СТИ
НИЯУ МИФИ
Северский технологический институт

УЧЕБНАЯ ДИСЦИПЛИНА

ВВЕДЕНИЕ В СПЕЦИАЛЬНОСТЬ

ЛЕКЦИЯ 6

Система контроля версий.Git

лектор: к.т.н., доцент каф. ЭАФУ
Иванов Константин Александрович

2025 г.

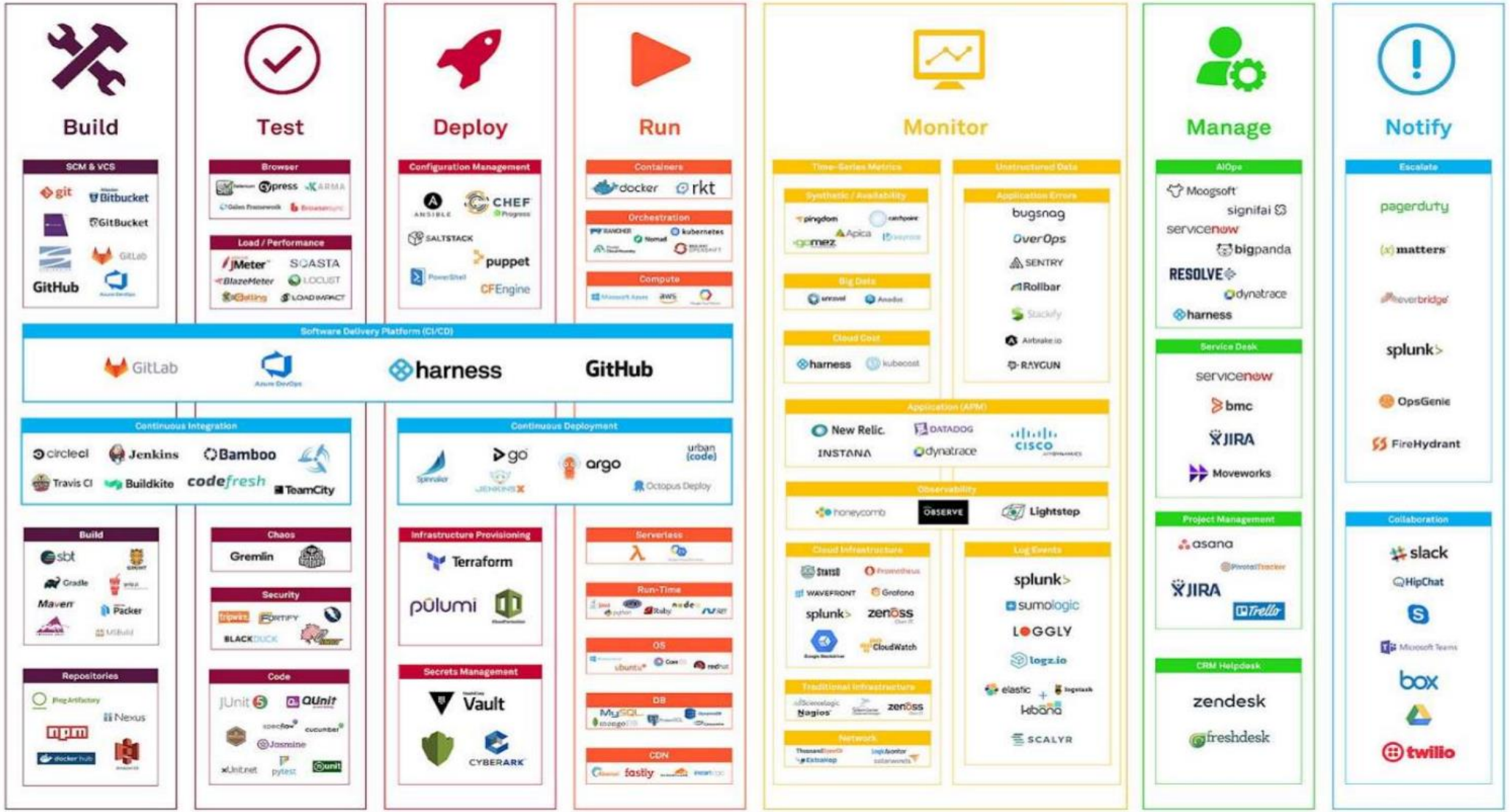
ПЛАН ЛЕКЦИИ

СТИ
НИЯУ МИФИ

1. Системе контроля версий
2. Git
3. GitHub

DevOps инструментарий

СТИ НИЯУ МИФИ



как хранить и не терять код программ?
как делиться кодом с товарищем?

СТИ
НИЯУ МИФИ

Есть множество способов хранения исходников программ:

- создать для них папку в разделе «Мои документы»,
- закидывать их в облако (Яндекс.Диск) и подписывать версии,
- загружать в «Избранное» в Telegram или «ВКонтакте»,
- можно записывать список изменений текстом в приватном Telegram-канале,
- можно деплоить проект с помощью простого скачивания и распаковки ZIP-архива с файлами вашей программы,
- можно сообщать о багах в вашем любимом софте сообществу анонимов в паблике в VK...

как хранить и не терять код программ?
как делиться кодом с товарищем?

СТИ
НИЯУ МИФИ

Контроль версий (англ. Version control), также известный как контроль исходного кода (англ. Source control) - это практика отслеживания и управления изменениями в программном коде.

Системы контроля версий (англ. version control systems) -

это программные инструменты, которые помогают группам разработчиков управлять изменениями исходного кода с учетом хронологии.

Главные возможности системы контроля версий:

- Полная история изменений каждого файла за длительный период
- Возможность отслеживать каждое изменение, внесенное в программное обеспечение

Репозиторий

СТИ
НИЯУ МИФИ

Репозиторий — место, где хранятся и поддерживаются какие-либо данные. Чаще всего данные в репозитории хранятся в виде файлов, доступных для дальнейшего распространения по сети.

Также под репозиторием понимается каталог файловой системы, в котором могут находиться:

- сами контролируемые файлы
- журналы конфигураций сборки проекта
- журнал операций, выполняемых над репозиторием

Репозиторий

СТИ
НИЯУ МИФИ

Управление исходным кодом (source control management, SCM) используется для отслеживания изменений в репозитории исходного кода.

SCM отслеживает текущую историю изменений в базе кода и помогает разрешать конфликты при объединении обновлений от нескольких участников.

Виды систем контроля версий

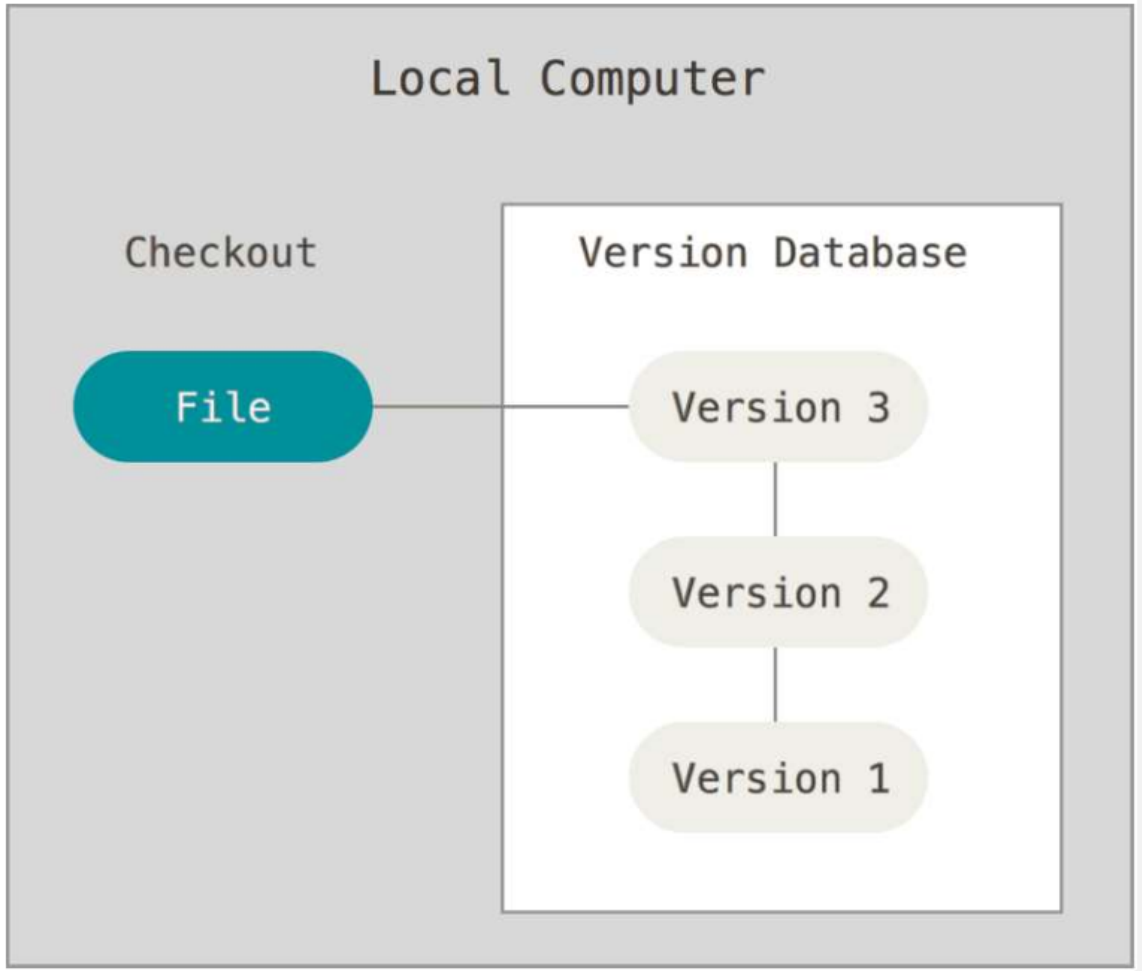
СТИ НИЯУ МИФИ

Локальные системы контроля версий

Одной из наиболее известных VCS такого типа является rcs (Revision Control System).

Утилита основана на работе с наборами патчей между парами версий (патч — файл, описывающий различие между файлами), которые хранятся в специальном формате на диске.

Она позволяет пересоздать любой файл на любой момент времени, последовательно накладывая патчи.

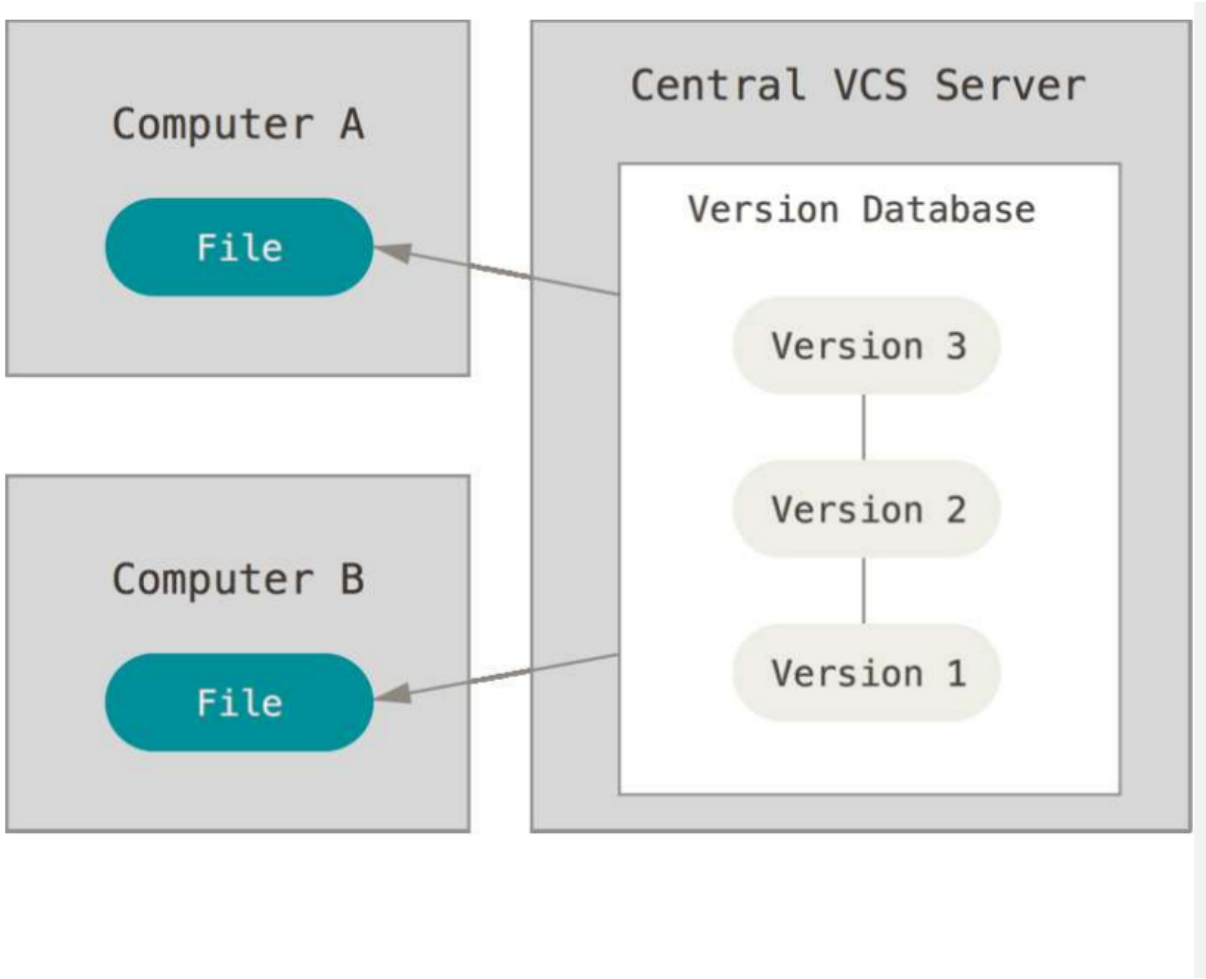


Виды систем контроля версий

СТИ НИЯУ МИФИ

Централизованные системы контроля версий

В таких системах, например Apache Subversion (SVN), есть центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые получают копии файлов из него.



Виды систем контроля версий

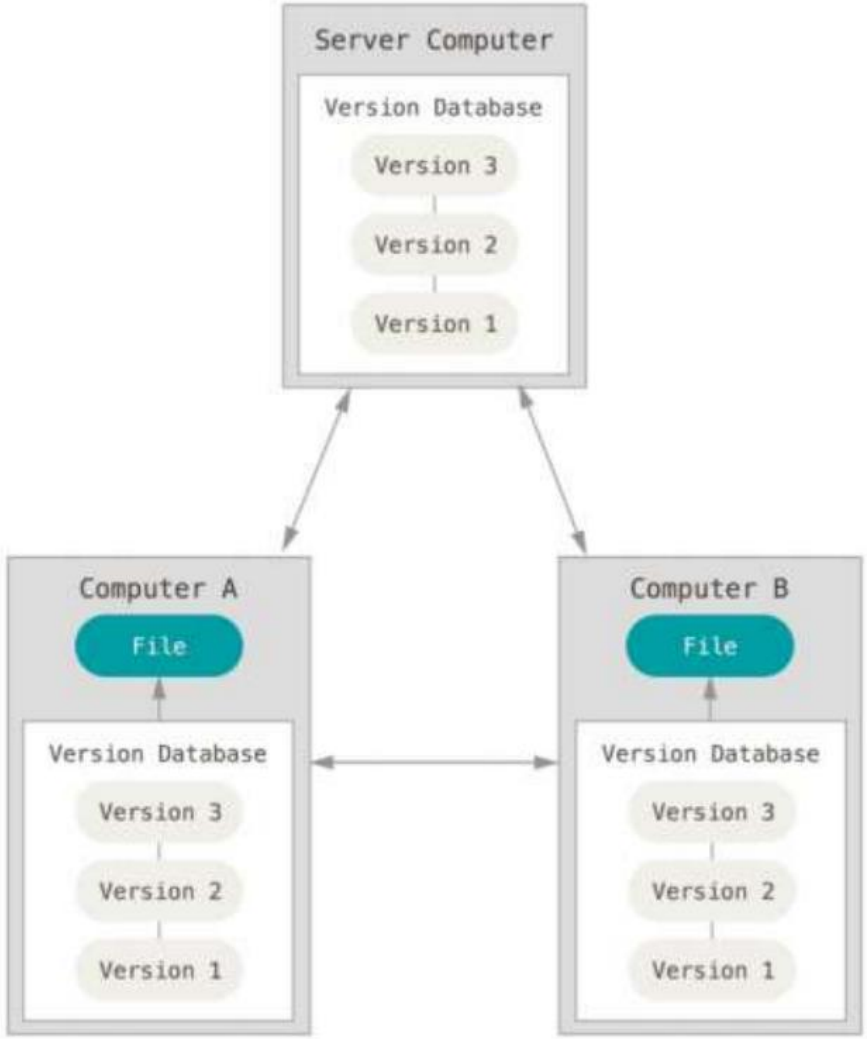
СТИ
НИЯУ МИФИ

Распределённые системы

контроля версий

В таких системах как Git, Mercurial клиенты не просто выгружают последние версии файлов, а полностью копируют репозиторий.

При этом можно выделить центральный репозиторий, в который будут отправляться изменения из локальных и с ним же эти локальные репозитории будут синхронизироваться.



История Git

СТИ
НИЯУ МИФИ

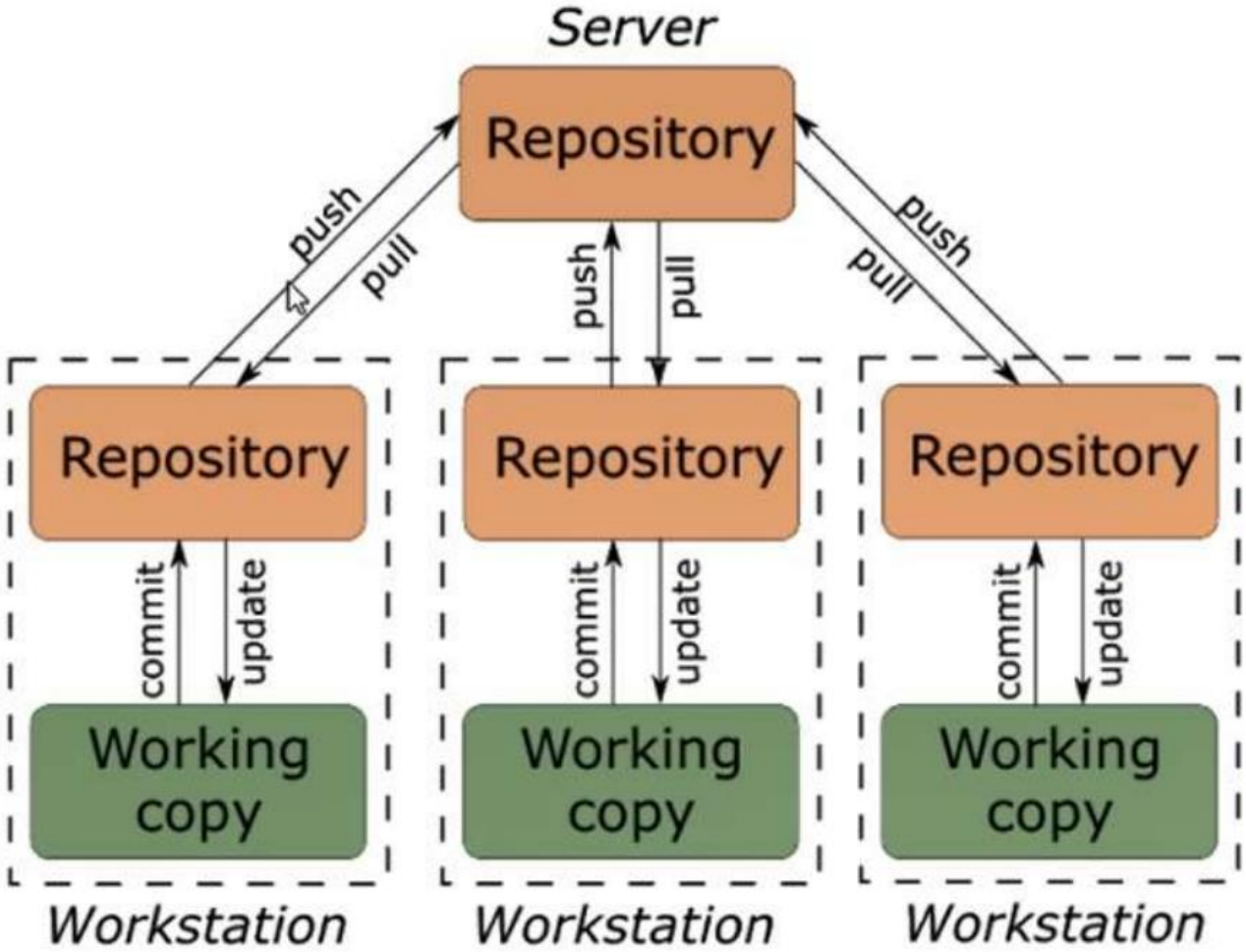
В 2002 году проект ядра Linux начал использовать проприетарную децентрализованную VCS BitKeeper.
В 2005 году отношения между сообществом разработчиков ядра Linux и коммерческой компанией, которая разрабатывала BitKeeper, прекратились, и бесплатное использование утилиты стало невозможным.
Так началась разработка Git...

Система контроля версий Git

СТИ
НИЯУ МИФИ

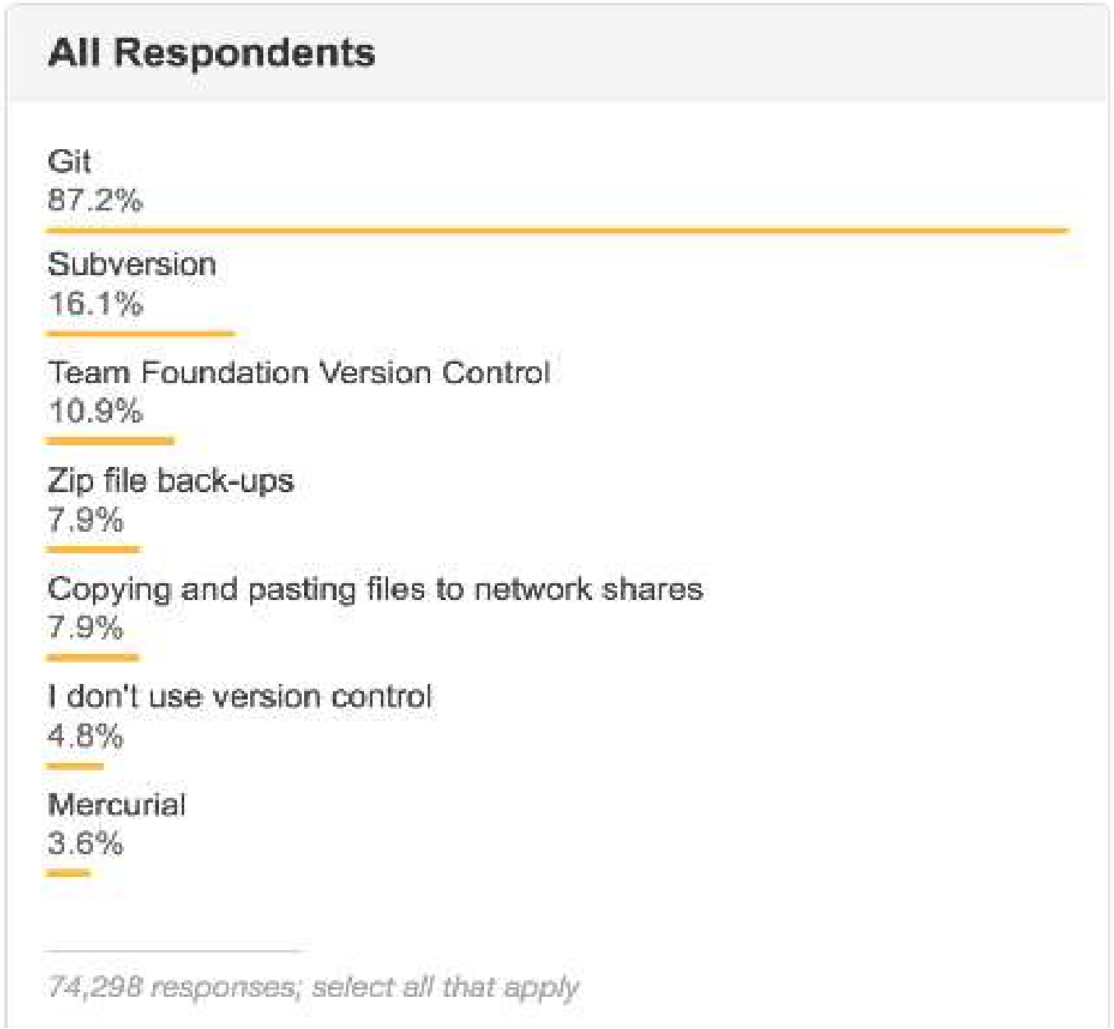
- Скорость
- Понятная архитектура
- Хорошая поддержка нелинейной разработки (тысячи параллельных веток)
- Полная децентрализация
- Возможность эффективного управления большими проектами (ядро Linux) :

-скорость работы и разумное использование дискового пространства



Система контроля версий Git

СТИ
НИЯУ МИФИ



(опрос 2018)

Основное отличие Git от любой другой системы контроля версий — это **подход к работе со своими данными**.

Концептуально, большинство других систем хранят информацию в виде **списка изменений в файлах**.

Эти системы (CVS, Subversion, Perforce и т.д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле по времени (это называют контролем версий, основанным на различиях).

Основное отличие Git от любой другой системы контроля версий — это **подход к работе со своими данными**.

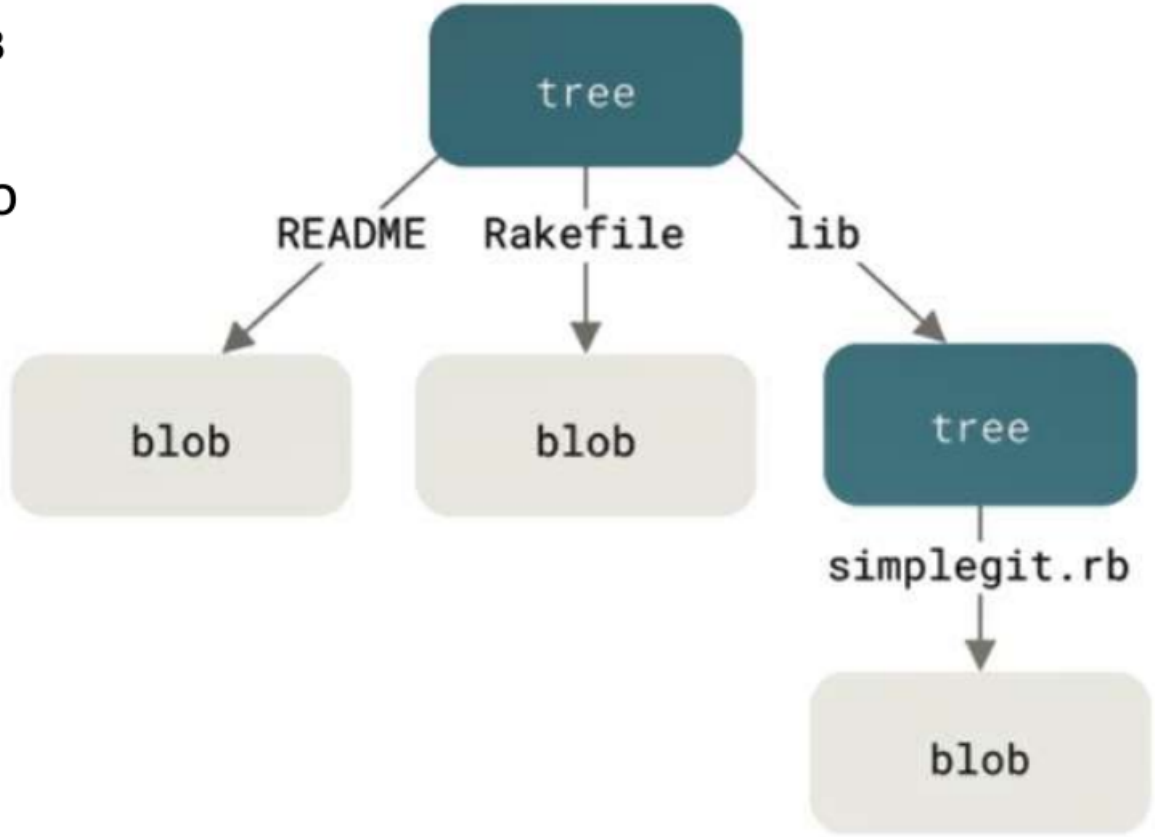
Подход Git к хранению данных больше похож на **набор снимков** мини-файловой системы.

Каждый раз, когда вы делаете коммит, т.е. сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

Принцип работы Git

СТИ
НИЯУ МИФИ

Когда файл добавляется для отслеживания в Git, он сжимается с помощью алгоритма сжатия zlib. Результат хэшируется с помощью хэш-функции SHA-1. Этот уникальный хэш соответствует содержимому в этом файле. Git хранит его в базе объектов, которая находится в скрытой папке .git/objects. Имя файла это сгенерированный хэш, а файл содержит сжатый контент. Такие файлы называются блобами и создаются каждый раз при добавлении в репозиторий нового файла (изменённой версии существующего файла).

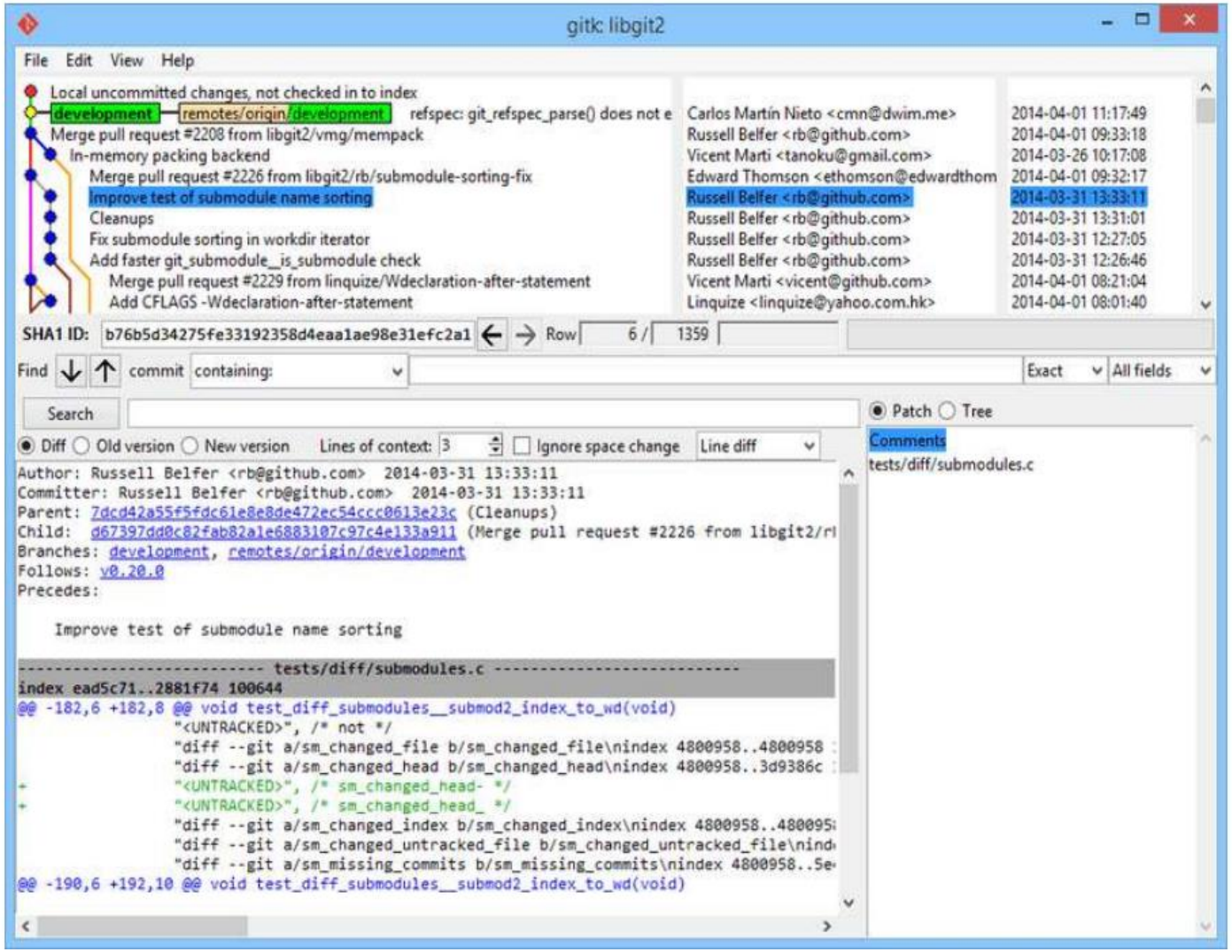


Git

Git – программа имеет интерфейс командной строки (CLI),

но к ней есть несколько готовых интерфейсов GUI для удобства.

Графический интерфейс инструмента gitk



Репозиторий

СТИ
НИЯУ МИФИ

Чтобы git начал следить за изменениями, надо сказать ему за чем именно наблюдать (не будет же он за всеми файлами следить!)

Покупки

Для этого необходимо создать репозиторий в нужной нам папке. Например, сделаем новую папку «Покупки»

$ mkdir Покупки

перейдем в нее с помощью команды

$ cd Покупки

создадим наш первый репозиторий

$ git init

Теперь git сможет следить за всеми файлами в папке «Покупки»

Создаем изменения

Создадим новый файлик с покупками «Продукты» и запишем первые заказы.

Покупки

Продукты

Молоко
Яблоки
Пельмени

Отслеживаем изменения

Файл создали, а дальше

что? Как git его видит?

Чтобы это узнать, есть команда

$ git status

которая нам скажет, что файл «Продукты» не отслеживается,

давайте это исправим!

Покупки

Продукты

Молоко
Яблоки
Пельмени

Отслеживаем изменения

СТИ
НИЯУ МИФИ

Чтобы **git** начал следить за нашим файлом надо явно добавить его с помощью команды

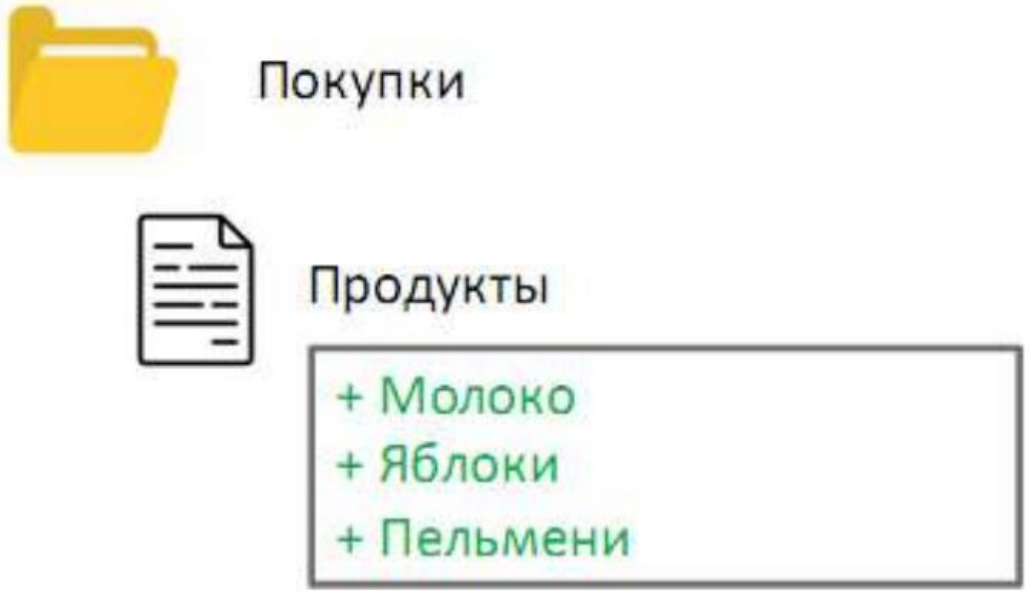
$ git add Продукты

Теперь команда

$ git status

покажет нам, что файл можно закоммитить (to be committed)

Commit - совершить … что это значит?



Репозиторий

предполагается, что вы будете хранить в нём файлы с исходным кодом и какие-нибудь дополнительные материалы - необходимую для GUI или вёрстки графику (картинки, иконки, звуки).

Репозитории могут быть публичными и приватными,

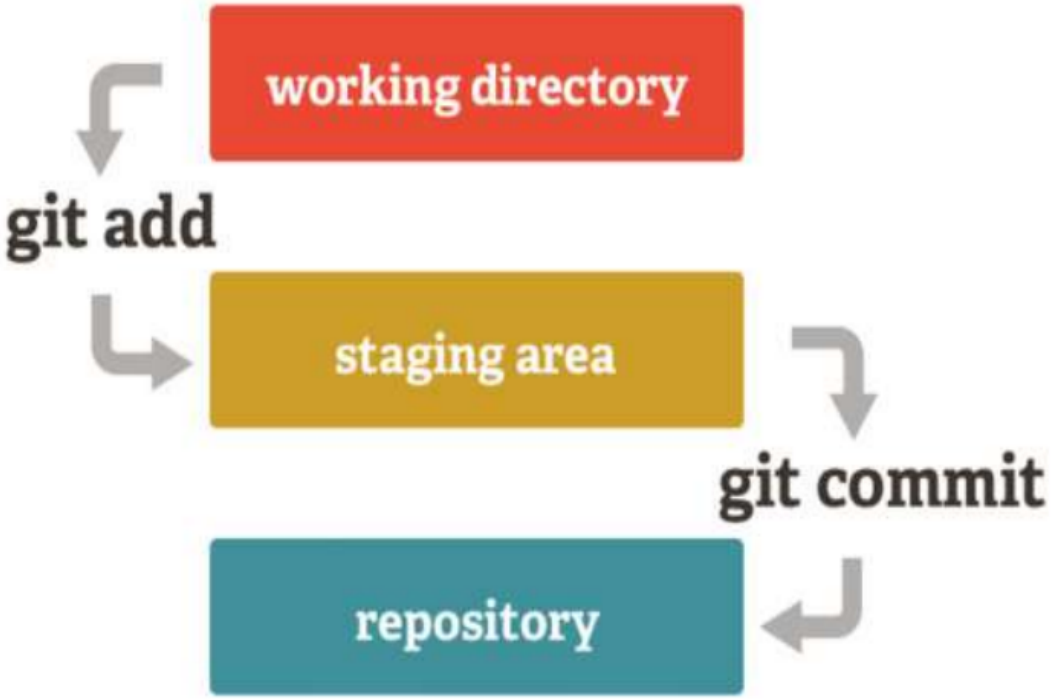
в них можно создавать другие папки и отслеживать изменения версий.

Управлять своими репозиториями можно прямо через интерфейс сайта GitHub, командную строку, десктопное приложение GitHub или различные средства разработки (IDE).

Состояния в git

есть три основных состояния, в которых могут находиться ваши файлы:

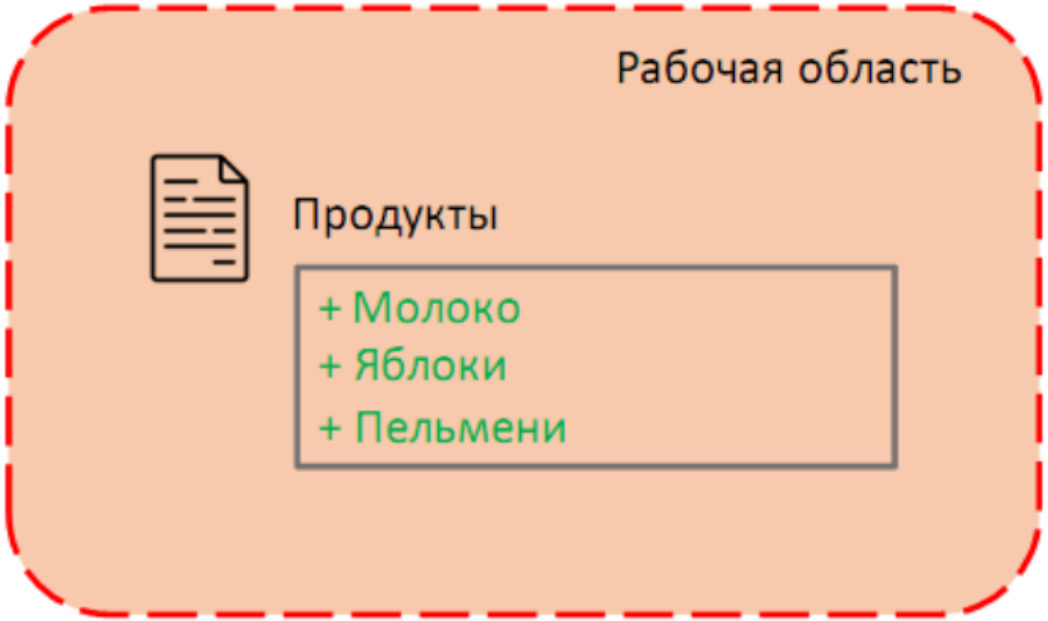
- к **изменённым** (modified) относятся файлы, которые поменялись, но ещё не были зафиксированы.
- **индексированный** (staged) — это изменённый файл в его текущей версии, отмеченный для включения в следующий коммит.
- **зафиксированный** (committed) значит, что файл уже сохранён в вашей локальной базе.



Рабочая область

или рабочая копия, или working directory – пространство в репозитории, является снимком одной версии проекта.

Файлы извлекаются из сжатой базы данных в каталоге Git и помещаются на диск, для того чтобы их можно было использовать или редактировать.

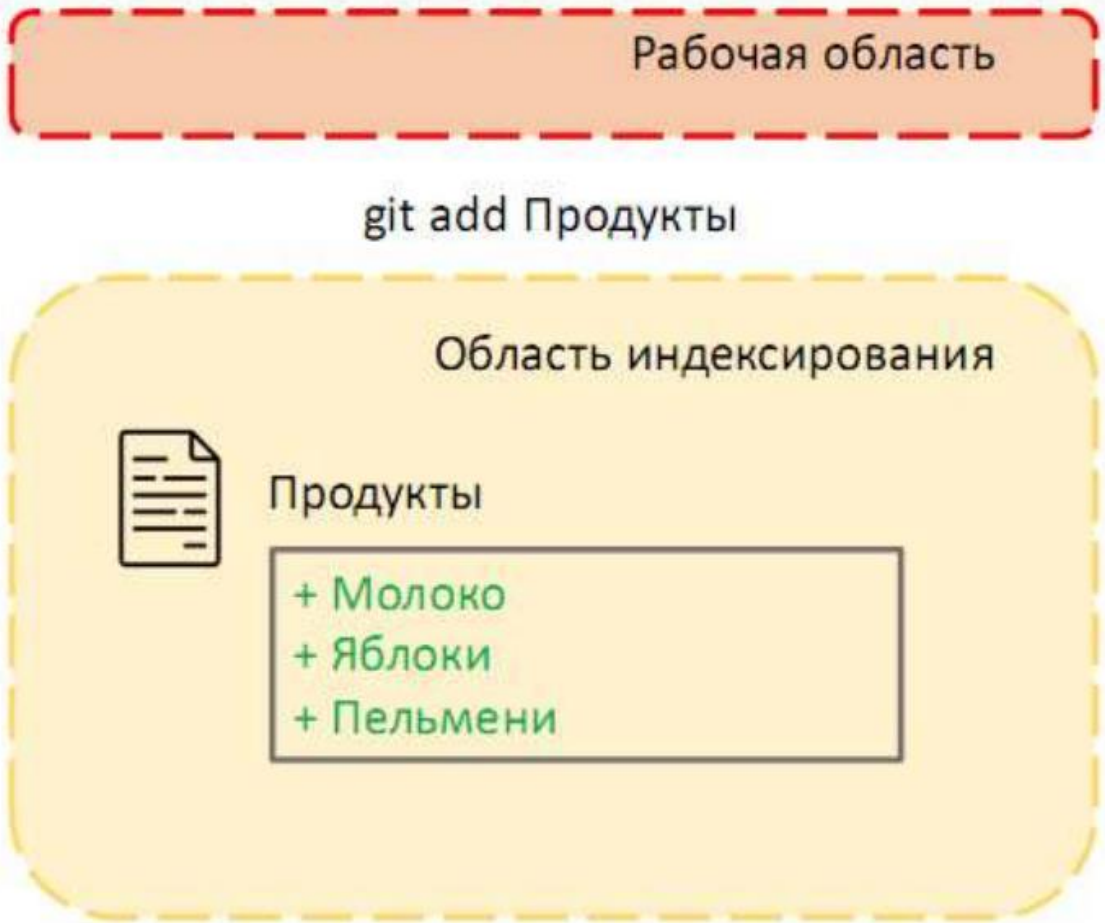


Область индексирования

Область индексирования, или индекс – заготовка для коммита, которую потом можно сохранить в истории.

Просто выберите понравившиеся изменения и добавьте их в индекс с помощью команды

$ **git add**



Каталог git

СТИ
НИЯУ МИФИ

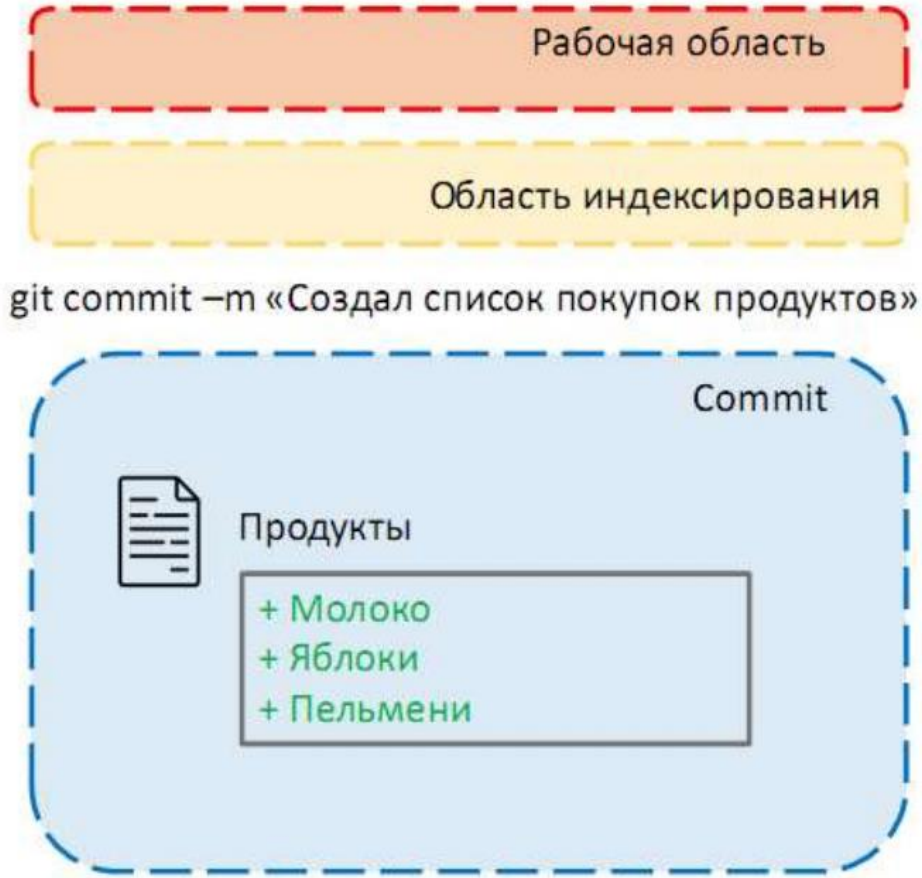
Каталог git – это цепочка сохраненных изменений (коммитов) в репозитории, а сам коммит – это и есть сохраненное состояние репозитория в какой-то момент времени.

Чтобы из индекса сделать новый коммит достаточно сделать команду

$ git commit

И не забудьте указывать, что вы сделали в этом коммите с помощью опции –m

$ git commit –m «Создал список покупок продуктов»



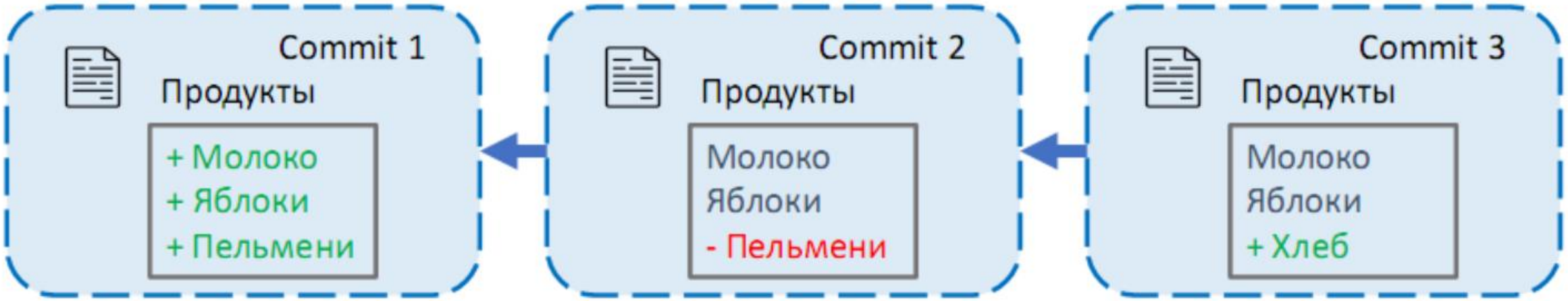
Каталог git

СТИ НИЯУ МИФИ

Базовый подход в работе с Git выглядит так:

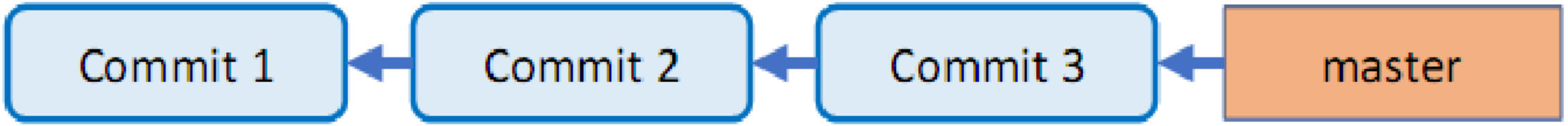
- Изменяете файлы вашей *рабочей копии*.
- Выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки только этих изменений в *индекс*.
- Когда вы делаете коммит, используются файлы из собранного индекса, и этот снимок сохраняется в ваш *каталог Git*.

Ветка (branch)



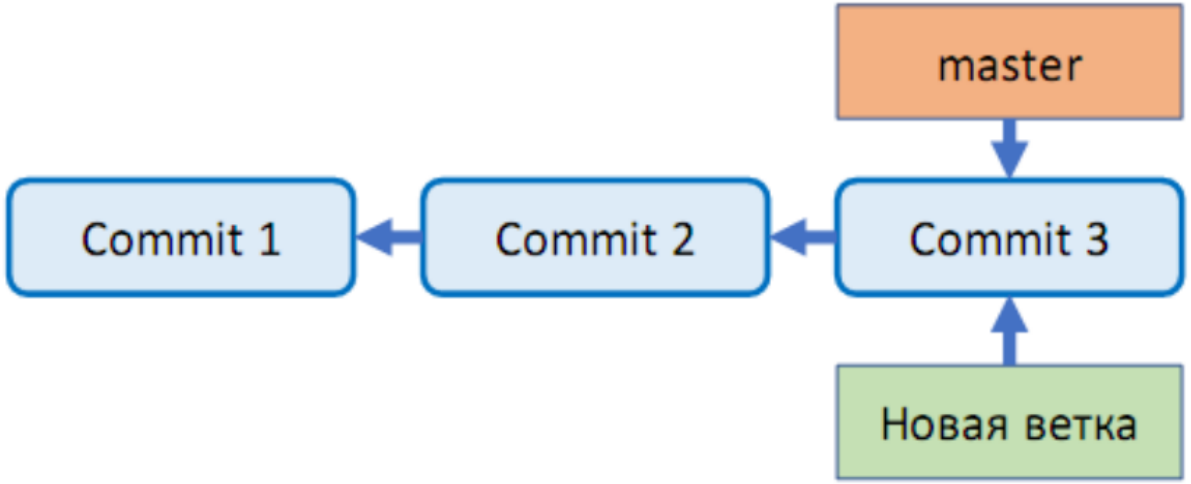
Коммиты неразрывно связаны друг с другом, последовательность коммитов называется веткой. Ветка нужна для того, чтобы понять, какие изменения нужны, чтобы получить текущую версию файла.

Основная ветка (master)



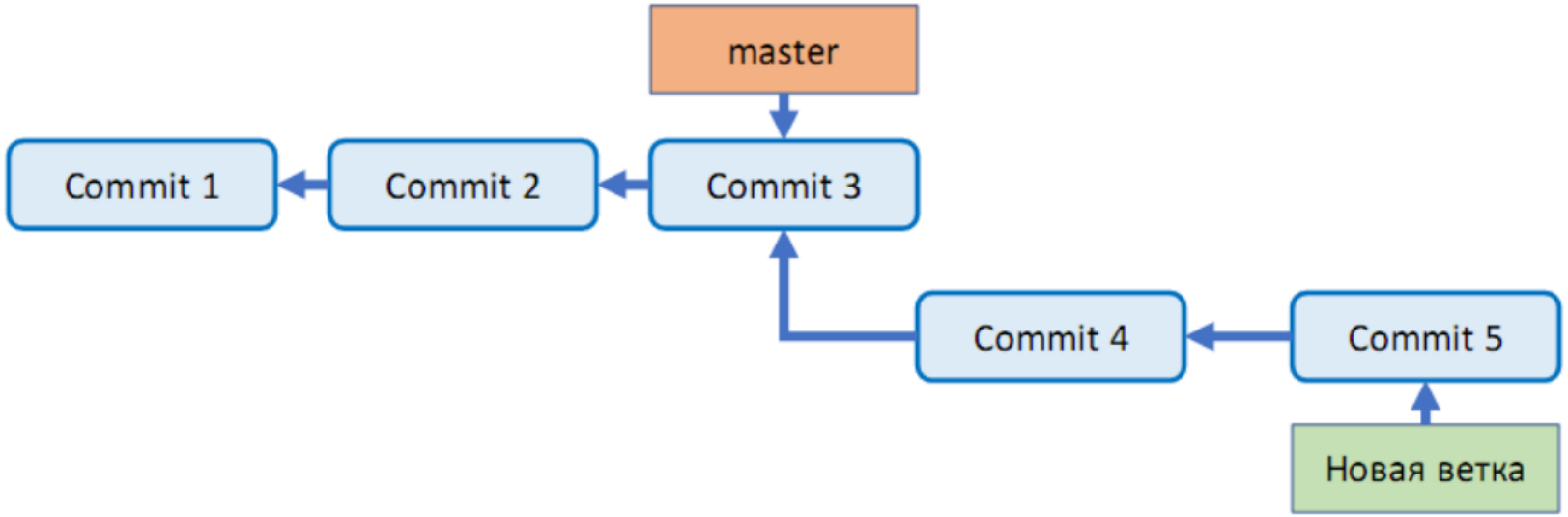
Когда вы создаете новый репозиторий, то вместе с ним создается и основная ветка, в которую и будут добавляться все новые коммиты. Обычно она называется **master** или **main**.

Множество веток



Когда работаешь с репозиторием в одиночку, можно обойтись и одной веткой, но система контроля версий создавалась для совместной разработки кода многими людьми, поэтому для того, чтобы не мешать друг другу можно создать отдельные ветки (пути изменений) под различные нужды.

Множество веток



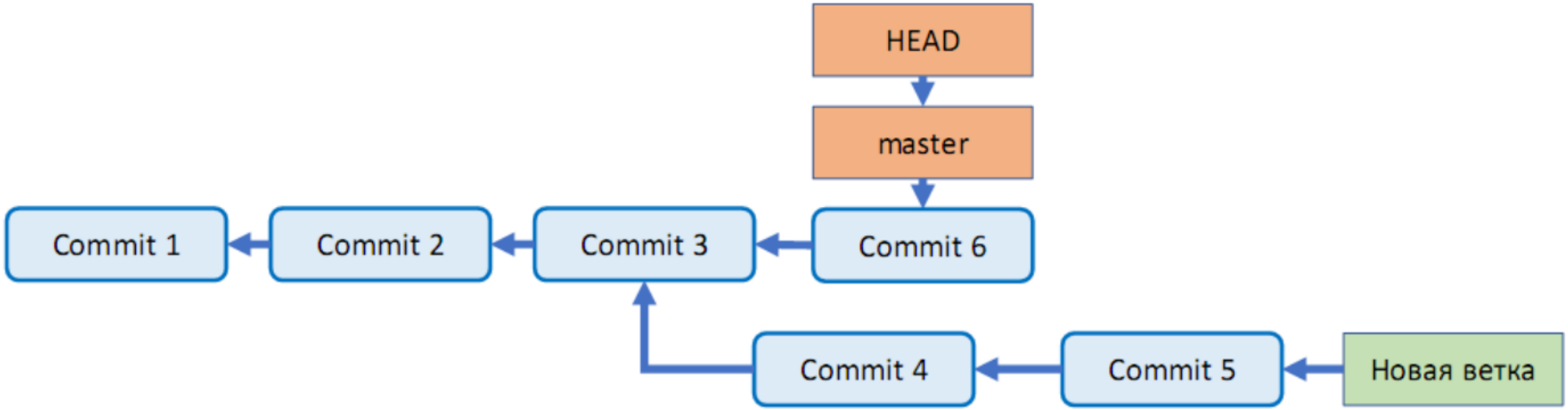
Ветку можно создать с помощью команды

$ git branch новая_ветка

Сама ветка по сути – это указатель на последний коммит,

к которому добавятся новые правки.

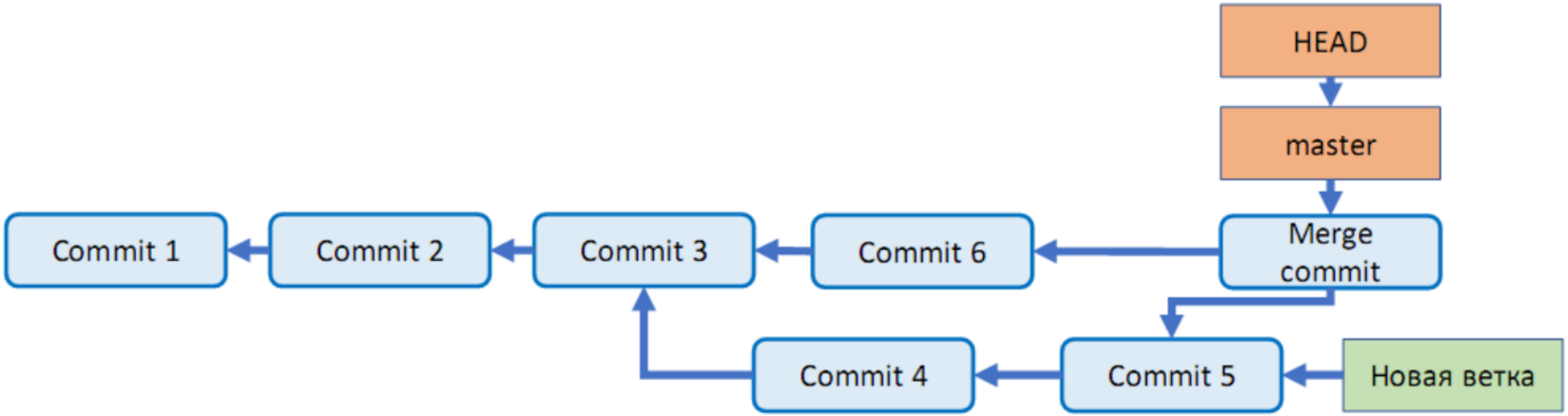
Переключение веток



Существует указатель HEAD, который указывает на коммит, чье состояние сейчас развернуто в рабочей области. Те файлы, которые есть в папочке, образовались с помощью последовательности коммитов, на которую указывает HEAD.

Если переключить указатель HEAD с помощью команды git checkout на master или «Новая ветка», то мы получим разные цепочки изменений, а значит и разные состояния рабочих областей.

Слияние веток



Когда работа над отдельной задачей завершается, то необходимо внести изменения в основную ветку. Эта процедура называется «слияние» и выполняется командой **git merge**

В результате появляется новый коммит, который ссылается на два предыдущих коммита одновременно.

Облачный репозиторий

До текущего момента мы работали на локальной машине, в нашей папочке.

Но как же делиться кодом с коллегами, как совместно решать множество задач?

Для этого существуют удаленные репозитории кода, которые хостятся в интернете.

Из самых известных это GitHub, GitLab, GitFlic

Облачный репозиторий

СТИ
НИЯУ МИФИ

Git — это программа, которую нужно установить и подключить к проекту для управления системой контроля версий.

GitHub — это сайт-хранилище для историй версий проектов: вы подключаете Git, регистрируетесь на GitHub, создаёте онлайн-репозиторий и переносите файлы с Git на GitHub.

Git — это самая популярная система контроля версий, а GitHub — онлайн-хранилище кода.

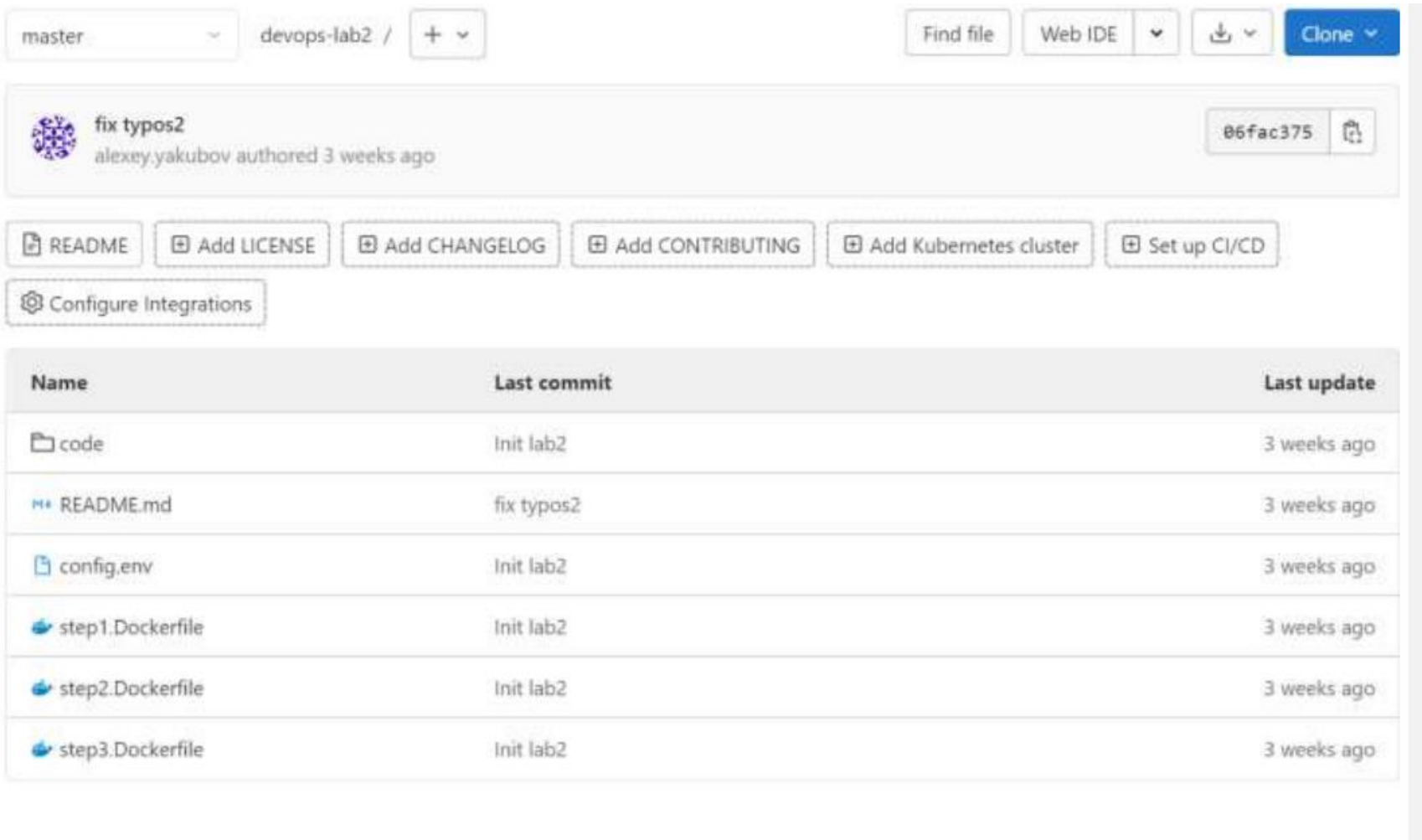
Git и GitHub настроены на взаимодействие и поэтому часто используются как единый механизм работы с проектом.

Общий репозиторий

СТИ
НИЯУ МИФИ

Локальный репозиторий можно загрузить в удаленный, чтобы хранить код в облаке и работать совместно с коллегами.

$ git push



Общий репозиторий

СТИ НИЯУ МИФИ

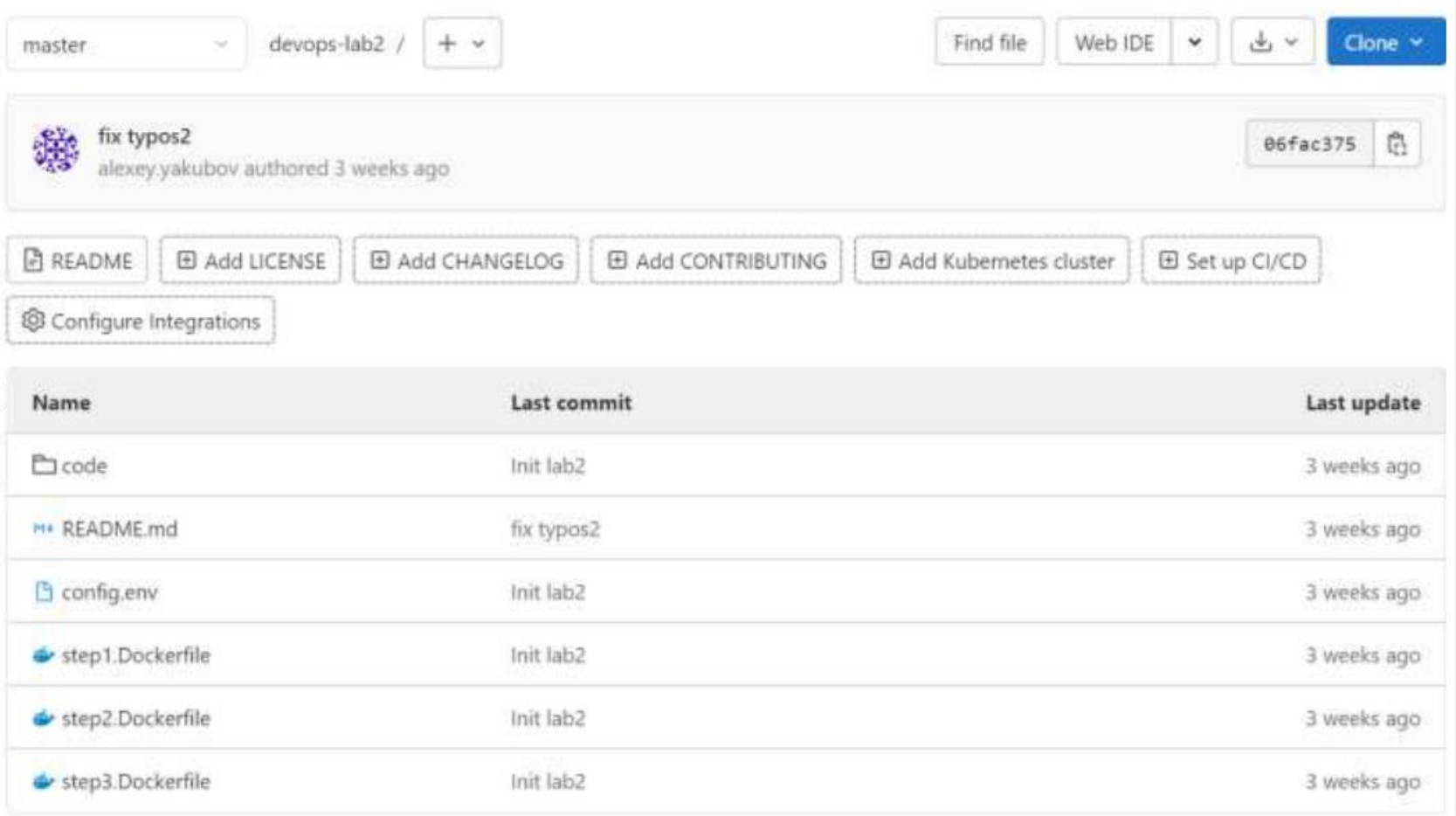
Разумеется из удаленного репозитория

можно скачать все

изменения, операция

называется pull и выполняется с помощью команды

```
$ git pull
```



Общий репозиторий

СТИ
НИЯУ МИФИ

Скопировать репозиторий для внесения изменений в копию можно двумя основными способами:

- клонировать (clone) — то есть просто скопировать на локальный компьютер или сервер;
- форкнуть (от английского fork — развилка) — сделать отдельную копию репозитория (обычно чужого) для продолжения разработки «по другому пути развилки».

Общий репозиторий

СТИ
НИЯУ МИФИ

Если вы форкнули чужой проект, чтобы предложить автору конкретные улучшения, нужно по готовности «запУлить» их в исходный репозиторий, то есть сделать запрос на изменения

pull request

Команды для GitHub CLI начинаются с сокращения gh — ***gh repo clone.***

Ведение проекта

СТИ
НИЯУ МИФИ

Основное описание вашего проекта задаётся в файле Readme.md, который можно создать сразу в репозитории или после.

Расширение ***md*** — сокращение от названия популярного языка упрощённой (проще чем html) разметки текста — Markdown.

Содержимое файла Readme отображается на главной странице репозитория и отвечает на вопрос, что это за проект, чем он может быть полезен другим разработчикам и как им пользоваться.

Ведение проекта

СТИ
НИЯУ МИФИ

Чтобы оформить Readme стильно, можно почитать руководство по markdown-разметке.

В оформлении ***md*** можно использовать — заголовки разных уровней, выделение жирным/курсивом, изображения, эмодзи, ссылки, диаграммы, графики и так далее. Файл Readme может быть довольно длинным, но всё же для оформления большой документации GitHub рекомендует создать «Вики» (wiki)

Ведение проекта

СТИ
НИЯУ МИФИ

На GitHub можно захостить сайт с помощью функции GitHub Pages.

Это просто:

- Зайдите в настройки репозитория.

- В блоке Code and automation выберите Pages.

- Выберите источник (Deploy from a branch, затем нужную ветку).

- Кликните на Save. Обновите страницу, и вверху страницы появится ссылка на ваш новый сайт.

Сервисом Git пользуются все: это один из важных общих навыков вне зависимости от выбранного вами языка программирования и направления разработки.

И, как уроки ОБЖ, тот же ***git clone*** когда-нибудь вас спасёт.

Поэтому важно начинать пользоваться Git как можно раньше — хотя бы даже для бэкапов учебного кода, и уже скоро это станет полезной привычкой.

Merge request

СТИ НИЯУ МИФИ

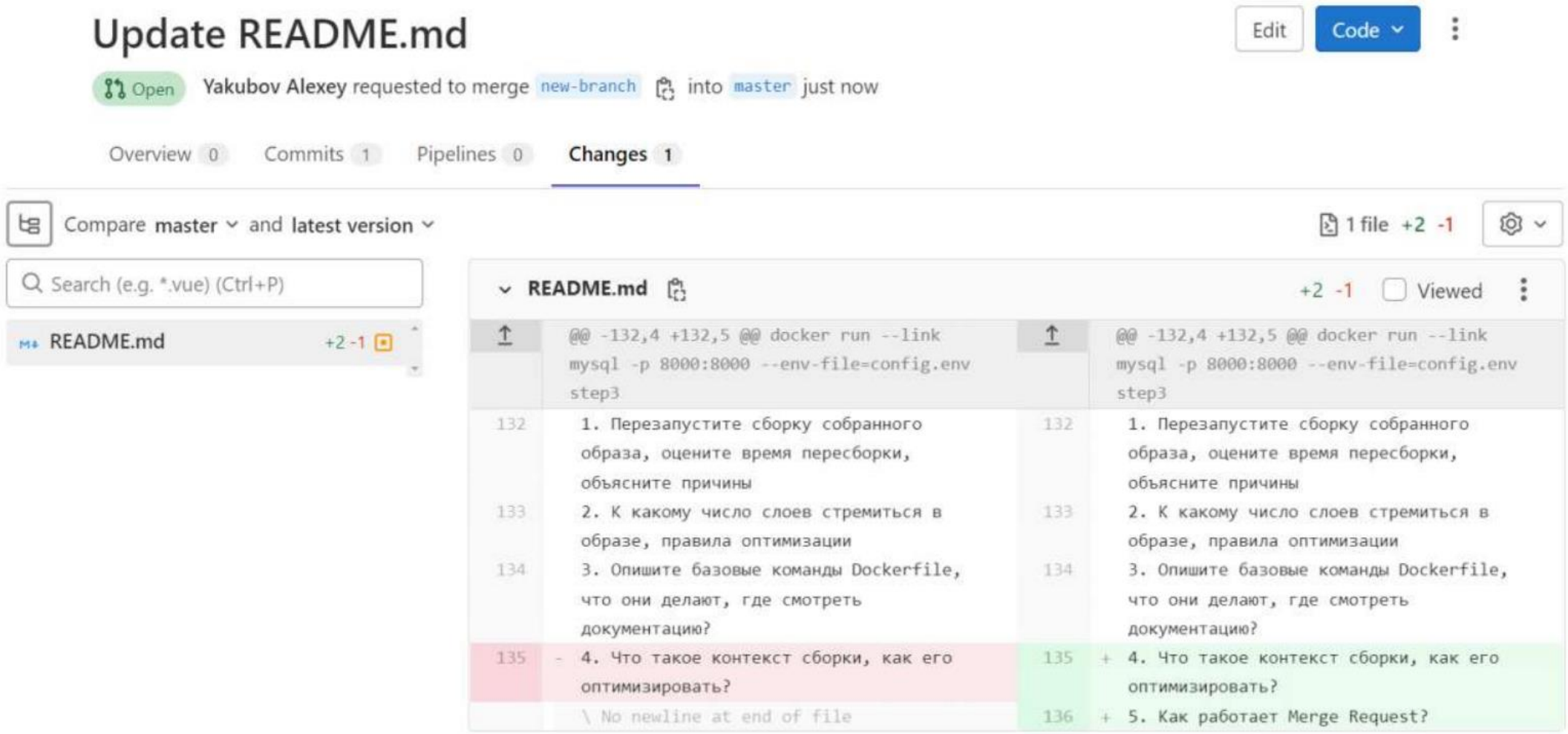
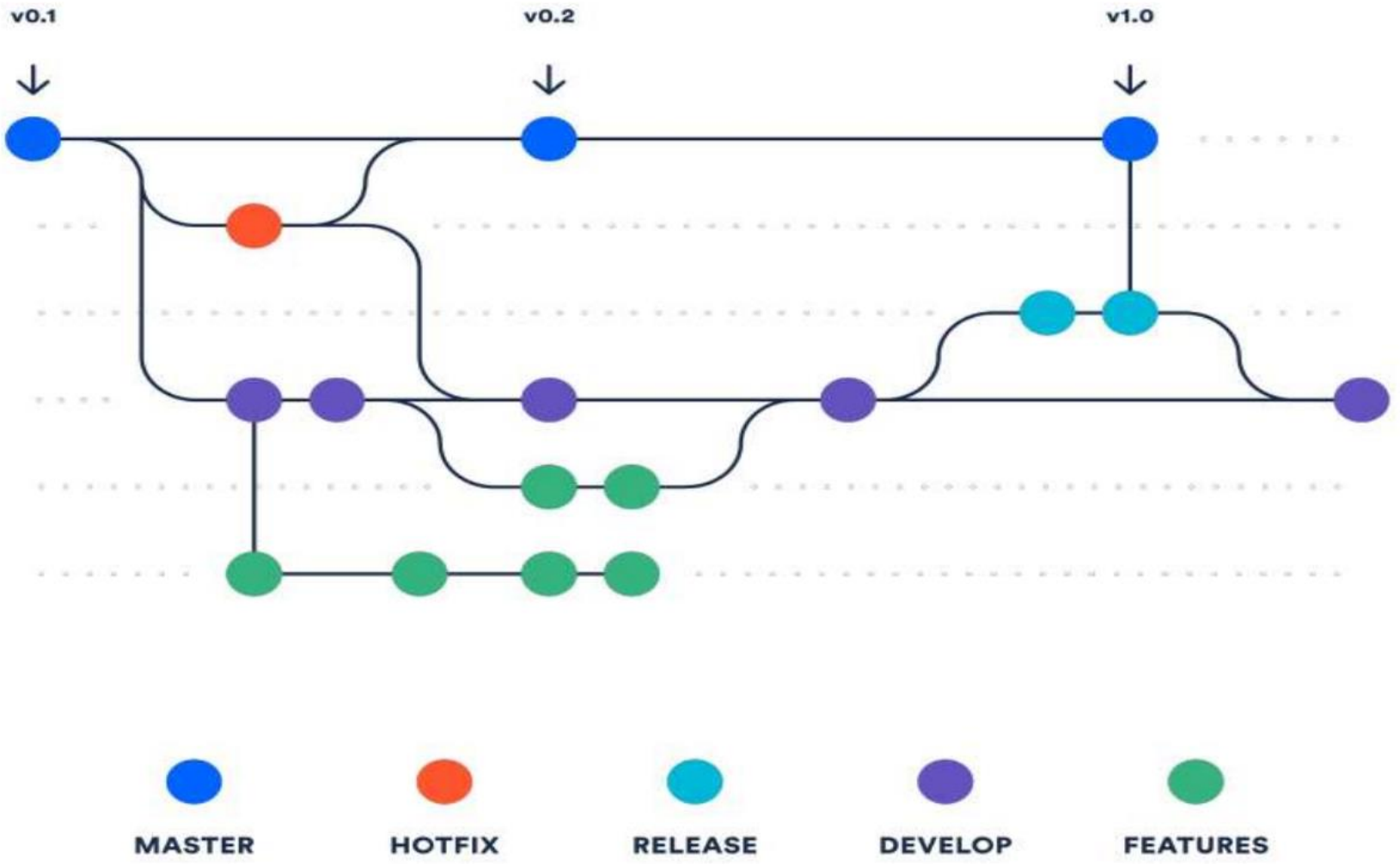
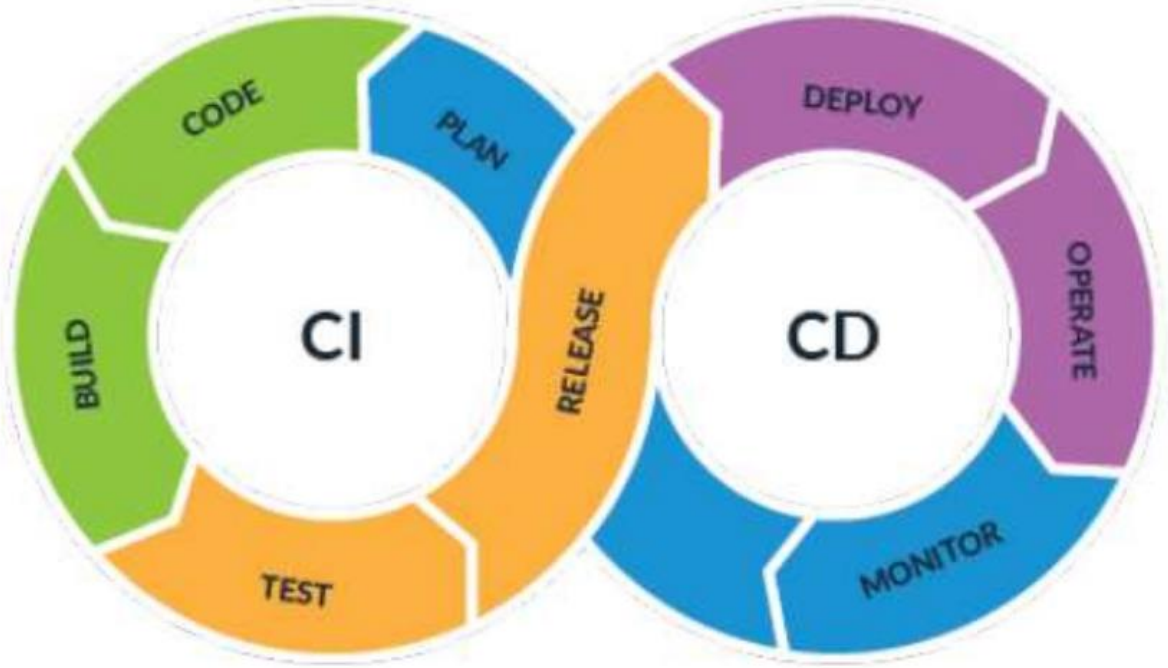


Иллюстрация контроля версий



CI/CD

Удаленный репозиторий кода позволяет настраивать автоматические действия, которые выполняются при появлении нового кода.
Это позволяет автоматически проверять новый код
и даже автоматически доставлять новый код пользователям



GIT

СТИ
НИЯУ МИФИ

Git — это консольная утилита, которая отслеживает и фиксирует изменения в файлах. Это система контроля версий

Git является распределенным, то есть не зависит от одного центрального сервера, на котором хранятся файлы. Вместо этого он работает полностью локально, сохраняя данные в папках на жестком диске, которые называются репозиторием

Так же репозитории можно хранить и в интернете. Обычно для этого используют такие сервисы как GitHub, GitLab, Bitbucket

GitHub - это веб сервис, которые позволяет хранить и управлять репозиториями Git.

GitHub

СТИ
НИЯУ МИФИ

GitHub — сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше. Обычно он используется вместе с Git и даёт разработчикам возможность сохранять их код онлайн, а затем взаимодействовать с другими разработчиками в разных проектах.

Также GitHub может похвастаться контролем доступа, багтрекингом, управлением задачами и вики для каждого проекта. Цель GitHub — содействовать взаимодействию разработчиков.

https://github.com/

GitHub – это …

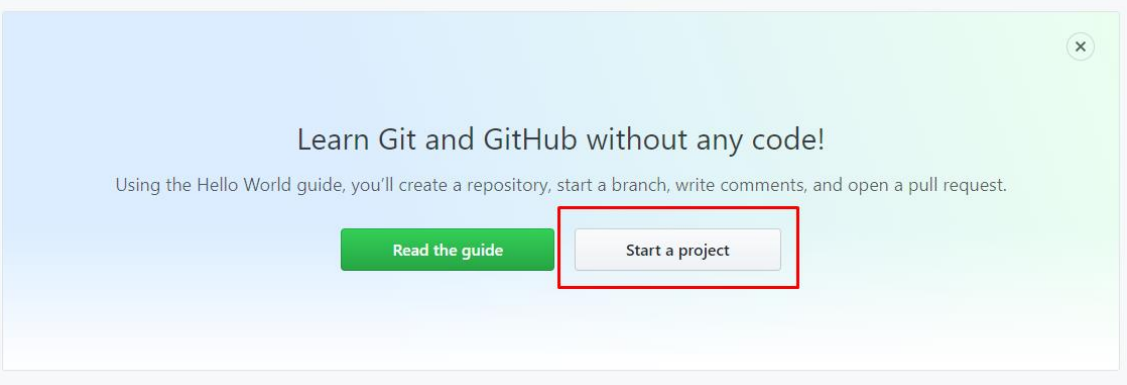
СТИ
НИЯУ МИФИ

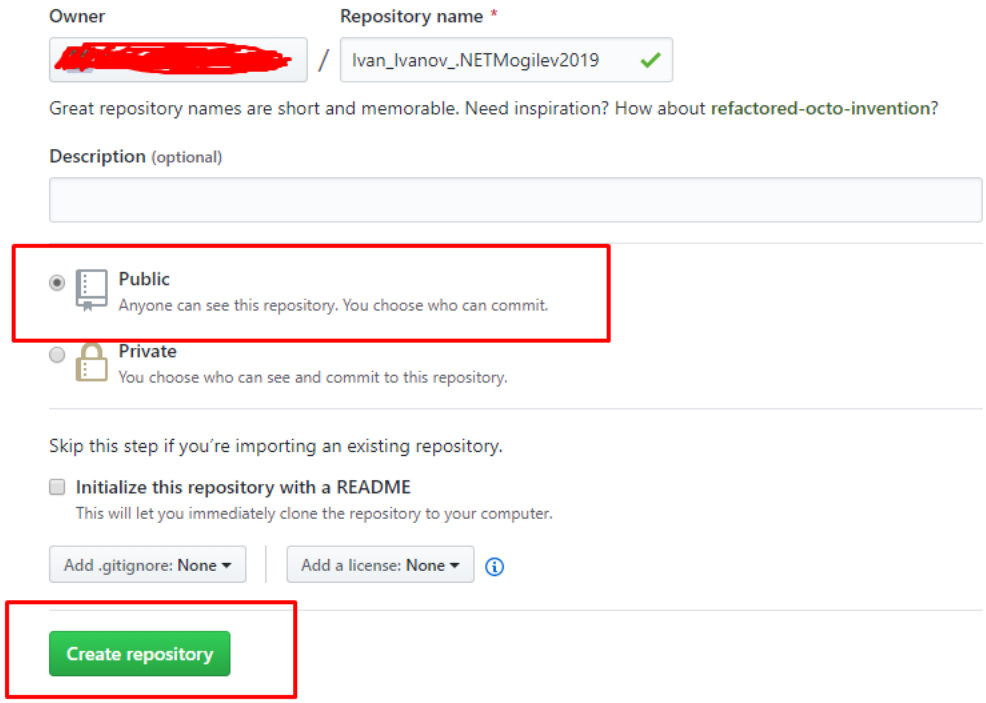
GitHub – это крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.
Помимо участия в определённом проекте, GitHub позволяет пользователям общаться с единомышленниками.

Регистрация

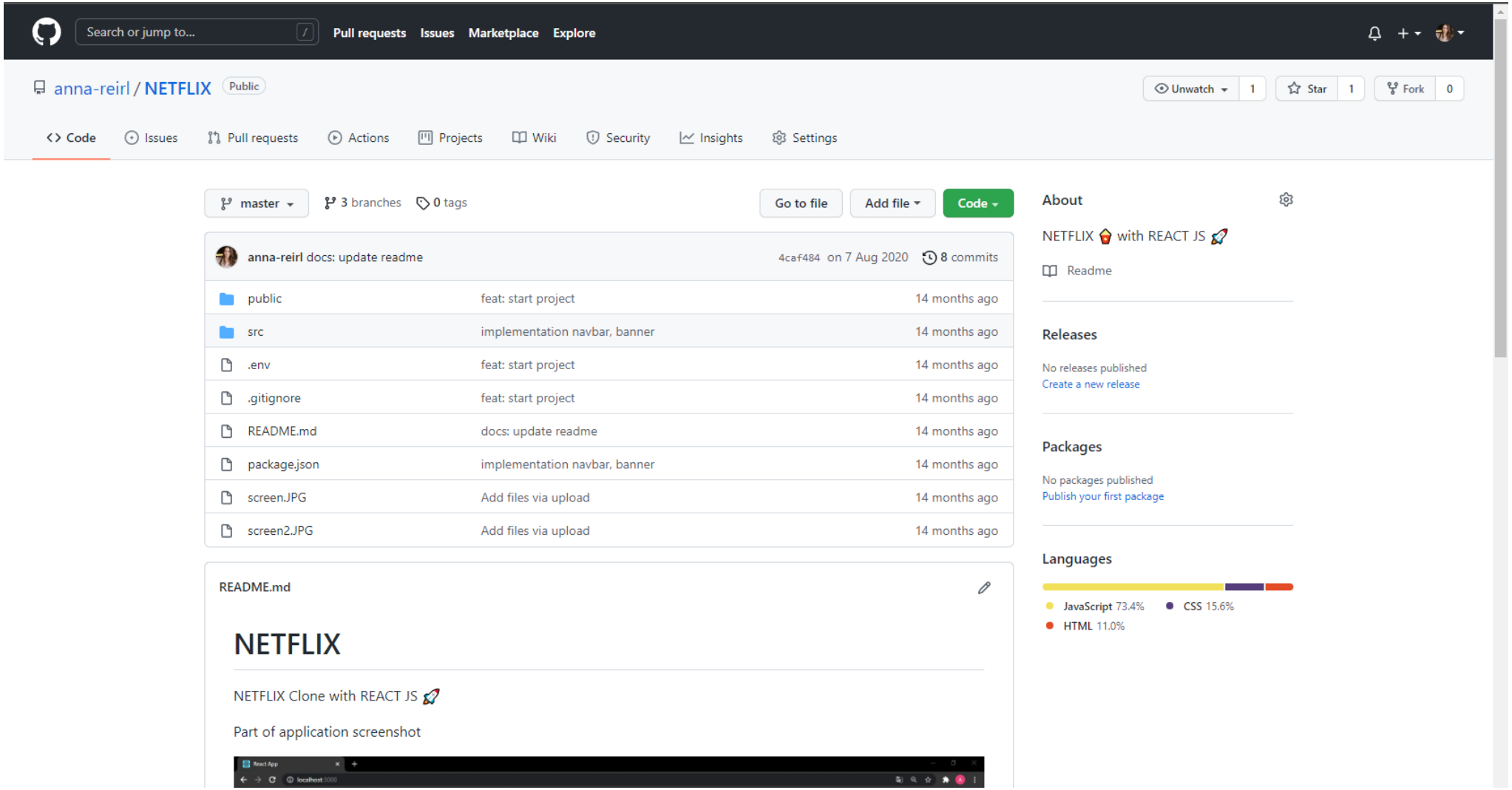
Все задания, которые Вы будите выполнять, должны храниться в репозитории. Это обеспечит доступ к Вашим работам людям, которые будут их проверять. Для этого Вам необходимо зарегистрироваться на портале https://github.com/ и создать Public-репозиторий.





GitHub

СТИ
НИЯУ МИФИ



GitHub

СТИ
НИЯУ МИФИ

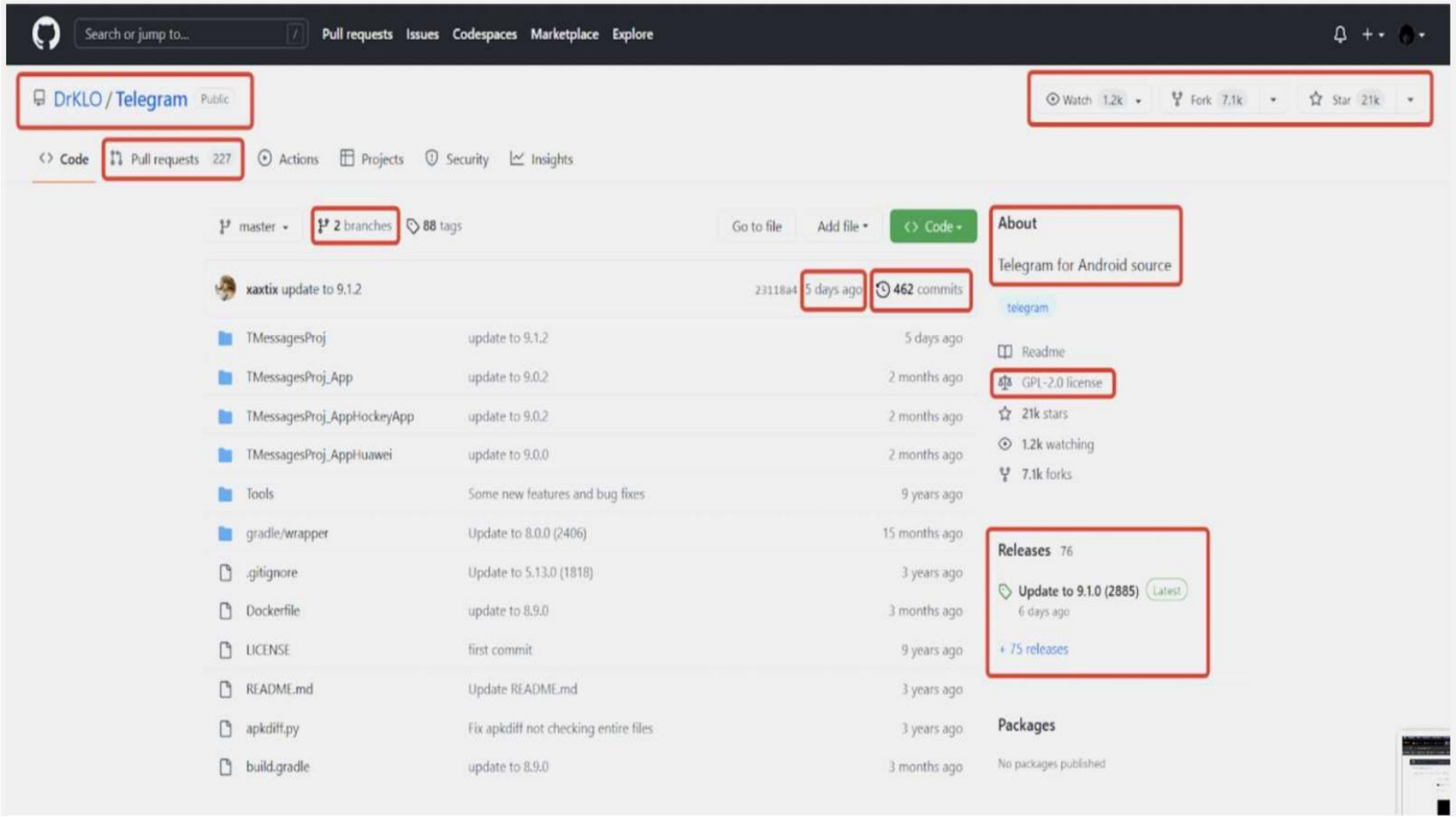
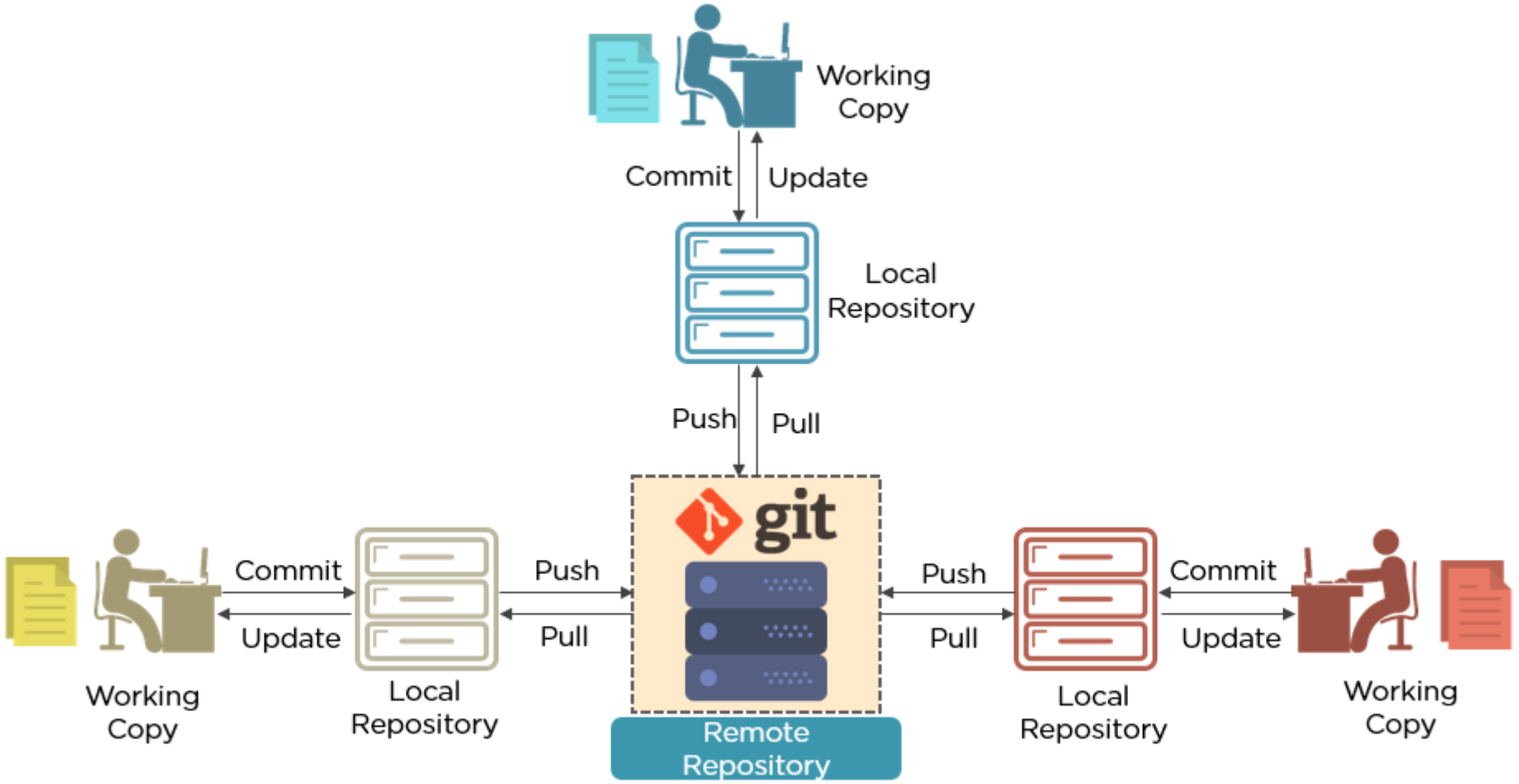
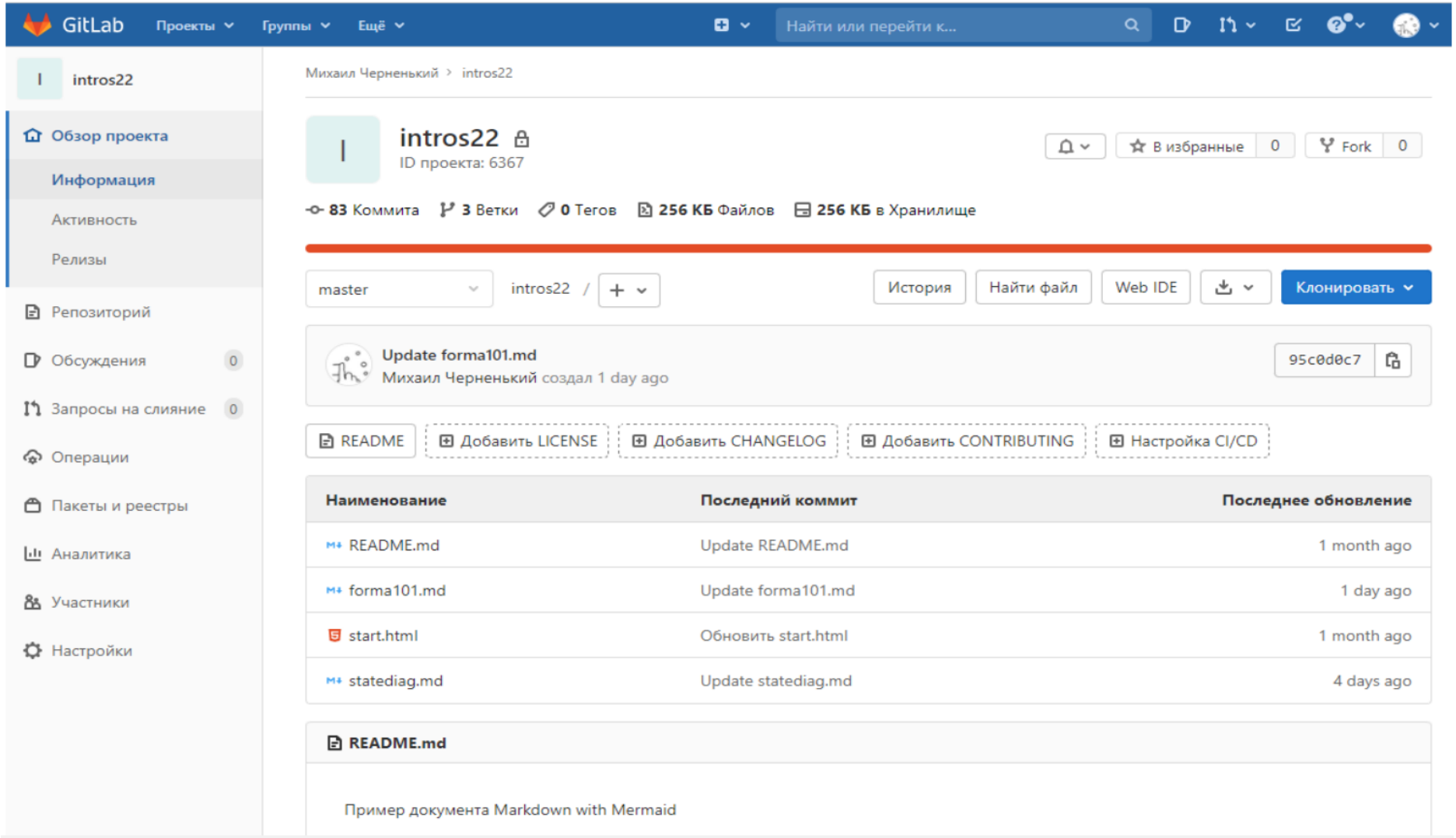


Схема работы с репозиторием Git



Gitlab

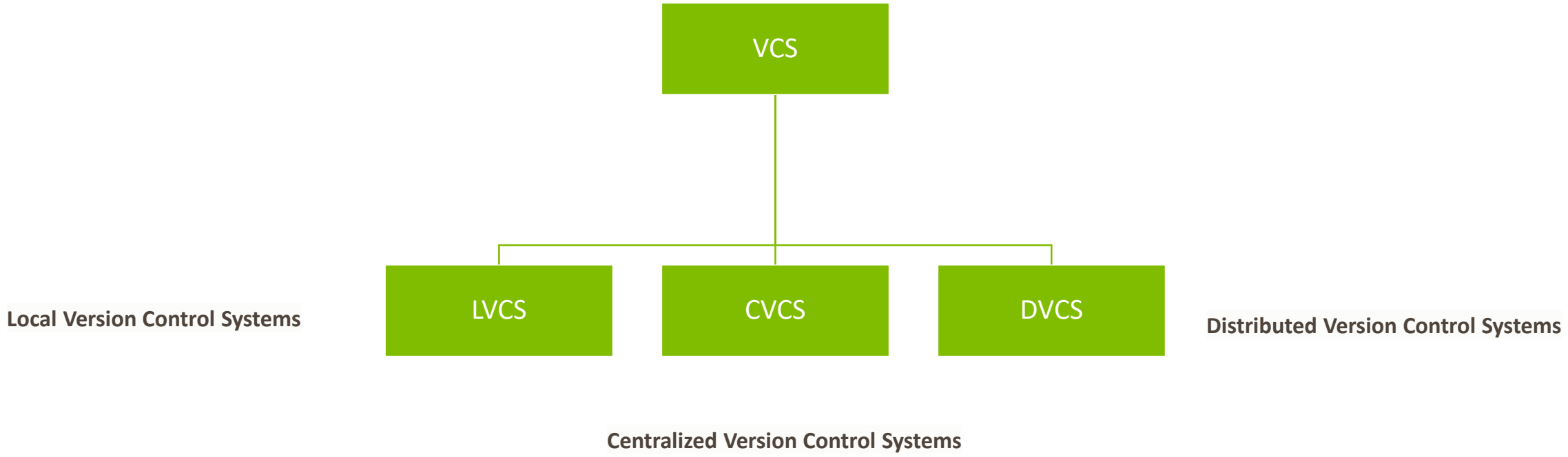
СТИ
НИЯУ МИФИ



Система контроля (управления) версий

СТИ НИЯУ МИФИ

Система контроля версий (*Version Control System, VCS*) — программное обеспечение для облегчения работы с изменяющейся информацией. 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, и многое другое.



Локальная система управления версиями

СТИ НИЯУ МИФИ

Локальные СКВ обычно хранят на компьютере список изменений, внесенных в файлы. Основываясь на этих данных, система контроля версий воссоздает нужную версию файла (актуальную на определенный момент времени).

Достоинства:

— возможность восстановления данных из определенной версии (точно определяется по времени записи);

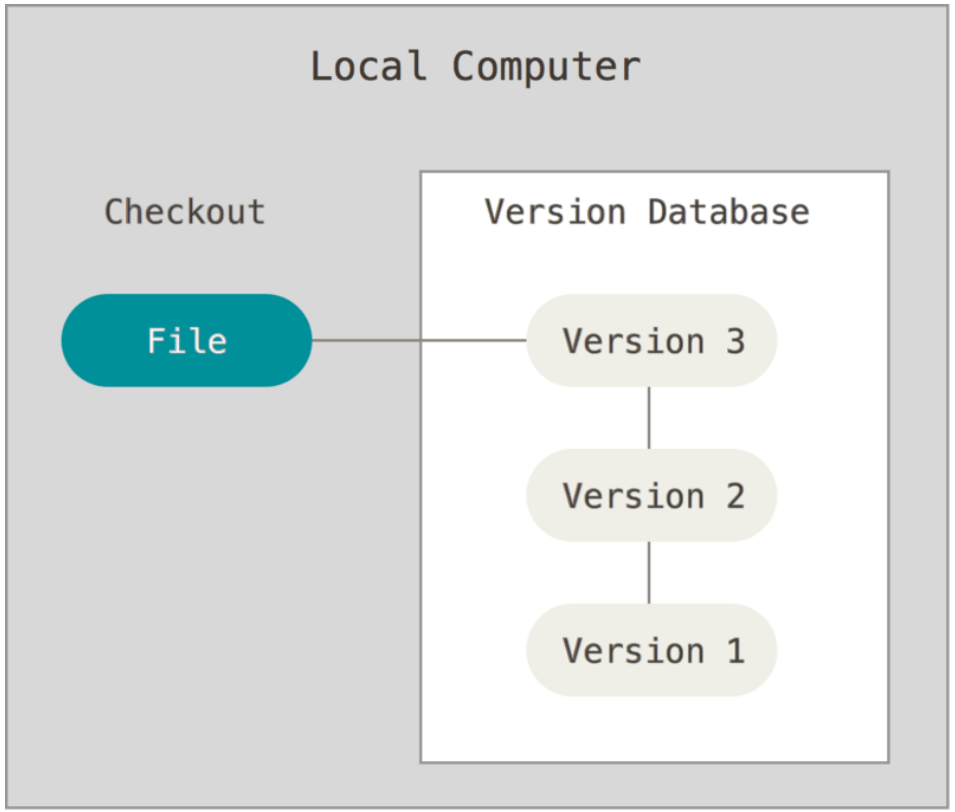
— высокая скорость выполнения восстановления (база данных четко структурирована, поэтому сложностей при поиске не возникает, сетевая задержка отсутствует, поскольку данные хранятся непосредственно на рабочем компьютере).

Недостатки:

— возможность потери данных вследствие возникновения физических поломок оборудования;

— отсутствие возможности совместной разработки.

RCS (Revision Control System, Система контроля ревизий), которая до сих пор устанавливается на многие компьютеры



Централизованная система управления версиями

Централизованные системы контроля версий предполагают сохранение версий проектов на общий сервер, с которого потом получают нужные версии клиенты.

Достоинства:

— возможность восстановления данных из определенной версии (точно определяется по времени записи);

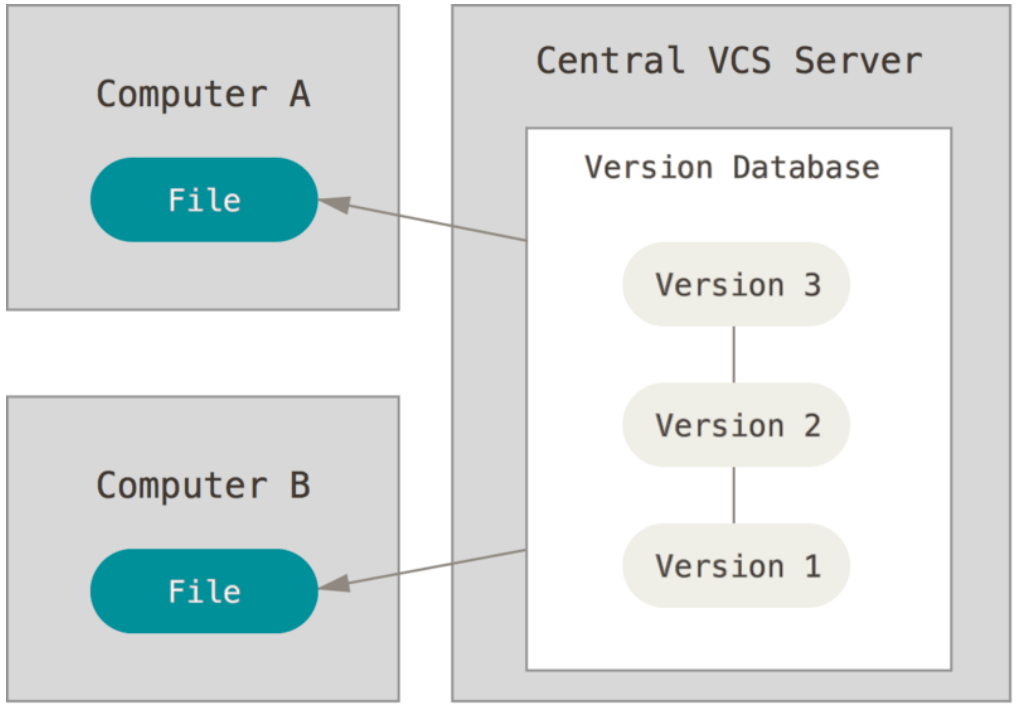
— возможность ведения командной разработки проекта;

Недостатки:

— отсутствие доступа к данным при сбое работы сервера;

— довольно низкая скорость работы (из-за возникновения сетевых задержек).

CVS, Subversion(SVN) и Perforce



Децентрализованная система управления версиями

В децентрализованных системах контроля версий при каждом копировании удалённого репозитория (расположенного на сервере) происходит полное копирование данных в локальный репозиторий (установленный на рабочем компьютере). Каждая копия содержит все данные, хранящиеся в удалённом репозитории. В случае, возникновения технической неисправности на стороне сервера, удаленный репозиторий можно перезаписать с любой сохраненной копии.

Достоинства:

— возможность восстановления данных из определенной версии (точно определяется по времени записи);

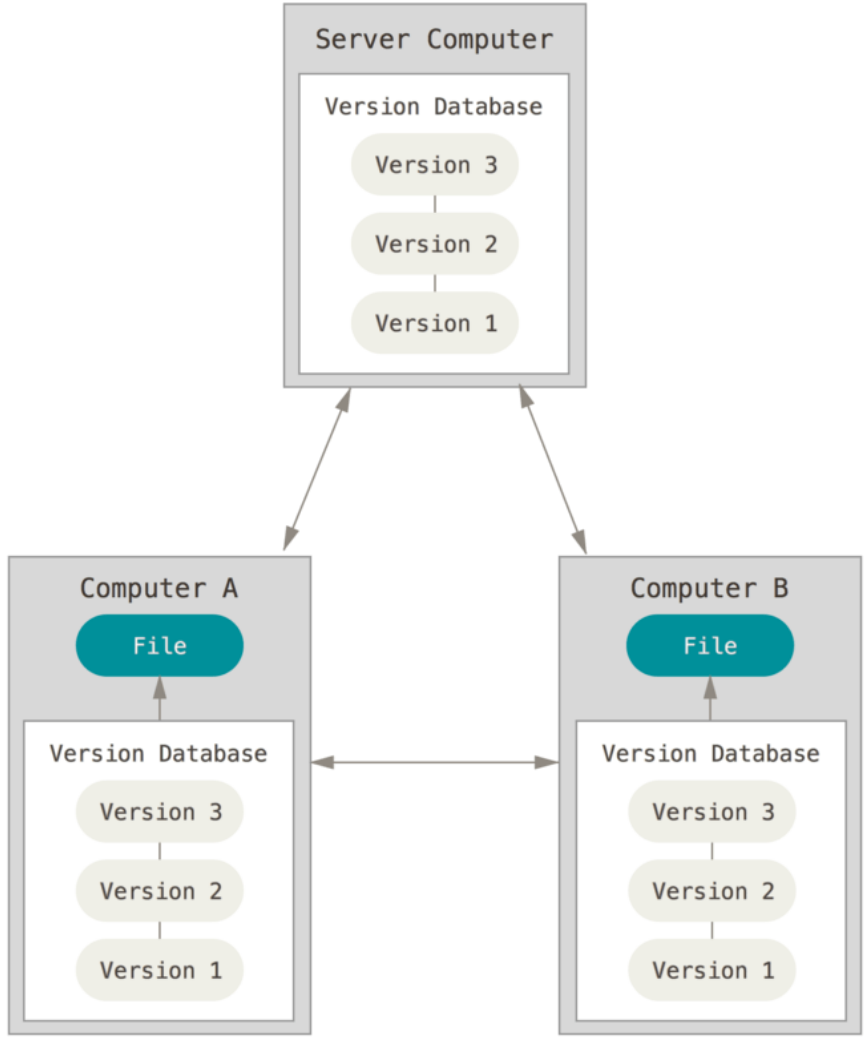
— возможность ведения командной разработки проекта;

— при сбое работы сервера система сохраняет данные в локальном репозитории, что позволяет эффективно вести процесс разработки, а после восстановления работы сервера, передать все изменения в удаленный репозиторий;

— при физической поломке сервера данные можно легко перенести в новый удалённый репозиторий с любого локального репозитория;

— высокая скорость работы (в ходе работы данные записываются и получаются из локального репозитория, поэтому сетевые задержки отсутствуют).

GIT



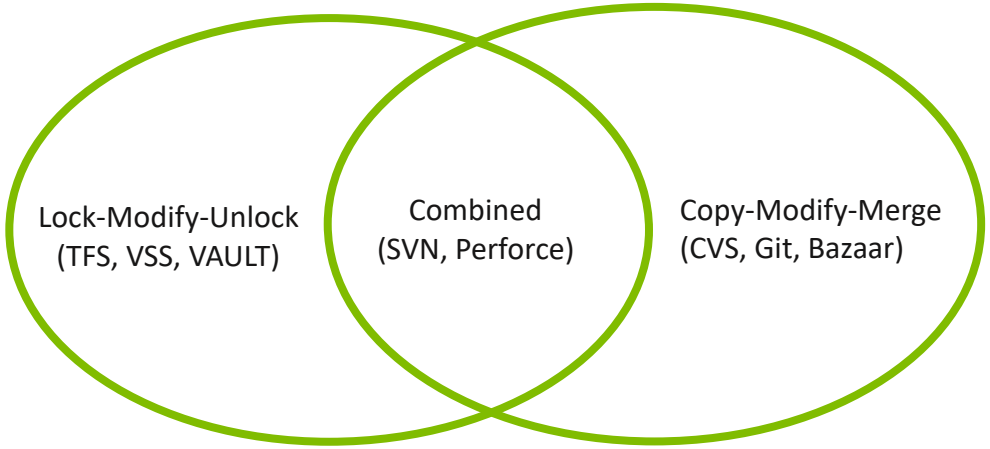
Классификация VCS по способу сохранения данных

СТИ НИЯУ МИФИ

Две модели сохранения данных

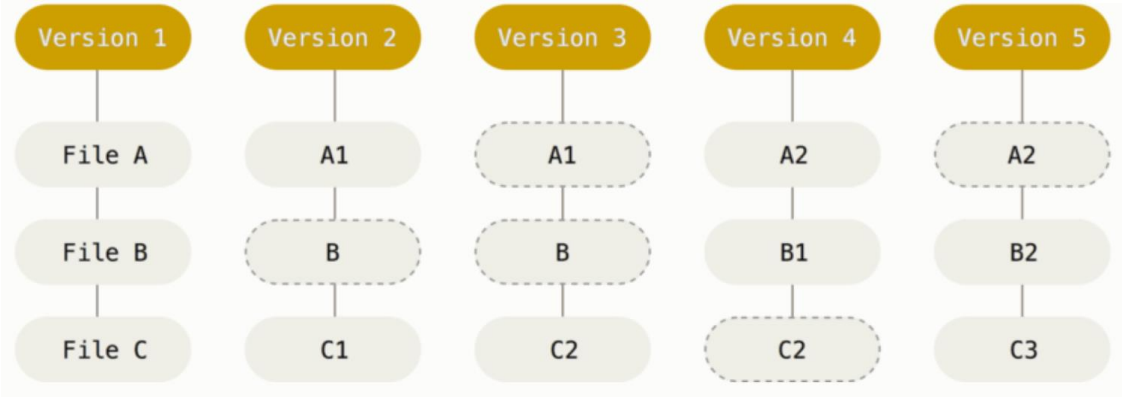
Блокировка — изменение — разблокировка.
Согласно этой модели, когда кто-либо начинает работу с файлом, этот файл блокируется, и все остальные пользователи теряют возможность его редактирования.
Очевидным недостатком такой модели является то, что файлы могут оказаться надолго заблокированными, что приводит к простоям в разработке проекта.

Копирование — изменение — слияние.
В данной модели каждый разработчик свободно редактирует свою локальную копию файлов, после чего выполняется слияние изменений.
Недостаток этой модели в том, что может возникать необходимость разрешения конфликтов между изменениями файла.



Commit - точка сохранения вашего проекта

Подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.
Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранен.
Git представляет свои данные как, скажем, поток снимков.



Три состояния, в которых могут находиться ваши файлы при работе с Git

- **изменен** (modified) - файлы, которые изменились, но еще не были зафиксированы.
- **индексирован** (staged) - изменённые файлы в его текущей версии, отмеченные для включения в следующий коммит.
- **зафиксирован** (committed) - файлы уже сохраненные в вашей локальной базе

Базовый подход в работе с Git выглядит так:

1. Изменяете файлы вашей рабочей копии.
2. Выборочно добавляете в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки **только** этих изменений в индекс. (git add)
3. Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в ваш каталог Git. (git commit)

Ветка в GIT

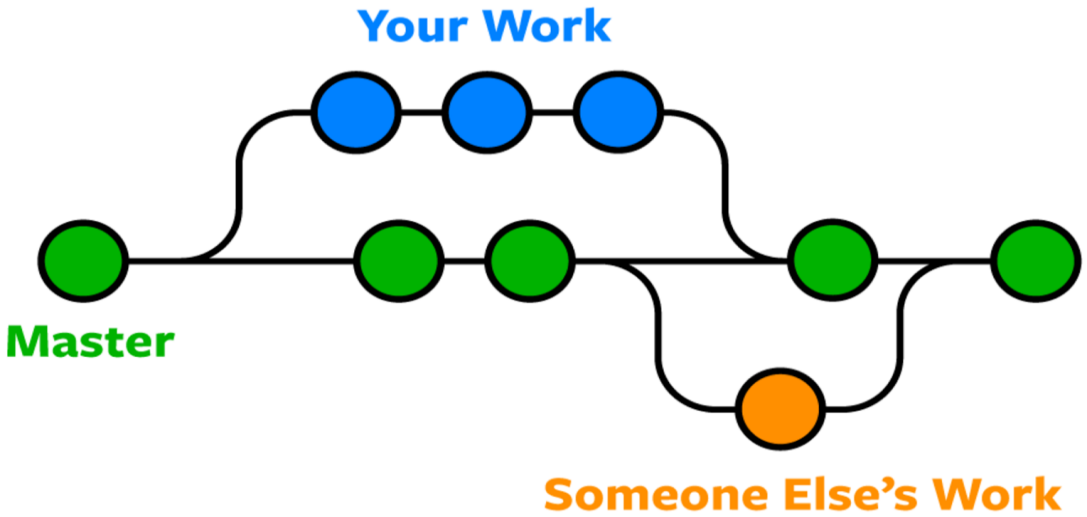
Ветка в Git — это простой перемещаемый указатель на один из коммитов.

По умолчанию, имя основной ветки в Git — **master**. Как только вы начнёте создавать коммиты, ветка **master** будет всегда указывать на последний коммит. Каждый раз при создании коммита указатель ветки **master** будет передвигаться на следующий коммит автоматически.

Что происходит при создании ветки? Всего лишь создаётся новый указатель для дальнейшего перемещения.

Что такое мердж или слияние веток

Это перенос кода из одной ветки в другую



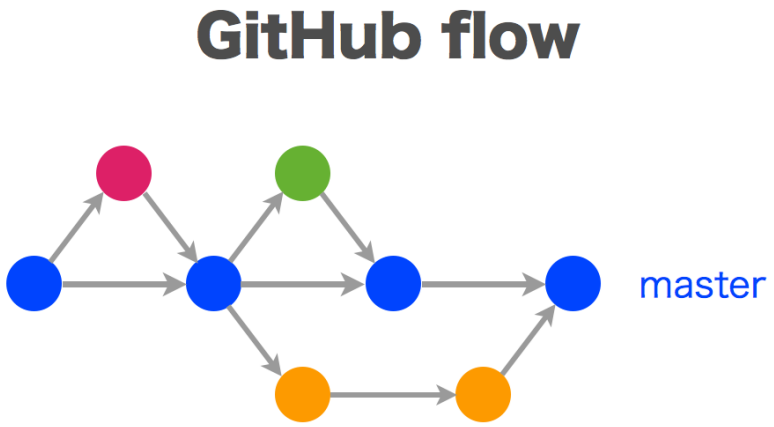
Бранч-стратегии при разработке в Git

СТИ НИЯУ МИФИ

GitHub Flow
Стратегией пользуются в команде сервиса GitHub

Основные правила:

- в коде в мастер-ветке не допускаются ошибки, и он должен быть готов к развертыванию в любой момент;
- чтобы начать разрабатывать новую функцию, необходимо создать feature-ветку в master-ветке и дать ей очевидное для всех имя. Когда работа будет готова, ее нужно смерджить в master-ветку через pull request;
- после мерджа изменений их нужно сразу же развернуть на сервере.



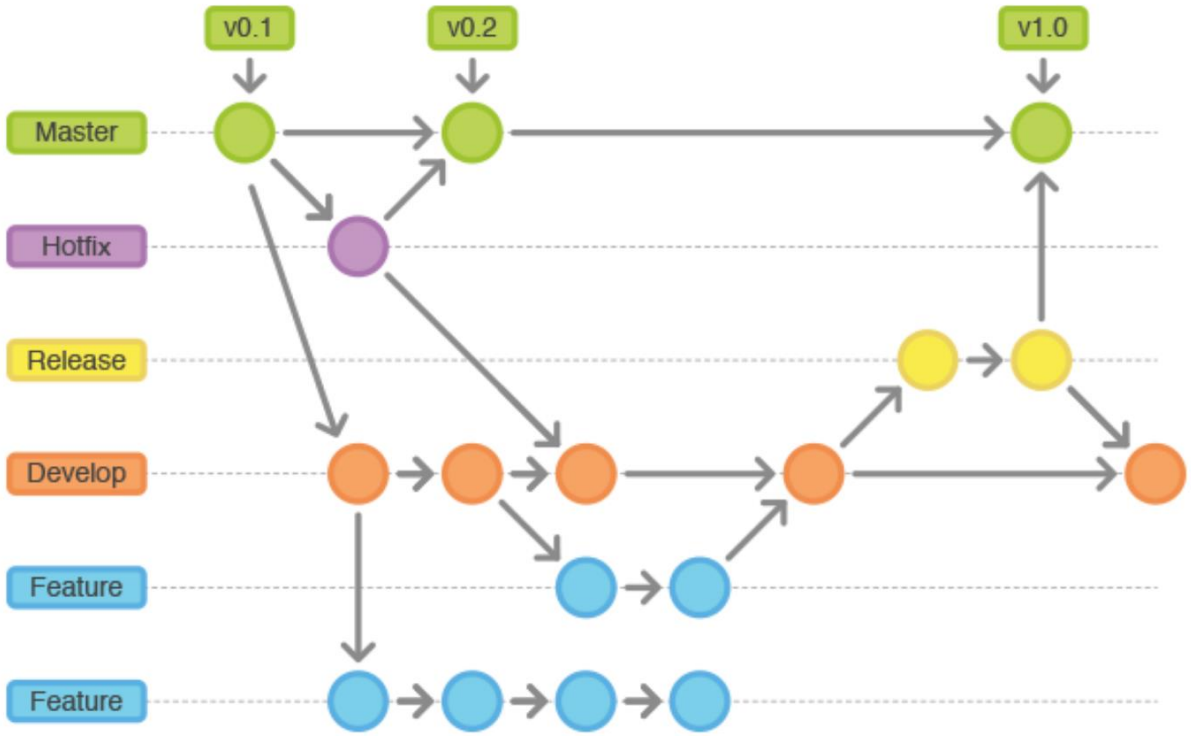
Этот подход обычно используют для продуктов с одной версией, которая обновляется не очень часто. Например, веб-сайтов.

Бранч-стратегии при разработке в Git

СТИ НИЯУ МИФИ

GitFlow
Этот подход считается одним из оптимальных для проектов, где постоянно разрабатывается несколько версий для разных платформ.

Есть два типа постоянных веток: **master**-ветка, чтобы понимать, как выглядит последняя актуальная версия, и **development**-ветка, где ведется разработка. От нее идут три вида временных веток.



- **Feature**, для добавления новых возможностей. После завершения работы нужно создать pull request в development-ветку.
- **Release**, для работы над новыми версиями. Важно добавить в название номер версии, это поможет не запутаться и отследить изменения.
- **Hotfix**, для быстрого исправления багов.

Бранч-стратегии при разработке в Git

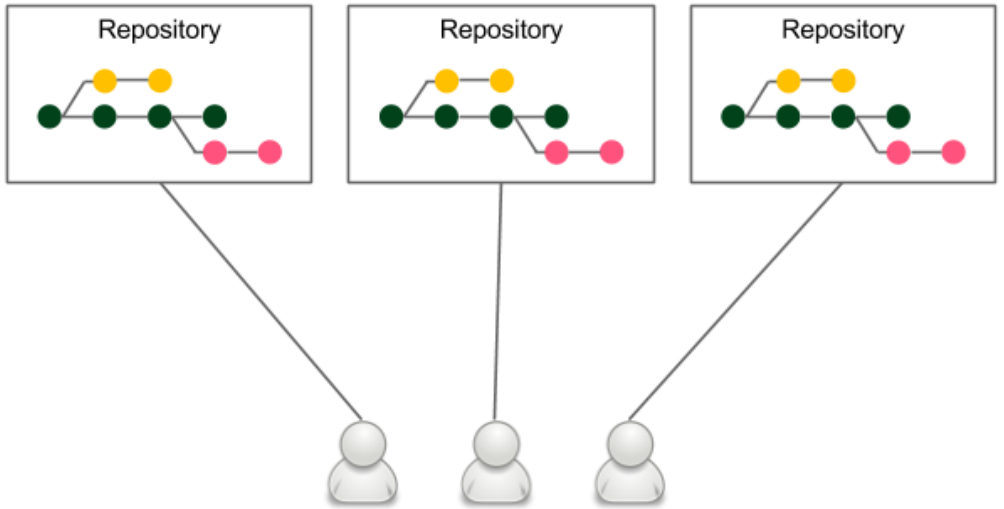
СТИ НИЯУ МИФИ

Forking Workflow
Стратегией пользуются в команде Linux

В рамках Forking Workflow стратегии разработка ведется так, что есть два репозитория:

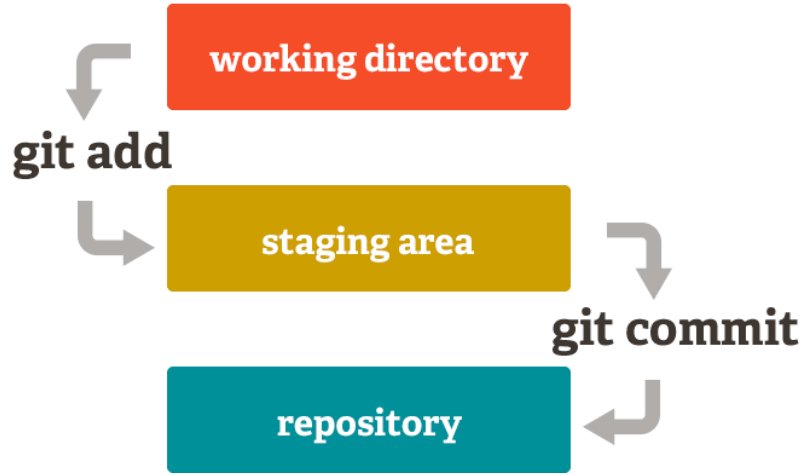
1. **Оригинальный репозиторий**, в который будут смердживаться все изменения.
2. **Форк репозиторий** (это копия оригинального репозитория во владении другого разработчика, который хочет внести изменения в оригинальный).

Подход очень близок к идеологии open source, его цель — использовать все преимущества open-source-сообщества в рамках проекта. При этом большая часть рабочего процесса в части ветвления копирует GitFlow. Feature-ветки здесь будут мерджиться с локальными репозиториями разработчиков. Таким образом, разработка становится гибкой даже для очень больших команд с подрядчиками.



Команды GIT

Staging area - d отличие от других систем, в Git есть нечто, называемое "промежуточной областью" или "индексом". Это промежуточная область, где коммиты могут быть отформатированы и просмотрены перед завершением коммита.



Local repo - Локальный репозиторий. Это репозиторий, который хранится на нашей машине, в рабочей папке проекта. Это та самая скрытая папка .git.

(Remote repo) Удаленный репозиторий – это репозиторий, размещенный в локальной или интернет сети. Удаленный репозиторий используется для того, чтобы делиться и обмениваться кодом между разработчиками в рамках сети. Его также можно использовать, если вы разрабатываете проект на нескольких устройствах

Команды GIT

СТИ
НИЯУ МИФИ

git init - Создать пустой репозиторий Git или вновь инициализировать существующий можно параметром init. При инициализации он создаст скрытую папку. В ней содержатся все объекты и ссылки, которые Git использует и создаёт в истории работы над проектом.

git add . – добавить все папки в область

git status - Просмотреть статус нужного репозитория можно по ключевому слову status: его действие распространяется на подготовленные, неподготовленные и неотслеживаемые файлы.

git commit -m "Your short summary about the commit” - При создании коммита в репозитории можно добавить однострочное сообщение с помощью параметра commit с флагом -m. Само сообщение вводится непосредственно после флага, в кавычках.

git branch new_branch_name - Создать новую ветку можно с помощью параметра branch, указав имя ветки.

git checkout -b new_branch_name - Для перехода нужно добавить флаг -b и параметр checkout.

Команды GIT

СТИ
НИЯУ МИФИ

СОЗДАНИЕ РЕПОЗИТОРИЯ

С нуля -- Создать новый локальный репозиторий
$ git init [project name]

Скачать с существующего репозитория
$ git clone my_url

ОТСЛЕЖИВАНИЕ РЕПОЗИТОРИЯ

Просмотреть список новых или измененных файлов, которые еще не закомитены
$ git status

Показать изменения в файлах, которые еще не поставлены
$ git diff

Показать изменения в индексированных файлах
$ git diff —cached

Показать все индексированные и не индексированные изменения файлов
$ git diff HEAD

Показать различия между двумя комитами
$ git diff commitl commit2

Показать дату изменения и автора для файла
$ git blame [file]

Показать изменения для определенного комита и/или файла
$ git show [commit] : [file]

Показать полную историю изменений
$ git log

Показать историю изменений для файла/папки включая различия(diffs)
$ git log -p [file/directory]

РАБОТА С ВЕТКАМИ

Показать все локальные ветки
$ git branch

Показать все локальные и удаленные ветки
$ git branch –av

Переключится к ветке my_branch и обновить рабочую директорию
$ git checkout my_branch

Создание новой ветки с именем new_branch
$ git branch new_branch

Удалить ветку с именем my_branch
$ git branch -d my_branch

Объединить branch_a в branch_b
$ git checkout branch_b
$ git merge branch_a

Добавить Tag к текущему комиту
$ git tag my_tag

ИЗМЕНЕНИЯ

Индексировать файл готовый к комиту
$ git add [file]

Индексировать все файлы готовые к комиту
$ git add .

Зафиксировать индексированные файлы с комментарием в историю
$ git commit -m "commit message"

Зафиксировать все отслеживаемые файлы с комментарием
$git commit -am "commit message"

Неиндексированные файлы, получить изменения файла
$ git reset [file]

Откатить все до последней фиксации
$ git reset —hard

СИНХРОНИЗАЦИЯ

Получить последние изменения с удаленного сервера (без слияния)
$ git fetch

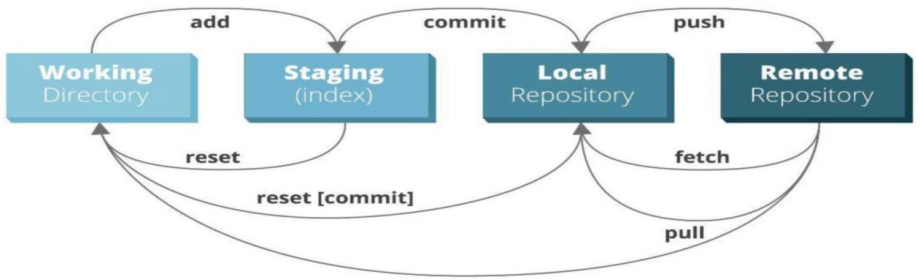
Получить последние изменения с удаленного сервера и слияния
$ git pull

Получить последние изменения с удаленного сервера и перебазировать
$ git pull —rebase

Применить локальные изменения на удаленный сервер
$ git push

ФИНАЛ

В случае сомнений, используйте помощь
$ git command —help



Команды GIT

СТИ
НИЯУ МИФИ

Git — это распределённая система версий

Системы контроля версий бывают локальными, централизованными или распределёнными.

Локальная система хранит файлы на одном устройстве, централизованная использует общий сервер, а распределённая — общее облачное хранилище и локальные устройства участников команды. В локальной системе удобно работать с большими проектами, но сложно взаимодействовать с удалённой командой.

В централизованной системе налажена удалённая работа, но всё привязано к одному серверу. Любой сбой или взлом может повредить файлы проекта.

В распределённой системе налажена удалённая работа. Если с файлами основного репозитория что-то случится — проект легко восстановить из копии любого участника команды.

Git — это программа, которую нужно установить и подключить к проекту для управления системой контроля версий.

GitHub — это сайт-хранилище для историй версий проектов: вы подключаете Git, регистрируетесь на GitHub, создаёте онлайн-репозиторий и переносите файлы с Git на GitHub.

Git — это самая популярная система контроля версий, а GitHub — онлайн-хранилище кода. Git и GitHub настроены на взаимодействие и поэтому часто используются как единый механизм работы с проектом.

Команды : работа с удаленным репозиторием

СТИ НИЯУ МИФИ

```
# Получение работоспособной копии проекта из удаленного репозитория

git clone https://github.com/skrynko-a/unit_tests_example.git

# Для того, чтобы при клонировании запрашивались ваши учетные данные удаленного репозитория, нужно клонировать проект с указанием вашего логина на удаленном репозитории (например GitLab)

git clone https://<name>@github.com/skrynko-a/unit_tests_example.git

# Запрос изменений с сервера

git pull

git pull origin master   # после origin ветка из которой вам нужно “подлить” изменения

git pull origin dev
```

Команды : создание репозитория

СТИ
НИЯУ МИФИ

```
# У вас на локальном компьютере есть проект, для которого вы хотите создать удаленный репозиторий.
Переходим в директорию с вашим проектом и выполняем

git init

# Для добавления файлов используется команда:

git add file_name

git add -A # Все файлы

# Делаем первый коммит

git commit - m “My first commit”

# Устанавливаем соединение с удаленным репозиторием и выливаем туда наш проект

git branch -M master

git remote add origin https://github.com/ololo/example_unit_tests_py.git

git push -u origin master
```

Команды : работа с ветками

СТИ НИЯУ МИФИ

```
# Создание новой ветки
git branch your_new_branch_name
# Посмотреть какие ветки есть
(активная ветка помечена *)
git branch
# Переключиться между ветками
git checkout branch_name
# Сохранить изменение в ветке локально
git commit -m "Added new tests for authorization"
# Отправить изменения на удаленный репозиторий
git push
```

```
# Если вам необходимо вылить ваши изменения
из ветки branch_name в dev или master:
Git pull origin dev (из ветки в которой
работаете)
git push (отправляем на сервер изменение в
вашей ветке)
git checkout dev
Git pull origin branch_name
Git push
```

Команды : отслеживание изменений

СТИ НИЯУ МИФИ

```
# У каждого коммита есть свой уникальный идентификатор в виде строки цифр и букв. Чтобы просмотреть список всех коммитов и их идентификаторов, можно использовать команду:

git log

# Возвращение файла к предыдущему состоянию

git checkout 09bd8cc1 file.cs

# Если вам нужно  отменить изменения внесенные последним коммитом и вы не успели отправить изменения на сервер

$ git revert HEAD

# Для остальных будем использовать идентификаторы:

$ git revert b10cc123
```