Что такое система контроля версий.

Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Для контроля версий файлов в этой книге в качестве примера будет использоваться исходный код программного обеспечения, хотя на самом деле вы можете использовать контроль версий практически для любых типов файлов.

1. Локальные системы контроля версий.

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельный каталог (возможно даже, каталог с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть в каком каталоге вы находитесь и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные VCS с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

2. Централизованные системы контроля версий.

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (Centralized Version Control System, далее CVCS). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение CVCS являлось стандартом на протяжении многих лет.

Такой подход имеет множество преимуществ, особенно перед локальными VCS. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать CVCS, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные VCS страдают от той же самой проблемы: когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

3. Распределенные системы контроля версий.

Здесь в игру вступают распределённые системы контроля версий (Distributed Version Control System, далее DVCS). В DVCS (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных. Более того, многие DVCS могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

4. Что такое GIT и основные отличия от других СКВ.

Git — система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием.

Так же необходимо описать следующие команды:

1. Git init

Команда git init создает новый репозиторий Git. Его можно использовать для преобразования существующего неверсионного проекта в репозиторий Git или для инициализации нового пустого репозитория.

2. Git add

Команда git add добавляет изменение в рабочем каталоге в промежуточную область. Он сообщает Git, что вы хотите включить обновления определенного файла в следующий коммит. Однако git add на самом деле не оказывает существенного влияния на репозиторий — изменения фактически не записываются, пока вы не запустите git commit .

3. Git commit

Команда git commit делает снимок текущих изменений проекта. Зафиксированные снимки можно рассматривать как «безопасные» версии проекта — Git никогда не изменит их, если вы явно не попросите его об этом. Перед выполнением git commit команда git add используется для продвижения или «поэтапного» внесения изменений в проект, которые будут сохранены в коммите.

4. Git status

Команда git status отображает состояние рабочего каталога и промежуточной области. Он позволяет вам увидеть, какие изменения были подготовлены, а какие нет и какие файлы не отслеживаются Git.

5. Git reset

Git reset — это мощная команда, которая используется для отмены локальных изменений состояния репозитория Git . Сброс Git работает с «Три деревьями Git». Этими деревьями являются история коммитов (HEAD), промежуточный индекс и рабочий каталог. Существует три параметра командной строки, соответствующие трем деревьям.

6. Git log

Команда git log используется для просмотра истории зафиксированных изменений в репозитории Git. Каждый набор изменений, внесенных разработчиком, записывается как коммит в Git. Команда git log отображает вывод по умолчанию для быстрого просмотра истории коммитов.

7. Git branch

Создание новой ветки.