УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите приведенную лекцию, законспектируйте основные понятия контроля версий программного обеспечения.

Ответы на вопросы, фотоотчет, предоставить преподавателю на e-mail: r.bigangel@gmail.com до 06.02.2023.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)111-37-59, (Viber, WhatsApp), vk.com: https://vk.com/daykini

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 4

Тема: Инструментальные средства контроля версий ПО и поддержки коллективной разработки.

Цель: Изучить основные средства контроля версий ПО

1. Понятие системы контроля версий

Система контроля версий (СКВ) — 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.

2. Использование системы контроля версий

Такого рода системы в большинстве своем используются при разработке программного обеспечения, чтобы можно было хранить исходные коды разрабатываемых программ. СКВ позволяет разработчикам хранить прошлые версии файлов из разработки и доставать их оттуда. Она хранит информацию о версии каждого файла (и полную структуру проекта) в коллекции, обычно называемой репозиторием. Но, тем не менее, данные системы могут использоваться и в других областях знаний, которые включают в себя огромное

количество часто изменяющихся электронных документов. Например, они всё чаще применяются в САПР, обычно, в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления.

Использование СКВ, безусловно, обязательно для разработчика, если проект больше нескольких сот строк или если для проекта совместно работают несколько разработчиков.

Системы контроля версий решают следующие проблемы:

– хранение версий файлов;

– возможность получить любые предыдущие версии хранимых файлов;

– просмотр изменений внесенных между заданными в запросе версиями;

– сохранение и просмотр комментариев и авторов к внесенным изменениям.

3. Типичный порядок работы с системой

Каждая СКВ имеет свои специфические особенности в наборе команд, порядке работы пользователей и администрировании. Тем не менее, общий порядок работы для большинства СКВ совершенно стереотипен. Здесь предполагается, что проект, каким бы он ни был, уже существует и на сервере размещён его репозиторий, к которому разработчик получает доступ.

3.1 Начало работы с проектом

Первым действием, которое должен выполнить разработчик, является извлечение рабочей копии проекта или той его части, с которой предстоит работать. Это действие выполняется с помощью стандартной команды извлечения версии (checkout или clone) либо специальной команды, фактически выполняющей то же самое действие. Разработчик задаёт версию, которая должна быть скопирована, по умолчанию обычно копируется последняя (или выбранная

администратором в качестве основной) версия.

По команде извлечения устанавливается соединение с сервером, и проект в виде дерева каталогов и файлов копируется на компьютер разработчика. Обычной практикой является дублирование рабочей копии: помимо основного каталога с проектом на локальный диск дополнительно записывается ещё одна его копия. Работая с проектом, разработчик изменяет только файлы основной рабочей копии. Вторая локальная копия хранится в качестве эталона, позволяя в любой момент без обращения к серверу определить, какие изменения внесены в конкретный файл или проект в целом и от какой версии была сделана рабочая копия; как правило, любая попытка ручного изменения этой копии приводит к ошибкам в работе программного обеспечения СКВ.

3.2 Ежедневный цикл работы

При некоторых вариациях, определяемых особенностями системы и деталями принятого бизнес-процесса, обычный цикл работы разработчика в течение рабочего дня выглядит следующим образом:

1) Обновление рабочей копии. По мере внесения изменений в основную версию проекта рабочая копия на компьютере разработчика стареет: расхождение её с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений. Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версии, для чего разработчик выполняет операцию обновления рабочей копии (update) насколько возможно часто.

2) Модификация проекта. Разработчик модифицирует проект, изменяя входящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу СКВ.

3) Фиксация изменений. Завершив очередной этап работы над заданием, разработчик фиксирует (commit) свои изменения, передавая их на сервер, либо в основную ветвь, если работа над заданием полностью завершена, либо в отдельную ветвь разработки данного задания. СКВ может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей

копии. При наличии в системе поддержки отложенных изменений (shelving) изменения могут быть переданы на сервер без фиксации. Если утверждённая политика работы в СКВ это позволяет, то фиксация изменений может проводиться не ежедневно, а только по завершении работы над заданием; в этом случае до завершения работы все связанные с заданием изменения сохраняются только в локальной рабочей копии разработчика.

3.3 Ветвления

Делать мелкие исправления в проекте можно путём непосредственной правки рабочей копии и последующей фиксацией изменений прямо в главной ветви (стволе) на сервере. Однако при выполнении сколько-нибудь значительных по объёму работ такой порядок становится неудобным: отсутствие фиксации промежуточных изменений на сервере не позволяет работать над чем-либо в групповом режиме, кроме того, повышается риск потери изменений при локальных авариях и теряется возможность анализа и возврата к предыдущим вариантам кода в пределах данной работы. Поэтому для таких изменений обычной практикой является создание ветвей (branch), создание версии нового варианта проекта или его части, разработка в котором ведётся параллельно с изменениями в основной версии. Когда работа, для которой создана ветвь, выполнена, ветвь реинтегрируется в основную ветвь. Это может делаться командой слияния (merge), либо путём создания патча (patch), содержащего внесённые в ходе разработки ветви изменения и применения этого патча к текущей основной версии проекта.

3.4 Слияние версий

Три вида операций, выполняемых в системе управления версиями, могут приводить к необходимости объединения изменений:

- Обновление рабочей копии;
- Фиксация изменений;
- Слияние ветвей.

В каждой системе существует механизм автоматического слияния, который работает, основываясь на следующих принципах:

– Изменения могут состоять в модификации содержимого файла, создании нового файла или каталога, удалении или переименовании ранее существовавшего файла или каталога в проекте;

– Если два изменения относятся к разным и не связанным между собой файлам и/или каталогам, они всегда могут быть объединены автоматически. Их объединение состоит в том, что изменения, сделанные в каждой версии проекта, копируются в объединяемую версию;

– Создание, удаление и переименование файлов в каталогах проекта могут быть объединены автоматически, если только они не конфликтуют между собой. В этом случае изменения, сделанные в каждой версии проекта, копируются в объединяемую версию.

Конфликтующими обычно являются:

– Удаление и изменение одного и того же файла или каталога;

– Удаление и переименование одного и того же файла или каталога;

– Создание в разных версиях файла с одним и тем же именем и разным содержимым;

– Изменения в пределах одного текстового файла, сделанные в разных версиях, могут быть объединены, если они находятся в разных местах этого файла и не пересекаются. В этом случае в объединённую версию вносятся все сделанные изменения;

– Изменения в пределах одного файла, если он не является текстовым, всегда являются конфликтующими и не могут быть объединены автоматически.

Во всех случаях базовой версией для слияния является версия, в которой было произведено разделение сливаемых версий. Если это операция фиксации изменений, то базовой версией будет версия последнего обновления перед фиксацией, если обновление — то версия предыдущего обновления, если слияние ветвей — то версия, в которой была создана соответствующая ветвь. Соответственно, сопоставляемыми наборами изменений будут наборы изменений, сделанных от базовой до текущей версии во всех объединяемых вариантах.

Абсолютное большинство современных систем управления версиями ориентировано, в первую очередь, на проекты разработки программного обеспечения, в которых основным видом содержимого файла является текст. Соответственно, механизмы автоматического слияния изменений ориентируются на обработку текстовых файлов.

При определении допустимости слияния изменений в пределах одного и того же текстового файла работает типовой механизм построчного сравнения текстов (примером его реализации является системная утилита GNU diff), который сравнивает объединяемые версии с базовой и строит список изменений, то есть добавленных, удалённых и заменённых наборов строк. Найденные наборы изменённых строк, которые не пересекаются между собой, считаются совместимыми и их слияние делается автоматически. Если в сливаемых файлах находятся изменения, затрагивающие одну и ту же строку файла, это приводит к конфликту. Такие файлы могут быть объединены только вручную. Любые файлы, кроме текстовых, с точки зрения СКВ являются бинарными и не допускают автоматического слияния.

3.5 Конфликты и их разрешение

Ситуация, когда при слиянии нескольких версий сделанные в них изменения пересекаются между собой, называют конфликтом. При конфликте изменений система управления версиями не может автоматически создать объединённый проект, и вынуждена обращаться к разработчику. Как уже говорилось выше, конфликты могут возникать на этапах фиксации изменений, обновления или слияния ветвей. Во всех случаях при обнаружении конфликта соответствующая операция прекращается до его разрешения.

Для разрешения конфликта система, в общем случае, предлагает разработчику три варианта конфликтующих файлов: базовый, локальный и серверный. Конфликтующие изменения либо показываются разработчику в специальном программном модуле объединения изменений, либо просто помечаются специальной разметкой прямо в тексте объединённого файла.

Конфликты в файловой системе разрешаются проще: там может конфликтовать только удаление файла с одной из прочих операций, а порядок файлов в каталоге не имеет значения, так что разработчику остаётся лишь выбрать, какую операцию нужно сохранить в сливаемой версии.

3.6 Блокировки

Механизм блокировки позволяет одному из разработчиков взять только в собственное использование файл или группу файлов для внесения в них изменений. На то время, пока файл заблокирован, он остаётся доступным всем остальным разработчикам только на чтение, и любая попытка внести в него изменения отвергается сервером. Технически блокировка может быть организована по-разному. Типичным для современных систем является следующий механизм:

– Файлы, для работы с которыми требуется блокировка, помечаются специальным флагом «блокируемый»;

– Если файл помечен как блокируемый, то при извлечении рабочей копии с сервера он получает в локальной файловой системе атрибут «только для чтения», что препятствует его случайному редактированию;

– Разработчик, желающий изменить блокируемый файл, вызывает специальную команду блокировки (lock) с указанием имени этого файла. В результате работы этой команды происходит следующее:

1. сервер проверяет, не заблокирован ли уже файл другим разработчиком; если это так, то команда блокировки завершается с ошибкой.

2. файл на сервере помечается как «заблокированный», с сохранением идентификатора заблокировавшего его разработчика и времени блокировки;

3. если блокировка на сервере прошла удачно, на локальной файловой системе с файла рабочей копии снимается атрибут «только для чтения», что позволяет начать его редактировать.

– Если в процессе работы выясняется, что файл изменять не нужно, он может вызвать команду снятия блокировки (unlock**,** release lock). Все изменения

файла будут отменены, локальный файл вернётся в состояние «только для чтения», с файла на сервере будет снят атрибут «заблокирован» и другие разработчики получат возможность изменять этот файл;

– По завершении работы с блокируемым файлом разработчик фиксирует изменения. Обычно блокировка при этом снимается автоматически.

3.7 Версии проекта, теги

Система управления версиями обеспечивает хранение всех существовавших вариантов файлов и, как следствие, всех вариантов проекта в целом, имевших место с момента начала его разработки. Но само понятие «версии» в разных системах может трактоваться двумя различными способами.

Одни системы поддерживают версионность файлов. Это означает, что любой файл, появляющийся в проекте, получает собственный номер версии. При каждой фиксации разработчиком изменений, затрагивающих файл, соответствующая часть фиксируемых изменений применяется к файлу, и файл получает новый, обычно следующий по порядку, номер версии.

Для других систем понятие «версия» относится не к отдельному файлу, а к репозиторию целиком. Вновь созданный пустой репозиторий имеет версию 1 или 0, любая фиксация изменений приводит к увеличению этого номера. Номера версии отдельного файла здесь, фактически, не существует.

Однако оба варианта не слишком удобны. Для более удобной пометки версий проекта системы контроля версиями поддерживают понятие тегов.

Тег - это символическая метка, которая может быть связана с определённой версией файла и/или каталога в репозитории. С помощью соответствующей команды всем или части файлов проекта, отвечающим определённым условиям может быть присвоена заданная метка. Таким образом можно идентифицировать версию проекта, зафиксировав его состояние на некоторый желаемый момент. Как правило, система тегов достаточно гибкая и позволяет пометить одним тегом и не одновременные версии файлов и каталогов. Это позволяет собрать «версию проекта» любым произвольным образом.

4. Типы

4.1 Локальные системы контроля версий

Наиболее предпочтительной СКВ для слежения за домашними разработками будет локальный тип. Для решения этой задачи были разработаны специальные СКВ с простой базой данных, в которой хранятся все локальные файлы.

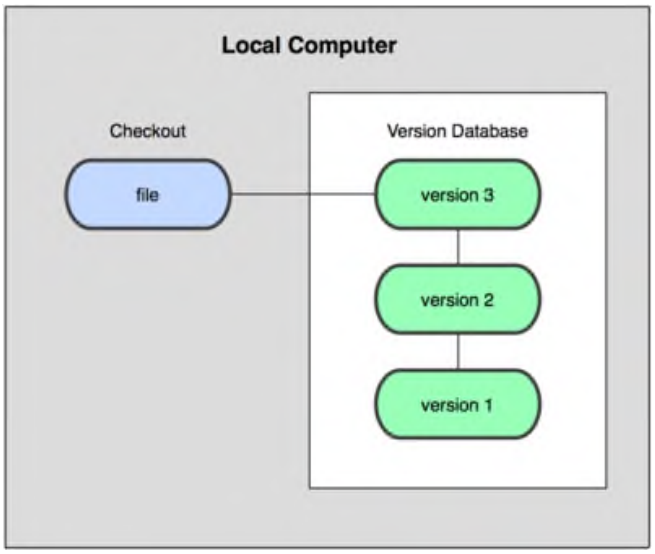

Рисунок 4.1 Схема локальной СКВ

Одной из наиболее популярных СКВ такого типа является RCS, которая до сих пор устанавливается на многие компьютеры.

4.2 Централизованная модель

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет бóльшую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, так называемая «рабочая копия». Может быть получена последняя версия или любая из предыдущих, которая может быть

выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время.

В таких системах, например CVS, Subversion и Perforce, 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.

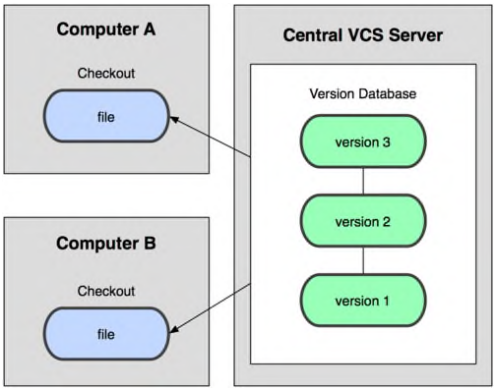

Рисунок 4.2 Схема централизованного контроля версий

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.

Однако при таком подходе есть и несколько серьёзных недостатков. Наиболее очевидный — 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 СКВ подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

4.3 Распределённые системы контроля версий

Такие системы используют распределённую модель вместо традиционной клиент-серверной. Они, в общем случае, не нуждаются в централизованном хранилище: вся история изменения документов хранится на каждом компьютере, в локальном хранилище, и при необходимости отдельные фрагменты истории локального хранилища синхронизируются с аналогичным хранилищем на другом компьютере. Поэтому в случае, когда отключается сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. В некоторых таких системах локальное хранилище располагается непосредственно в каталогах рабочей копии.

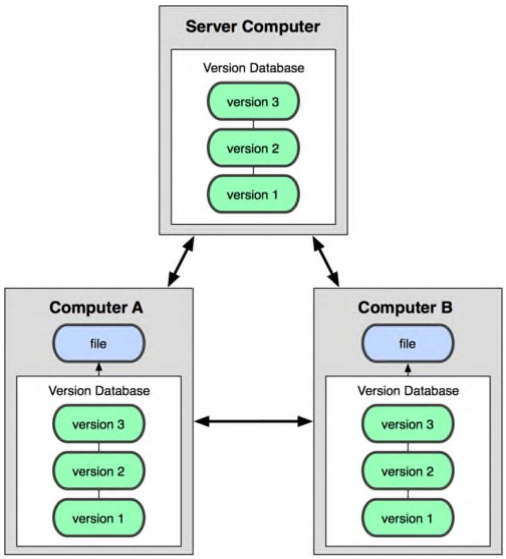

Рисунок 4.3 Распределённая модель СКВ

Когда пользователь такой системы выполняет обычные действия, такие как извлечение определённой версии документа, создание новой версии и тому подобное, он работает со своей локальной копией хранилища. По мере внесения изменений, хранилища, принадлежащие разным разработчикам, начинают различаться, и возникает необходимость в их синхронизации. Такая синхронизация может осуществляться с помощью обмена патчами или так называемыми наборами между пользователями.

Описанная модель логически близка созданию отдельной ветки для каждого разработчика в классической системе управления версиями. Отличие состоит в

том, что до момента синхронизации другие разработчики этой ветви не видят. Пока разработчик изменяет только свою ветвь, его работа не влияет на других участников проекта и наоборот. По завершении обособленной части работы, внесённые в ветви изменения сливают с основной (общей) ветвью. Как при слиянии ветвей, так и при синхронизации разных хранилищ возможны конфликты версий. На этот случай во всех системах предусмотрены те или иные методы обнаружения и разрешения конфликтов слияния.

С точки зрения пользователя распределённая система отличается необходимостью создавать локальный репозиторий и наличием в командном языке двух дополнительных команд: команды получения репозитория от удалённого компьютера и передачи своего репозитория на удалённый компьютер. Первая команда выполняет слияние изменений удалённого и локального репозиториев с помещением результата в локальный репозиторий; вторая — наоборот, выполняет слияние изменений двух репозиториев с помещением результата в удалённый репозиторий. Как правило, команды слияния в распределённых системах позволяют выбрать, какие наборы изменений будут передаваться в другой репозиторий или извлекаться из него, исправлять конфликты слияния непосредственно в ходе операции или после её неудачного завершения, повторять или возобновлять неоконченное слияние. Обычно передача своих изменений в чужой репозиторий завершается удачно только при условии отсутствия конфликтов. Если конфликты возникают, пользователь должен сначала слить версии в своём репозитории, и лишь затем передавать их другим.

Обычно рекомендуется организовывать работу с системой так, чтобы пользователи всегда или преимущественно выполняли слияние у себя в репозитории. То есть, в отличие от централизованных систем, где пользователи передают свои изменения на центральный сервер, когда считают нужным, в распределённых системах более естественным является порядок, когда слияние версий инициирует тот, кому нужно получить его результат (например, разработчик, управляющий сборочным сервером).

Основные преимущества распределённых систем — их гибкость и

значительно бóльшая (по сравнению с централизованными системами) автономия отдельного рабочего места. Каждый компьютер разработчика является, фактически, самостоятельным и полнофункциональным сервером, из таких компьютеров можно построить произвольную по структуре и уровню сложности систему, задав желаемый порядок синхронизации.

К недостаткам распределённых систем можно отнести увеличение требуемого объёма дисковой памяти: на каждом компьютере приходится хранить полную историю версий, тогда как в централизованной системе на компьютере разработчика обычно хранится лишь рабочая копия, то есть срез репозитория на какой-то момент времени и внесённые изменения. Менее очевидным, но неприятным недостатком является то, что в распределённой системе практически невозможно реализовать некоторые виды функциональности, предоставляемые централизованными системами. Это:

- Блокировка файла или группы файлов (для хранения признака блокировки нужен общедоступный и постоянно находящийся в онлайне центральный сервер). Это вынуждает применять специальные административные меры, если приходится работать с бинарными файлами, непригодными для автоматического слияния;
- Слежение за определённым файлом или группой файлов;
- Единая сквозная нумерация версий системы и/или файлов, в которой номер версии монотонно возрастает. В распределённых системах приходится обходиться локальными обозначениями версий и применять теги, назначение которых определяется соглашением между разработчиками или корпоративными стандартами фирмы;
- Локальная работа пользователя с отдельной, небольшой по объёму выборкой из значительного по размеру и внутренней сложности хранилища на удалённом сервере.

Можно выделить следующие типичные ситуации, в которых использование распределённой системы даёт заметные преимущества:

- Периодическая синхронизация нескольких компьютеров под управлением одного разработчика. Использование распределённой системы

избавляет от необходимости выделять один из компьютеров в качестве сервера;

- Совместная работа над проектом небольшой территориально распределённой группы разработчиков без выделения общих ресурсов. Как и в предыдущем случае, реализуется схема работы без главного сервера, а актуальность репозиториев поддерживается периодическими синхронизациями по схеме «каждый с каждым»;
- Крупный распределённый проект, участники которого могут долгое время работать каждый над своей частью, при этом не имеют постоянного подключения к сети. Такой проект может использовать централизованный сервер, с которым синхронизируются копии всех его участников. Возможна работа и без «групповых» серверов, тогда разработчики одной группы синхронизируют изменения между собой, после чего любой из них передаёт изменения на центральный сервер.