Введение в Git

Системы контроля версий

Система контроля версий является неотъемлемым инструментом в процессе разработки программного обеспечения, позволяя разработчикам эффективно сотрудничать и управлять изменениями в исходном коде. Она обеспечивает возможность отслеживания каждого изменения, внесенного в код, кто и когда его сделал, а также позволяет возвращаться к предыдущим версиям кода в случае необходимости. Это особенно важно в больших проектах, где над кодом работает множество людей, и изменения вносятся постоянно.

Система контроля версий позволяет разработчикам работать над общим проектом одновременно, не мешая друг другу и предоставляя инструменты для решения конфликтов изменений. Разработчики могут обмениваться обновлениями кода, объединяя свои усилия для создания конечного продукта, что значительно ускоряет процесс разработки.

Версионирование является еще одной важной функцией, позволяющей сохранять историю изменений кода. Каждое изменение или группа изменений может быть помечена как версия, что упрощает управление версиями продукта и позволяет легко отслеживать, когда и какие изменения были внесены.

Ветвление и слияние - это мощные возможности систем контроля версий, позволяющие разработчикам работать над различными задачами параллельно. Ветвление создает изолированные копии кода (ветки), что дает возможность работать над новыми функциями и исправлениями без риска нарушения стабильности основной кодовой базы. Слияние позволяет объединять изменения из различных веток обратно в основную ветку, обеспечивая интеграцию всех разработанных функций и исправлений.

На рынке существует несколько популярных систем контроля версий, включая **Git**, **Mercurial** и **SVN** (Subversion). Git является одной из самых популярных систем контроля версий сегодня благодаря своей мощности, гибкости и поддержке распределенной работы. Mercurial также предлагает распределенные возможности с упором на простоту и удобство использования. SVN, или Subversion, представляет собой централизованную систему контроля версий, которая была очень популярна до появления распределенных систем, но до сих пор используется в некоторых проектах за счет своей простоты и надежности.

Git

Git является наиболее известной и широко используемой распределенной системой контроля версий в мире разработки программного обеспечения. Ее популярность обусловлена многими факторами, включая ее применение при разработке такого значимого проекта, как ядро Linux. Одной из ключевых особенностей Git является ее **распределенный характер**, который позволяет каждому участнику проекта иметь полную копию всего **репозитория** на своем локальном компьютере. Эта функция обеспечивает ряд преимуществ, включая возможность работы над проектом в отсутствие сетевого соединения и повышенную устойчивость к сбоям и потере данных.

В отличие от централизованных систем контроля версий, где единственная центральная копия репозитория является основной точкой отказа, распределенность Git обеспечивает дополнительную надежность, так как каждый **клон** репозитория может служить полноценным **бэкапом**. Более того, распределенный подход упрощает совместную работу и обмен изменениями между разработчиками. Программисты могут работать независимо друг от друга, создавая новые функции или исправляя ошибки в своих локальных копиях репозитория, а затем синхронизировать свои изменения с удаленным репозиторием и изменениями других участников проекта.

Такая организация работы способствует не только повышению эффективности разработки, но и улучшению качества кода, поскольку каждое изменение может быть тщательно проверено и протестировано в изоляции перед тем, как оно будет интегрировано в основную кодовую базу. Кроме того, Git предоставляет мощные инструменты для управления версиями и ветвления, позволяя разработчикам легко переключаться между разными задачами и экспериментировать с новыми идеями без риска нарушения стабильности основного проекта.

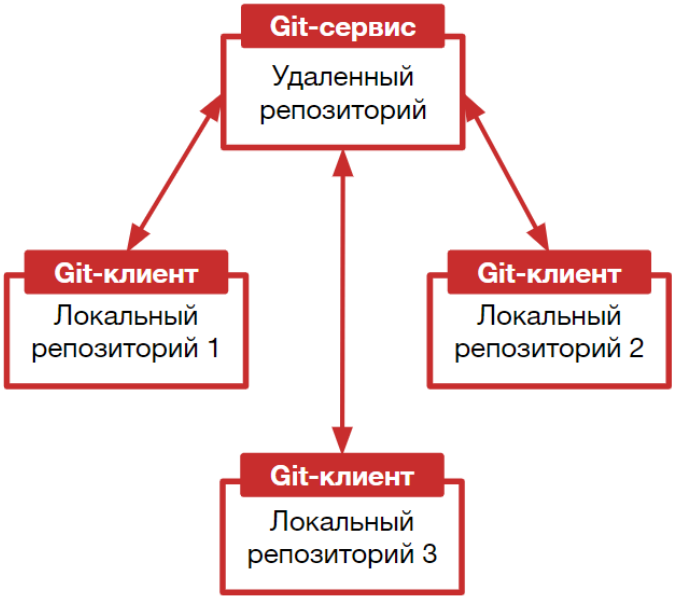
Репозиторий в контексте системы контроля версий, такой как Git, является центральным элементом для хранения исходного кода проекта. Он не только содержит самую актуальную версию всех файлов проекта, но и ведет полную историю их изменений. Эта история позволяет разработчикам просматривать предыдущие состояния кода, понимать, кто и почему внес определенные изменения, а также при необходимости возвращаться к более ранним версиям файлов. Репозитории могут быть как **локальными**, находящимися непосредственно на компьютере разработчика, так и **удаленными**, расположенными на сервере или в облаке.

Git-сервисы играют важную роль в современной разработке программного обеспечения, предоставляя не только хостинг для удаленных репозиториев, но и различные графические интерфейсы и инструменты для упрощения совместной работы над проектами. Самыми популярными сервисами в этой области являются **GitHub**, **GitLab** и **Bitbucket**. Они предлагают широкий спектр функциональности, включая управление задачами, код-ревью, интеграцию с различными инструментами и сервисами, а также возможности для непрерывной интеграции и доставки (CI/CD). Эти платформы

значительно облегчают коллаборацию в командах, позволяя разработчикам эффективно обмениваться кодом и быстро реагировать на изменения в проекте.

Git-клиент является приложением, которое используется для взаимодействия с Git-сервисами и репозиториями. Клиенты могут быть различных типов, включая текстовые интерфейсы, такие как **Bash** или другие командные оболочки, графические пользовательские интерфейсы (**GUI**), облегчающие визуальное взаимодействие с репозиториями, и интегрированные среды разработки (**IDE**), которые предоставляют единые средства для написания кода, его отладки и управления версиями. Выбор конкретного Git-клиента зависит от предпочтений разработчика и задач, которые перед ним стоят, но в любом случае эти инструменты значительно упрощают процесс работы с кодом и его версиями.

В целом, репозитории, Git-сервисы и Git-клиенты вместе образуют экосистему, которая поддерживает современные методологии разработки программного обеспечения, упрощает совместную работу и повышает эффективность процесса создания и поддержки программных продуктов.



Unix-команды для Git Bash

Git Bash предоставляет пользователям Windows среду, имитирующую работу командной строки Unix, что позволяет использовать множество стандартных Unix команд непосредственно в Windows. Это особенно полезно для разработчиков, которым необходимо работать с Git и другими инструментами разработки, традиционно оптимизированными для Unix-подобных систем. Вот описание основных команд, которые используются в Git Bash:

Переход в папку
Команда **cd ПУТЬ** используется для смены текущего каталога на тот, который указан в аргументе **ПУТЬ**. Если команда выполнена успешно, следующий ввод команд будет производиться из нового местоположения.

Просмотр содержимого текущей папки
Команда **ls** выводит список всех файлов и папок в текущем каталоге. Это позволяет быстро оценить содержимое каталога и находить нужные файлы.

Переход на уровень выше
Команда **cd ..** перемещает текущий рабочий каталог на один уровень вверх по иерархии каталогов. Это удобный способ навигации по файловой системе без необходимости вводить полный путь к родительскому каталогу.

Создание файла
Команда **touch НАЗВАНИЕ_ФАЙЛА** используется для создания нового файла с указанным названием. Если файл с таким именем уже существует, команда обновляет время последнего доступа к файлу. Это простой способ быстро создать новый файл без открытия текстового редактора.

Удаление файла
Команда **rm НАЗВАНИЕ_ФАЙЛА** удаляет файл с указанным названием. Эта операция необратима, поэтому следует использовать её с осторожностью, чтобы случайно не удалить важные файлы.

Удаление папки
Команда **rm -rf НАЗВАНИЕ_ПАПКИ** рекурсивно удаляет папку и все ее содержимое, включая подпапки и файлы. Флаг **-r** указывает на рекурсивное удаление, а флаг **-f** — на принудительное исполнение без запросов подтверждения. Как и команда удаления файла, эта команда должна использоваться с большой осторожностью.

Git Bash

Клонирование репозитория через Git — это начальный шаг для работы с проектом, позволяющий создать полную локальную копию удаленного репозитория. Этот процесс начинается с выполнения команды **git clone *URL-репозитория***, которая загружает все файлы проекта, ветки и историю изменений на локальный компьютер. После клонирования, переход в директорию проекта с помощью команды **cd** позволяет начать непосредственную работу с содержимым репозитория.

Осуществив переход, можно использовать команду **git status** для проверки текущего состояния локального репозитория, которая покажет, какие файлы были изменены, добавлены или готовы к коммиту. Одним из ключевых моментов в поддержании чистоты и порядка в репозитории является исключение из него ненужных файлов, таких как

скомпилированные объекты или служебные файлы разработки. Для этого создается файл **.gitignore**, в который вносятся паттерны файлов и директорий, которые Git должен игнорировать.

После создания **.gitignore**, команда **git status** поможет удостовериться, что этот файл теперь учитывается системой контроля версий. Чтобы файл **.gitignore** начал выполнять свою функцию, его необходимо добавить в индекс с помощью команды **git add .gitignore**, а затем зафиксировать изменения командой **git commit -m 'add gitignore'**. Это гарантирует, что правила игнорирования будут применяться к текущему и последующим коммитам.

Для обновления удаленного репозитория изменениями, внесенными локально, используется команда **git push**. Это отправит последние коммиты на сервер, обновляя удаленный репозиторий и синхронизируя его с локальной версией.

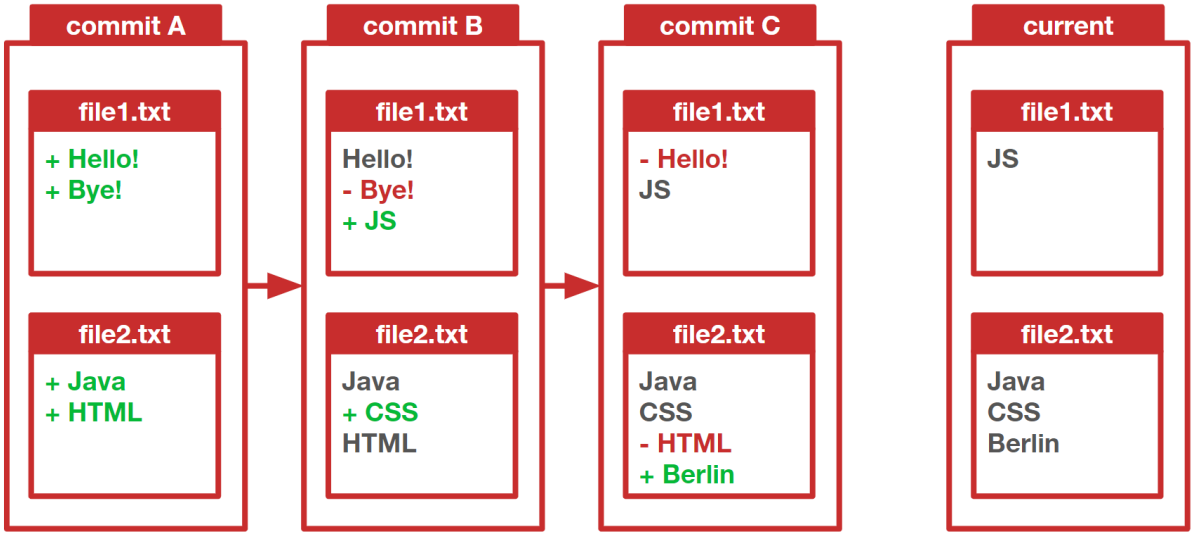
Для просмотра истории коммитов в репозитории служит команда **git log**, которая отображает список коммитов с информацией о авторе, дате и сообщении коммита. Это позволяет отслеживать изменения в проекте и понимать историю разработки.

Также важно уметь обновлять локальный репозиторий в соответствии с изменениями, внесенными в удаленный репозиторий, например, через интерфейс GitHub. Если код был изменен непосредственно на сайте GitHub, для обновления локальной копии используется команда **git pull**. Она синхронизирует локальный репозиторий с удаленным, загружая все последние изменения и интегрируя их с локальной рабочей копией.

Таким образом, процесс работы с Git включает в себя ряд шагов — от клонирования репозитория до его обновления и синхронизации с удаленным хранилищем, обеспечивая эффективное управление исходным кодом проекта.

Коммиты

Цепочка коммитов в Git представляет собой подробную историю всех изменений, произведенных в репозитории, и служит фундаментальным элементом для управления версиями исходного кода. Каждый коммит фиксирует состояние проекта в определенный момент времени, позволяя разработчикам не только отслеживать, кто и когда внес изменения, но и предоставляя возможность в любой момент вернуться к любому из этих сохраненных состояний.


commit A
file1.txt
+ Hello!
+ Bye!
file2.txt
+ Java
+ HTML
commit B
file1.txt
Hello!
- Bye!
+ JS
file2.txt
Java
+ CSS
HTML
commit C
file1.txt
- Hello!
JS
file2.txt
Java
CSS
- HTML
+ Berlin
current
file1.txt
JS
file2.txt
Java
CSS
Berlin