Лекция 1. Системы контроля версий. Основные сведения.

1.1. Возникновение систем контроля версий.

Исторически людям нужно было организовать совместную работу над каким — либо проектом или заданием. Необходимо было отследить, какие изменения и когда были сделаны, кто их сделал, и инструмент, чтобы откатить всё к предыдущей версии. Так же существовала проблема потери вашей работы из-за поломки компьютера / потери флеш-устройства / случайного удаления файлов. Если брать примеры одиночной работы, часты случаи, когда люди теряли свой многодневный труд из – за подбных поломок. Под влиянием всех этих факторов и родились системы контроля версий, или СКВ.

Системы управления версиями (Version Control Systems, VCS) – это программное обеспечение, призванное автоматизировать работу с историей файла, обеспечить мониторинг изменений, синхронизацию данных и организовать защищенное хранилище проекта. Основная задача систем управления версиями – упростить работу с изменяющейся информацией.

Все системы контроля версий обладают следующими возможностями:

- позволяют команде программистов одновременно работать над одним и тем же проектом;
- минимизируют конфликты между изменениями, вносимыми разработчиками, которые работают над одним проектом;
- автоматически создают архив каждой версии, включающий в себя все изменения проекта.

Далее мы рассмотрим виды, а для лучшего понимания материала введем некоторые понятия. Важно понимать, что общепринятой терминологии не существует, в разных системах могут использоваться различные названия для одних и тех же действий. Ниже приводятся некоторые из наиболее часто используемых вариантов. Приведены английские термины, в литературе на русском языке используется тот или иной перевод или транслитерация:

Changeset, activity (набор изменений) – представляет собой поименованный набор правок, сделанных в локальной копии для какой-то общей цели. В системах, поддерживающих наборы правок, разработчик может объединять локальные правки в группы и выполнять фиксацию логически связанных изменений одной командой, указывая требуемый набор правок в качестве параметра. При этом прочие правки останутся незафиксированными. Типичный пример: ведётся работа над добавлением новой функциональности, а в этот момент обнаруживается критическая ошибка, которую необходимо немедленно исправить. Разработчик создаёт набор изменений для уже сделанной работы и новый — для исправлений. По завершении исправления ошибки отдаётся команда фиксации только второго набора правок.

Check-in, commit, submit – создание новой версии, фиксация изменений. Распространение изменений, сделанных в рабочей копии, на хранилище документов. При этом в хранилище создаётся новая версия изменённых документов.

Check-out, clone – извлечение документа из хранилища и создание рабочей копии.

Patch – внесение изменений в файл.

Rebase – перенос точки ветвления (версии, от которой начинается ветвь) на более позднюю версию основной ветви. Например, после выпуска версии 1.0 проекта в стволе продолжается доработка (исправление ошибок, доработка имеющегося функционала), одновременно начинается работа над новой функциональностью в новой ветви. Через какое-то время в основной ветви происходит выпуск версии 1.1 (с исправлениями); теперь желательно, чтобы ветвь разработки новой функциональности включала изменения, произошедшие в стволе. Вообще, это можно сделать базовыми средствами, с помощью слияния (merge), выделив набор изменений между версиями 1.0 и 1.1 и слив его в ветвь. Но при наличии в системе поддержки перебазирования ветви эта операция делается проще, одной командой: по команде rebase (с параметрами: ветвью и новой базовой версией) система самостоятельно определяет нужные наборы изменений и производит их слияние, после чего для ветви базовой версией становится версия 1.1; при последующем слиянии ветви со стволом система не рассматривает повторно изменения, внесённые между версиями 1.0 и 1.1, так как ветвь логически считается выделенной после версии 1.1.

Revision (версия документа). Системы управления версиями различают версии по номерам, которые назначаются автоматически.

Update, sync – синхронизация рабочей копии до некоторого заданного состояния хранилища. Чаще всего это действие означает обновление рабочей копии до самого свежего состояния хранилища.

Working copy – рабочая (локальная) копия документов.

Репозиторий (repository, depot) — хранилище документов; место, где система управления версиями хранит все документы вместе с историей их изменения и другой служебной информацией.

Ветвь (branch) – направление разработки, независимое от других. Ветвь представляет собой копию части (как правило, одного каталога) хранилища, в которую можно вносить свои изменения, не влияющие на другие ветви.

Слияние (merge, integration) – объединение независимых изменений в единую версию документа. Осуществляется, когда два человека изменили один и тот же файл или при переносе изменений из одной ветки в другую.

Коммит (commit) – фиксация изменений или запись изменений в репозиторий. Коммит происходит на локальной машине.

Фиксация текущий версии проекта — сохранение версии вашей локальной базе.

Разберем общий вид разработки с использованием СКВ на примере:

Предположим, есть некий проект - программа, который вы разрабатываете совместно с несколькими отделами программистов, и вы – его руководитель. Вы создаёте хранилище (репозиторий), и добавляете в его настройки людей, которые могут вносить в него изменения или только читать файлы. Например, разработчикам проекта даёте права на чтение и запись, а аналитикам только право на чтение. В ходе разработки разработчики могут фиксировать свои изменения, и они сохранятся даже при потере копии на компьютере. Так же, если кусок кода перестал работать, СКВ позволяют определять, когда и где был он был изменён.

1.2. Виды СКВ.

Основными видами СКВ являются:

- локальные системы контроля версий;
- централизованные системы контроля версий;
- децентрализованные (распределённые) системы контроля версий.

Локальные СКВ — СКВ, хранящиеся только на локальном компьютере. Обладают очевидным недостатком — при повреждении жёсткого диска данные нельзя будет восстановить.

Централизованные СКВ (ЦСКВ) – такие системы, в которых вся работа производится с центральным хранилищем (все действия, так или иначе, зависят от него).

Распределённые системы контроля версий (РСКВ) – это СКВ, главной парадигмой которых является локализация данных на машине каждого разработчика проекта.

По способу взаимодействия с файлами разделяют: блокирующее/не блокирующее взаимодействие – блокирующие системы контроля версий позволяют наложить запрет на изменение файла, пока один из разработчиков работает над ним, в неблокирующих один файл может одновременно изменяться несколькими разработчиками.

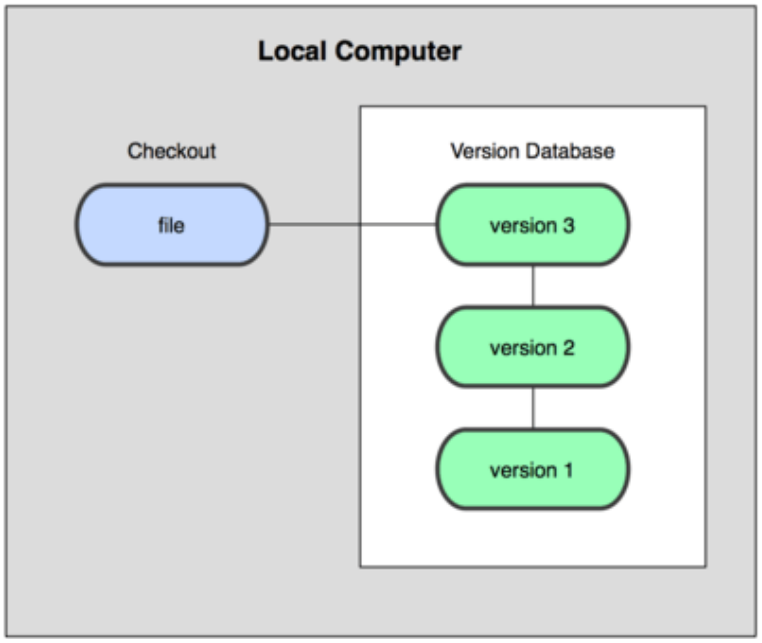
Также важными особенностями СКВ является: для текстовых данных - поддержка слияния изменений, а для бинарных данных - возможность блокировки.

1.2.1 Локальные системы контроля версий (ЛСКВ)

Многие предпочитают контролировать версии, просто копируя файлы в другой каталог (как правило добавляя текущую дату к названию каталога). Такой подход очень распространён, потому что прост, но он и чаще даёт сбои. Очень легко забыть, что ты не в том каталоге, и случайно изменить не тот файл, либо скопировать файлы не туда, куда хотел, и затереть нужные файлы.

Чтобы решить эту проблему, программисты уже давно разработали локальные СКВ с простой базой данных, в которой хранятся все изменения нужных файлов.

Одной из наиболее популярных СКВ такого типа является rcs, которая до сих пор устанавливается на многие компьютеры. Даже в современной операционной системе Mac OS X утилита rcs устанавливается вместе с Developer Tools. Эта утилита основана на работе с наборами патчей между парами версий (*патч* — файл, описывающий различие между файлами), которые хранятся в специальном формате на диске. Это позволяет пересоздать любой файл на любой момент времени, последовательно накладывая патчи.



Схематичное изображение организации ЛСКВ

1.2.2 Централизованные системы контроля версий (ЦСКВ)

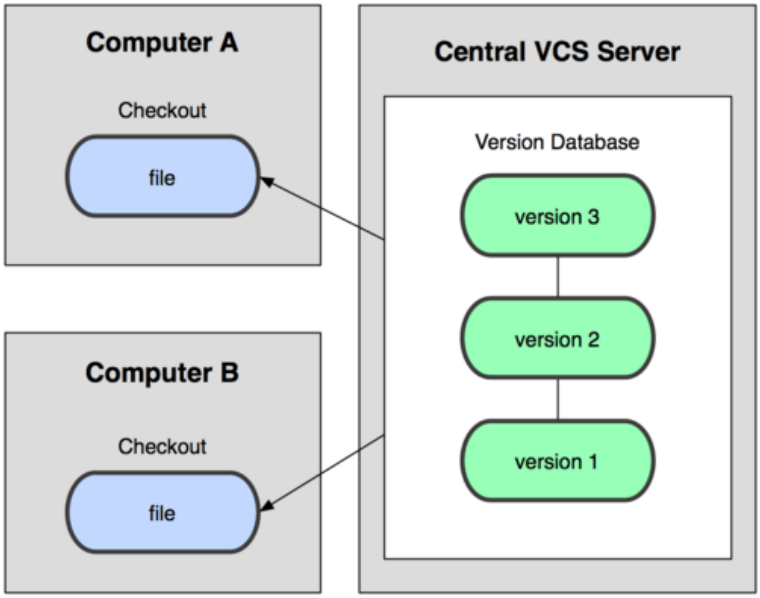
Следующей основной проблемой оказалась необходимость сотрудничать с разработчиками за другими компьютерами. Чтобы решить её, были созданы централизованные системы контроля версий. В таких системах, например CVS, Subversion и Perforce. В данных системах есть центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые получают копии файлов из него. Много лет это было стандартом для систем контроля версий.

Такой подход имеет множество преимуществ, особенно над локальными СКВ. К примеру, все знают, кто и чем занимается в проекте. У администраторов есть чёткий контроль над тем, кто и что может делать, и, конечно, администрировать ЦСКВ намного легче, чем локальные базы на каждом клиенте.

Основным недостатком является то, что централизованный сервер является уязвимым местом всей системы. Если сервер выключается на час, то в течение часа разработчики не могут взаимодействовать, и никто не может сохранить новой версии своей работы. Если же повреждается диск с центральной базой данных и нет резервной копии, вы

теряете абсолютно всё — всю историю проекта, разве что за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей.

Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

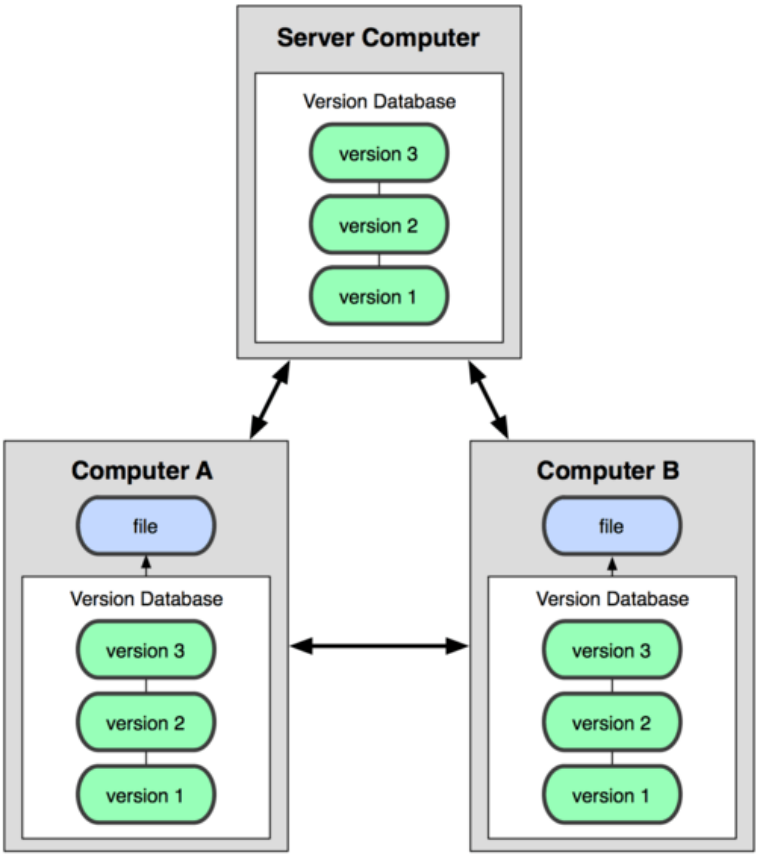


Схематичное изображение организации ЦСКВ

1.2.3 Распределённые (децентрализованные) системы контроля версий (РСКВ)

И в этой ситуации в игру вступают распределённые системы контроля версий. В таких системах как Git, Mercurial, Bazaar или Darcs клиенты не просто выгружают последние версии файлов, а полностью копируют весь репозиторий. Поэтому в случае, когда "умирает" сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных.

Кроме того, в большей части этих систем можно работать с несколькими удалёнными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных



Схематичное изображение организации РСКВ

системах.

В лабораторных работах мы будем подробно изучать ЦСКВ и РСКВ, так как они свободны от недостатков локальных репозиториев и в наши дни используются в реальных проектах.

1. 3. Примеры систем контроля версий

1.3.1 CVS

CVS является централизованной СКВ, и несколько лет назад являлась стандартом де-факто среди контроля версий. CVS приобрела популярность благодаря простой системе поддержки файлов и ревизий в актуальном состоянии. Является предком SVN.

Достоинства:

- Существует много IDE (Integrated Development Environment – среда для редактирования и исполнения исходных файлов проекта), которые используют CVS, включая Xcode (Mac), Eclipse, NetBeans и Emacs.

Недостатки:

- Перемещение или переименование файлов не включается в обновление версии;
- Предоставление символических ссылок на файлы связано с некоторыми рисками безопасности;
- Отсутствие поддержки атомарных операций может привести к повреждению исходного кода;
- Медленные операции установления меток и ветвления;
- Слабая поддержка двоичных файлов.

  В настоящее время активная разработка системы прекращена (последняя версия выпущена в мае 2008 года), в исходный код вносятся только небольшие исправления.

На данный момент CVS является устаревшей системой, потому что она имеет ряд недостатков, и имеются более молодые альтернативные системы управления версиями (например, Subversion, Git, Mercurial), свободные от большинства недостатков CVS.

1.3.2. SVN

Еще одна распространенная централизованная система управления версиями. Большинство проектов с открытым исходным кодом и крупные платформы, такие как Ruby, Python Apache, используют SVN.

Достоинства:

- Новая и значительно улучшенная система, основанная на CVS;
- Допускает атомарные операции (операция или выполнена, или не выполнена совсем (используется, например, при операции фиксации изменений));
- Операции в ветке проекта малозатратны;
- Лучше работает с бинарными файлами и компактно хранит их дифы;
- Доступны различные плагины IDE.

Недостатки:

- Выдает ошибки при переименовании файлов и каталогов;
- Недостаточно команд для управления репозиторием;
- Сложности с полным удалением информации о файлах попавших в репозиторий, так как в нем всегда остается информация о предыдущих изменениях файла, и непредусмотрено никаких штатных средств для полного удаления данных о файле из репозитория;
- SVN работает медленнее по сравнению с другими системами управления версиями;
- Отменённые изменения будут потеряны навсегда;
- Слабо поддерживаются операции слияния веток проекта;

1.3.3. GIT

Является децентрализованной СКВ. Благодаря распределенной форме управления без необходимости использования оригинального программного обеспечения многие проекты с открытым исходным кодом предпочитают Git. На данный момент является стандартом де-факто в мире разработки ПО.

Достоинства:

- Почти все отрицательные черты CVS/SVN устранены;
- Высокая скорость работы распределенной системы контроля версий;
- Легкость проведения различных операций с ветками проекта;
- Пользователи могут получить доступ к полному дереву истории в режиме офлайн;
- Операция слияния сделана удобно, что позволяет нам существенно изменить рабочий процесс на более оптимальный (будет подробнее показано в 3 лекции «git».).

Недостатки

- Высокий порог вхождения для пользователей SVN;

Замечание: ограничения поддержки были устранены, например, в Git Bash, которая имеет почти те же возможности для работы с Git, что доступны на Linux.

1.3.4. Mercurial

Является децентрализованной СКВ. Считается эффективной для крупных проектов, в которых участвует много разработчиков и проектировщиков. Mercurial – это высокопроизводительная система, предлагающая оптимальную скорость. Она также известна своей простотой и подробной документацией.

Достоинства:

- Низкий порог вхождения по сравнению с **Git**;
- Подробная документация;
- Распределенная модель;
- Высокопроизводительная система с отличной скоростью.

Недостатки:

- Нельзя объединить две родительские ветки;
- Меньше возможностей для нестандартных решений;
- Ориентирован на работу в консоли.

1.3.5. Bazaar

Уникальна тем, что может использоваться с распределенной и централизованной базой кода. Это делает ее универсальной системой контроля версий. Кроме этого Bazaar позволяет использовать детальный уровень управления. Ее можно легко развернуть в самых разных сценариях, что делает ее адаптивной и гибкой для всевозможных проектов.

Достоинства:

- Идеально подходит для разнообразных проектов;
- Включает в себя настраиваемый набор функций, который подходит для разнообразных проектов;
- В отличие от чисто распределённых систем контроля версий, которые не используют центральный сервер, Bazaar поддерживает работу как с сервером так и без него. Возможно даже использовать оба метода одновременно для одного и того же проекта.

Недостатки:

- Является новой и недостаточно проработанной системой управления версиями;
- Для полноценного функционирования необходимо устанавливать достаточно большое количество плагинов, позволяющих полностью раскрыть все возможности системы контроля версий.

Использованные источники:

- http://all-ht.ru/inf/prog/p_0_1.html
- https://git-scm.com/book/ru/v2