REQUEST ДЛЯ РАСПРЕДЕЛЕННОЙ РАЗРАБОТКИ

Модель Fork и Pull Request является основополагающим рабочим процессом в open-source проектах, использующих GitHub, и способствует эффективной распределенной разработке.

1

Создание форка

Fork создает независимую персональную копию репозитория, позволяя разработчику свободно экспериментировать без влияния на основной проект.

2

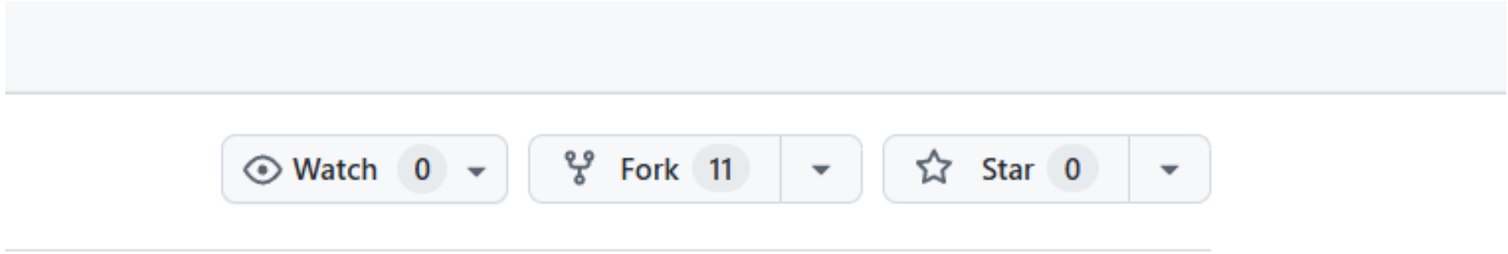
Изоляция изменений

Экспериментальные изменения разрабатываются в форке, обеспечивая полную изоляцию от основной кодовой базы.

3

Предложение изменений

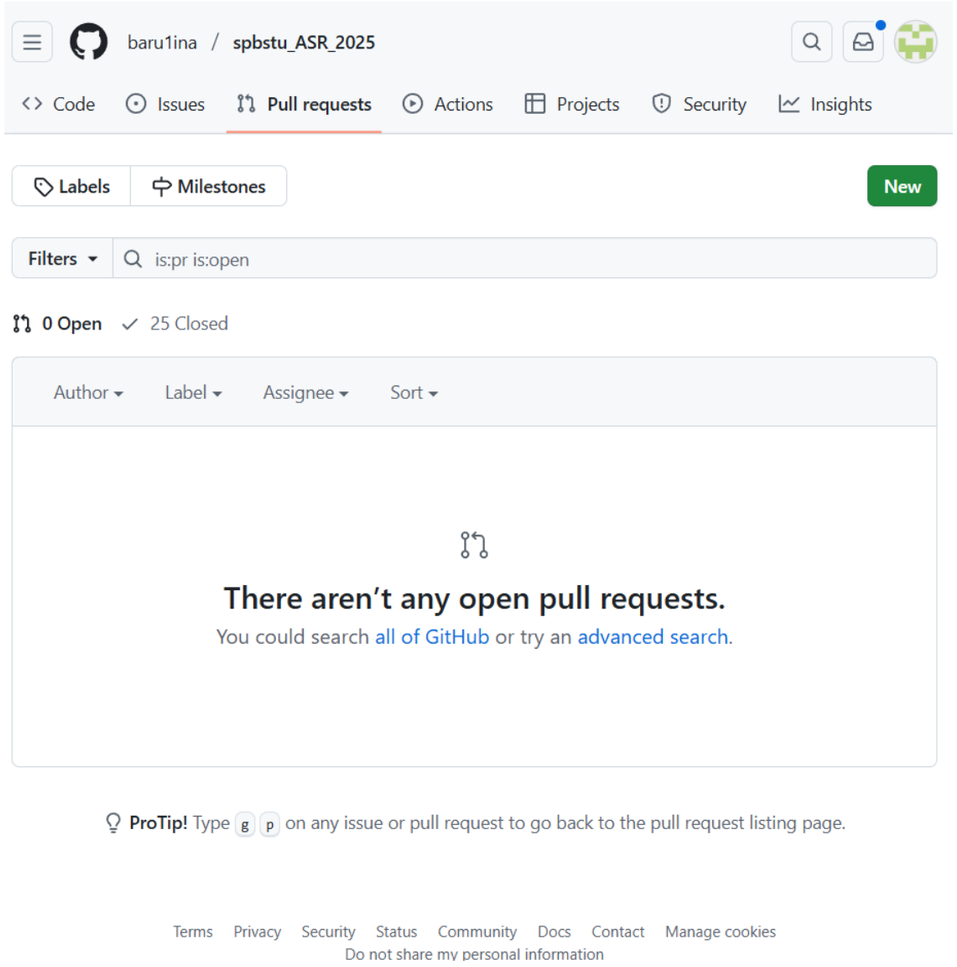
Pull Request — это формализованный запрос на интеграцию разработанных изменений из форка обратно в исходный репозиторий, запускающий процесс код-ревью.



19 ПРОЦЕСС КОД-РЕВЬЮ ЧЕРЕЗ PULL REQUEST

Pull Request служит центральной платформой для проведения код-ревью на GitHub.

- Обсуждение изменений
- Интеграция с CI/CD
- Автоматические проверки



20

ПОЛИТИКИ УПРАВЛЕНИЯ РЕПОЗИТОРИЕМ И ДОСТУПОМ

GitHub предоставляет мощные инструменты для настройки политик управления репозиторием, что позволяет формализовать и обезопасить процесс разработки.

Защищенные ветки

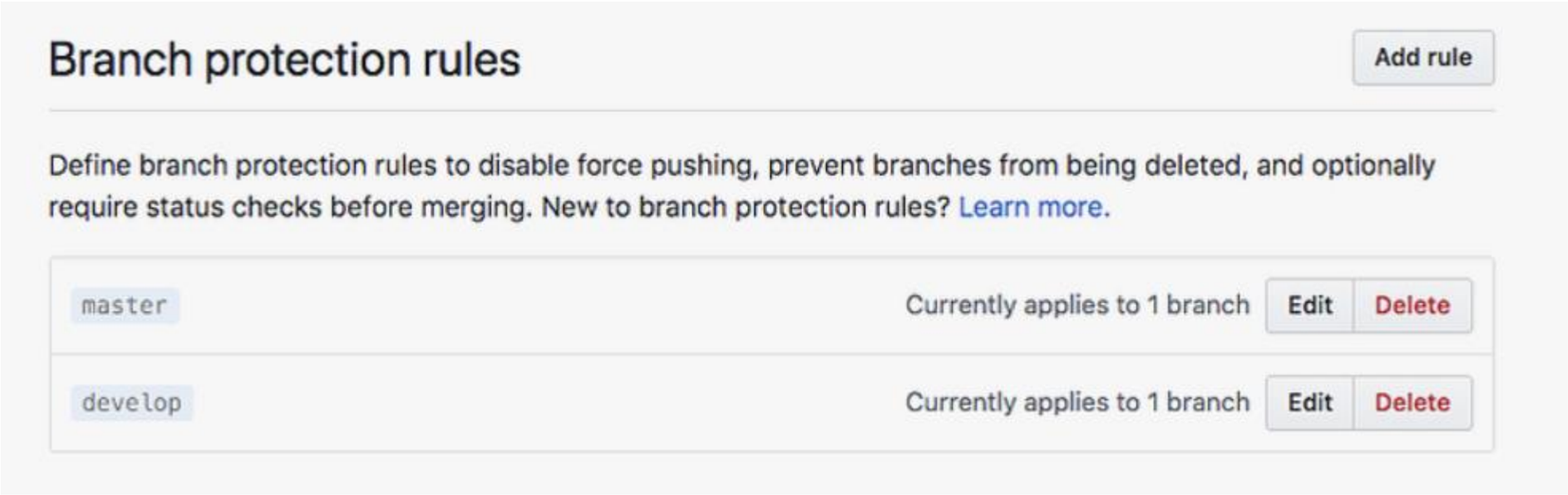
Позволяют установить строгие правила для критически важных веток, таких как `main` или `develop`.

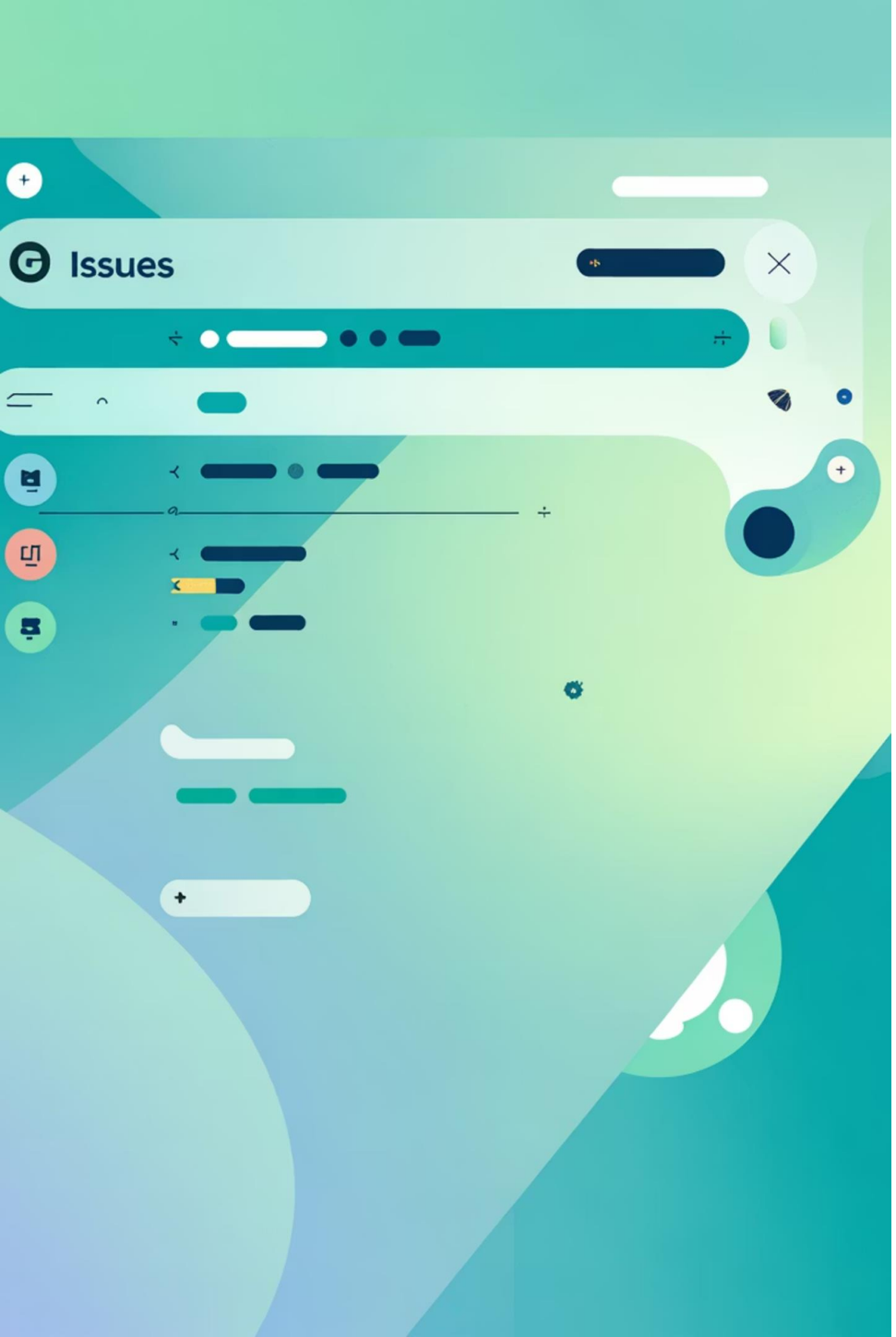
Требования к слиянию

Можно требовать обязательное код-ревью, прохождение всех тестов, а также линейную историю коммитов перед слиянием.

Управление доступом

Система тонкой настройки прав доступа обеспечивает разграничение ролей для каждого участника проекта, от чтения до администрирования.





21

ИНСТРУМЕНТЫ ПЛАНИРОВАНИЯ РАЗРАБОТКИ

Система Issues на GitHub предоставляет мощные инструменты для отслеживания ошибок, планирования новых функций и общего управления проектом, обеспечивая прозрачность и эффективность в процессе разработки.

1

Система отслеживания задач

Централизованное управление задачами, позволяющее легко приоритизировать, назначать и отслеживать прогресс каждой единицы работы.

2

Классификация задач

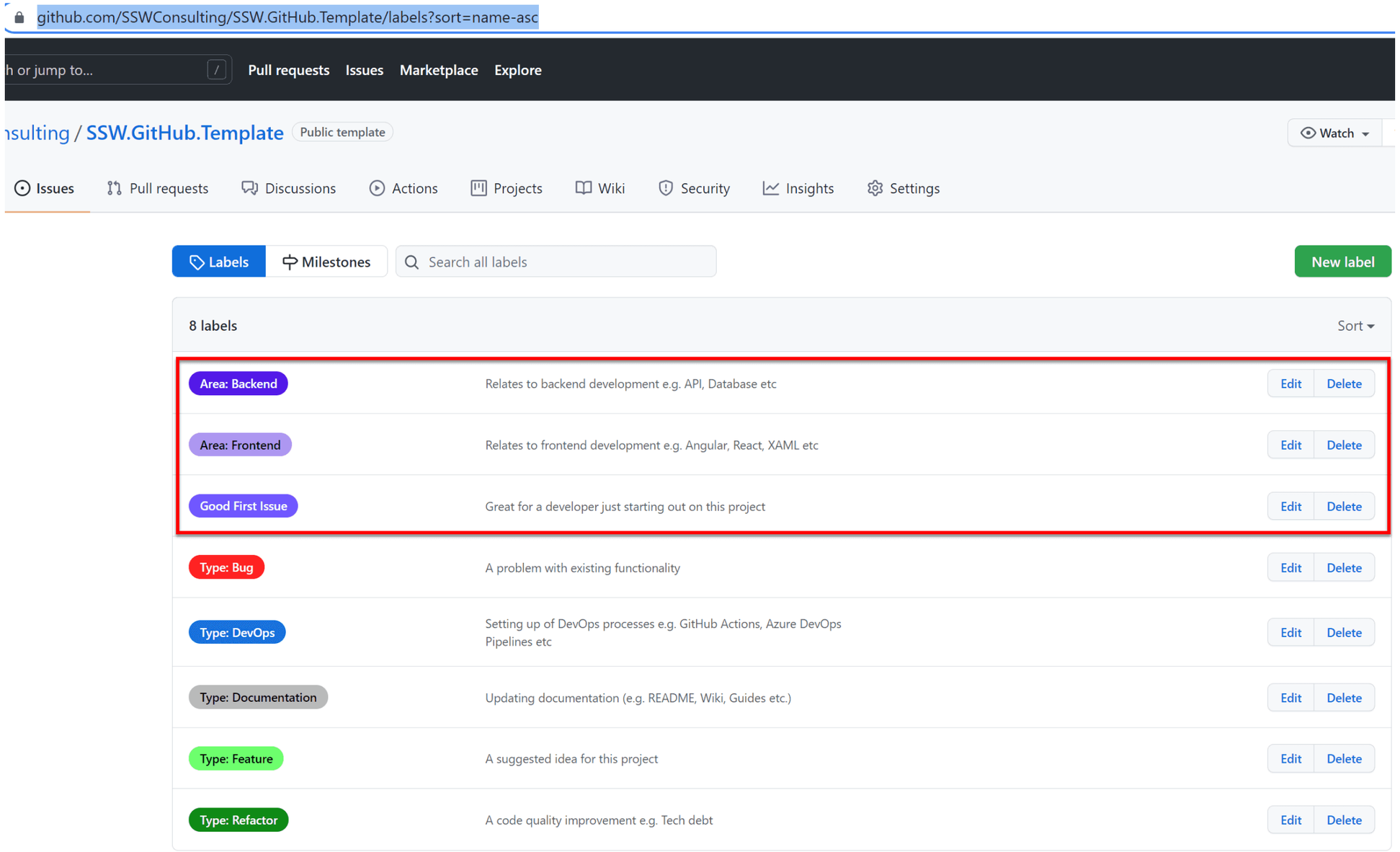
Использование настраиваемых меток для категоризации задач по типу, приоритету или компоненту, что упрощает навигацию и фильтрацию.

3

Планирование релизов

Вехи (Milestones) помогают организовать задачи в логические группы для предстоящих релизов или фаз проекта, обеспечивая четкое видение целей.

ИНСТРУМЕНТЫ ПЛАНИРОВАНИЯ РАЗРАБОТКИ



22 ИСКЛЮЧЕНИЕ ФАЙЛОВ ИЗ КОНТРОЛЯ ВЕРСИЙ

Файл .gitignore является критически важным компонентом любого Git-проекта, позволяя разработчикам точно определять, какие файлы и каталоги должны быть исключены из системы контроля версий.

Шаблоны игнорирования

Определение правил для игнорирования файлов с использованием различных шаблонов, включая подстановочные знаки (wildcards).

Глобальные настройки

Возможность создания глобального файла .gitignore для автоматического игнорирования файлов во всех репозиториях пользователя.

Автогенерируемые файлы

Исключение временных файлов, логов, скомпилированных артефактов и конфиденциальных данных, которые не должны быть частью репозитория.

Настройка .gitignore в начале проекта предотвращает случайное добавление нежелательных файлов и помогает поддерживать чистоту репозитория.


```
# Исключить все файлы с расширением .a
*.a

# Но отслеживать файл lib.a даже если он подпадает под исключение выше
!lib.a

# Исключить файл TODO в корневом каталоге, но не файл в subdir/TODO
/TODO

# Игнорировать все файлы в каталоге build/
build/

# Игнорировать файл doc/notes.txt, но не файл doc/server/arch.txt
doc/*.txt
```

РАСШИРЕННЫЕ ВОЗМОЖНОСТИ GIT

Git предлагает обширный набор инструментов, выходящий за рамки базовых операций, позволяя разработчикам глубоко анализировать историю, настраивать рабочие процессы и повышать производительность.

1

Визуализация истории

Использование `git log` с флагами для просмотра истории коммитов, включая граф ветвлений и статистику.

2

Поиск в истории изменений

Фильтрация истории по автору, дате, сообщению коммита или содержимому файлов для быстрого нахождения нужных изменений.

3

Кастомизация через псевдонимы

Создание коротких, пользовательских команд (псевдонимов) для автоматизации часто используемых последовательностей команд Git.

Эти расширенные функции значительно улучшают опыт работы с Git, делая его более мощным и адаптируемым инструментом для любого проекта.

ЭВОЛЮЦИЯ АРХИТЕКТУР VCS

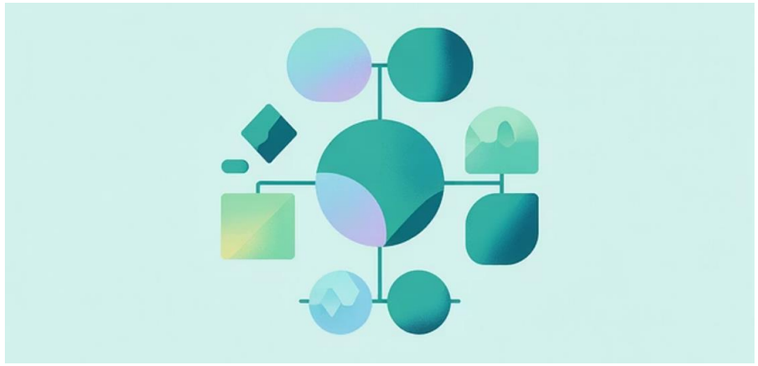
Сравнительный анализ систем контроля версий выявляет фундаментальные различия между централизованными и распределенными архитектурами, определяющие их эффективность и гибкость в современных процессах разработки.

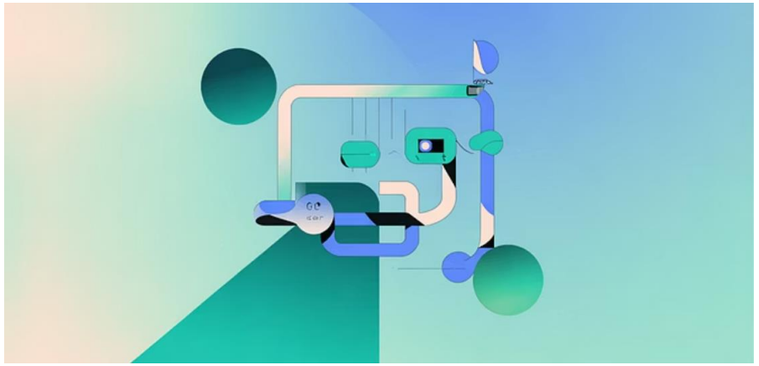
Характеристика	Централизованные VCS (SVN, CVS)	Распределенные VCS (Git, Mercurial)
Архитектура	Клиент-серверная модель	Распределенная модель (peer-to-peer)
Работа без сети	Невозможна	Полная функциональность
Производительность операций	Зависит от скорости сети и сервера	Локальные операции - максимальная скорость
Создание веток	Ресурсоемкая операция	Мгновенная операция
Резервное копирование	Единая точка отказа	Каждая копия - полная резервная копия
Рабочие процессы	Линейные, централизованные	Гибкие, нелинейные
Сложность обучения	Низкая	Высокая (более сложная модель)
Целостность данных	Зависит от сервера	Криптографическая гарантия через хэши
История изменений	Только на сервере	Полная история у каждого разработчика
Модель хранения	Дельта-изменения (различия)	Снимки состояния (snapshots)

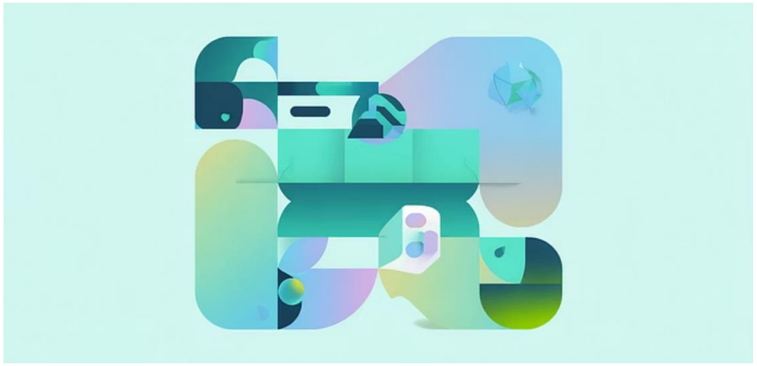
Распределенная архитектура Git обеспечивает превосходство в производительности, отказоустойчивости и адаптивности, что делает ее идеальным выбором для современных проектов разработки.

ИТОГИ РАССМОТРЕНИЯ АРХИТЕКТУРЫ GIT

Архитектура Git, основанная на инновационных принципах, заложила фундамент для эффективного управления версиями и совместной разработки, став золотым стандартом в индустрии программного обеспечения.







Эффективная модель хранения данных

Основанная на снимках состояния (snapshots), Git обеспечивает целостность данных через криптографические хеши.

Мощные механизмы ветвления и слияния

Легковесное создание веток и гибкие стратегии слияния поддерживают сложные рабочие процессы.

Комплексная экосистема

Интеграция с платформами, такими как GitHub, создает полноценную среду для collaborative software development.

Git и GitHub вместе формируют мощный инструмент, который оптимизирует процесс разработки, ускоряет код-ревью и улучшает общую координацию команды.

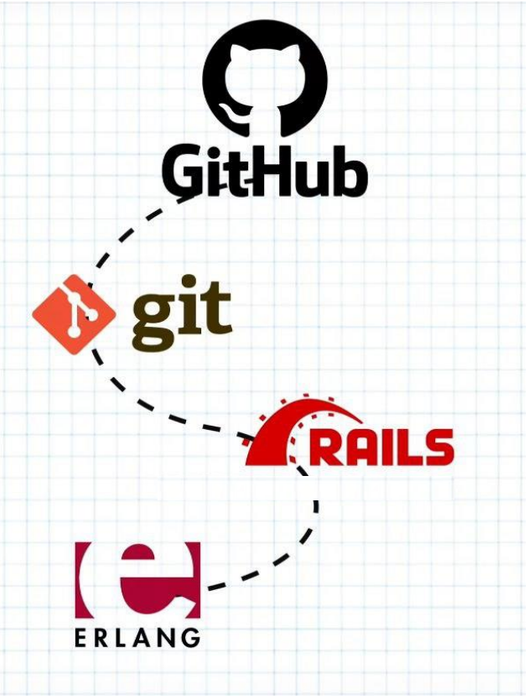
ИНТЕРЕСНЫЕ ФАКТЫ

Сервис был основан на системе контроля версий GIT, фреймворке RUBY ON RAILS и языке ERLANG

Первая версия была запущена в апреле 2008 года, а в 2018 их купили Microsoft и владеют по сей день

На лого Git.hub у кота не хвост, а щупальца

Этот логотип разработал Саймон Оаксли и выставил на продажу на Istock, Github выкупили его и переименовали из Octopus в Octocat и зарегистрировали как торговую марку. Никакого скрытого смысла – авторам просто понравился КОТЭ



Внутреннее устройство Git и GitHub

Презентацию подготовили:
студентки группы 504102/50201
Буталова Юлия
Худина Анастасия