Системы управления версиями

Конфигурационное управление (*software configuration management*, SCM) в программной инженерии — комплекс методов, направленных на систематический учёт изменений, вносимых разработчиками в программный продукт в процессе его разработки и сопровождения, сохранение целостности системы после изменений, предотвращение нежелательных и непредсказуемых эффектов, формализацию процесса внесения изменений.

В целом, конфигурационное управление отвечает на вопрос: «Кто-то уже сделал нечто, как нам это воспроизвести?»

Изначально управление конфигурацией применялось не в программировании.
Под *конфигурацией* понимался состав деталей конечного продукта и «взаимное расположение частей» физического изделия. Таким образом, конфигурацией можно управлять, контролируя документы, описывающие конечный продукт, требования к нему, всю его проектную и технологическую документацию.

В связи с высокой динамичностью сферы разработки ПО, в ней конфигурационное управление особенно полезно. К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д. Степень формальности выполнения данных процедур зависит от размеров проекта, и при правильной её оценке данная концепция может быть очень полезна.

Система управления версиями (*Version Control System, VCS* или *Revision Control System*) — 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.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии (PDM). Управление версиями используется в инструментах конфигурационного управления (*Software Configuration Management Tools*).

Программное обеспечение Википедии ведёт историю изменений для всех её статей, используя методы, аналогичные тем, которые применяются в системах управления версиями.

Общие сведения

Ситуация, в которой электронный документ за время своего существования претерпевает ряд изменений, достаточно типична. При этом часто бывает важно иметь не только последнюю версию, но и несколько предыдущих. В простейшем случае можно просто хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен (приходится хранить несколько практически идентичных копий), требует повышенного внимания и дисциплины и часто ведёт к ошибкам, поэтому были разработаны средства для автоматизации этой работы.

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. Сервер может использовать т. н. дельта-компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями,

что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

Иногда создание новой версии выполняется незаметно для пользователя (прозрачно), либо прикладной программой, имеющей встроенную поддержку такой функции, либо за счёт использования специальной файловой системы. В этом случае пользователь просто работает с файлом, как обычно, и при сохранении файла автоматически создаётся новая версия.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, то один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения. Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений, обычно, возможно только для текстовых файлов и при условии, что изменялись разные (непересекающиеся) части этого файла. Такое ограничение связано с тем, что большинство систем управления версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Часто выполнить слияние невозможно ни в автоматическом, ни в ручном режиме, например, если формат файла неизвестен или слишком сложен. Некоторые системы управления версиями дают возможность заблокировать файл в хранилище. Блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла (например, средствами файловой системы) и обеспечивает, таким образом, исключительный доступ только тому пользователю, который работает с документом.

Многие системы управления версиями предоставляют ряд других возможностей:

- Позволяют создавать разные варианты одного документа, т. н. *ветки*, с общей историей изменений до точки ветвления и с разными — после неё.
- Дают возможность узнать, кто и когда добавил или изменил конкретный набор строк в файле.
- Ведут журнал изменений, в который пользователи могут записывать пояснения о том, что и почему они изменили в данной версии.
- Контролируют права доступа пользователей, разрешая или запрещая чтение или изменение данных, в зависимости от того, кто запрашивает это действие.

Типичный порядок работы с системой

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

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 команды извлечения версии (обычно **checkout** или **clone**). 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 (или его часть — один из каталогов с подкаталогами) в виде дерева каталогов и файлов копируется на компьютер разработчика. Обычной практикой является дублирование рабочей копии: помимо основного каталога с проектом на локальный диск (либо в отдельный, специально выбранный каталог, либо

в системные подкаталоги основного дерева проекта) 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 «отпочкована» рабочая копия; как правило, любая попытка ручного изменения этой копии приводит к ошибкам в работе программного обеспечения VCS.

Ежедневный цикл работы

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

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 (см. ниже). 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 (реальная частота обновлений определяется частотой внесения изменений, зависящей от активности разработки и числа разработчиков, а также временем, затрачиваемым на каждое обновление — если оно велико, разработчик вынужден ограничивать частоту обновлений, чтобы не терять время).

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 VCS.

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). VCS 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 VCS 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.

Ветвления

Делать мелкие исправления в проекте можно путём непосредственной правки рабочей копии и последующей фиксации изменений прямо в главной ветви (в стволе) на сервере. Однако при выполнении объёмных 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**), то есть «отпочковывание» от ствола в какой-то версии нового варианта проекта или его части, разработка в котором ведётся параллельно с изменениями в основной версии. Ветвь создаётся специальной командой. Рабочая копия ветви может быть создана заново обычным образом (командой извлечения рабочей копии, с указанием адреса или идентификатора ветви), либо путём переключения имеющейся рабочей копии на заданную ветвь.

Базовый рабочий цикл при использовании ветвей остаётся точно таким же, как и в общем случае: разработчик периодически обновляет рабочую копию (если с ветвью работает более одного человека) и фиксирует в ней свою ежедневную работу. Иногда ветвь разработки так и остаётся самостоятельной (когда изменения порождают новый вариант проекта, который далее развивается отдельно от основного), но чаще всего, когда работа, для которой создана ветвь, выполнена, ветвь реинтегрируется в ствол (основную ветвь). Это может делаться командой слияния (обычно **merge**), либо путём создания патча (**patch**), содержащего внесённые в ходе разработки ветви изменения и применения этого патча к текущей основной версии проекта.

Слияние версий

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

- Обновление рабочей копии (изменения, сделанные в основной версии, сливаются с локальными).
- Фиксация изменений (локальные изменения сливаются с изменениями, уже зафиксированными в основной версии).
- Слияние ветвей (изменения, сделанные в одной ветви разработки, сливаются с изменениями, сделанными в другой).

Во всех случаях ситуация принципиально одинакова и имеет следующие характерные черты:

1. Ранее была сделана копия дерева файлов и каталогов репозитория или его части.
2. Впоследствии и в оригинальное дерево, и в копию были независимо внесены некоторые изменения.
3. Требуется объединить изменения в оригинале и копии таким образом, чтобы не нарушить логическую связность проекта и не потерять данные.

Совершенно очевидно, что при невыполнении условия (2) (то есть если изменения были внесены только в оригинал или только в копию) объединение элементарно — достаточно скопировать изменённую часть туда, где изменений не было. В противном случае слияние изменений превращается в нетривиальную задачу, во многих случаях требующую вмешательства разработчика. В целом механизм автоматического слияния изменений 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 (в случае, если система поддерживает операцию переименования).
  - 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, то есть файлов,

содержащих *текст*, состоящий из *строк* буквенно-цифровых символов, пробелов и табуляций, разделённых символами перевода строки.

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. Минимальной единицей данных для этого алгоритма является строка, даже самое малое отличие делает строки различными. С учётом того, что символы-разделители, в большинстве случаев, не несут смысловой нагрузки, механизм слияния может игнорировать эти символы при сравнении строк.

Те 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 VCS являются бинарными и не допускают автоматического слияния.

Конфликты и их разрешение

Ситуацию, 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 (в этом случае там демонстрируются сливаемые варианты и динамически изменяющийся в зависимости от команд пользователя объединённый вариант файла), либо просто помечаются специальной разметкой прямо в тексте объединённого файла (тогда разработчик должен сам сформировать желаемый текст в спорных местах и сохранить его).

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.

Блокировки

Механизм блокировки позволяет одному из разработчиков захватить в монопольное 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». Такая пометка может ставиться автоматически при добавлении файла в проект, обычно для этого предварительно создаётся список масок имён файлов, которые при добавлении должны становиться блокируемыми.
- 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 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 «файл заблокирован другим пользователем»

и разработчик, вызывавший её, должен ожидать, пока другой пользователь не снимет свою блокировку;

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.

- Разработчик работает с заблокированным файлом. 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, хотя в некоторых системах блокировку требуется снимать вручную после фиксации, либо указывать в команде фиксации изменений соответствующий параметр. Так или иначе, при этом файл после изменений теряет флаг «заблокирован» и может быть изменён другими разработчиками.

Массовое использование блокировок, когда все или большинство файлов в проекте являются блокируемыми и для любых изменений необходимо заблокировать соответствующий набор файлов, называется ещё стратегией «блокированного извлечения». Ранние системы управления версиями поддерживали исключительно эту стратегию, предотвращая таким способом появление конфликтов на корню. В современных VCS предпочтительным является использование неблокирующих извлечений, блокировки же считаются скорее неизбежным злом, которое нужно по возможности ограничивать. Недостатки использования блокировок очевидны:

- Блокировки просто мешают продуктивной работе, поскольку вынуждают ожидать освобождения блокированных файлов, хотя в большинстве случаев даже совместные изменения одних и тех же файлов, которые делаются в ходе разных по смыслу работ, не пересекаются и объединяются при слиянии автоматически.
- Частота возникновения конфликтов и сложность их разрешения в большинстве случаев не настолько велики, чтобы создать серьёзные затруднения. Возникновение же серьёзного конфликта изменений чаще всего сигнализирует либо о существенном расхождении во мнениях разных разработчиков относительно дизайна одного и того же фрагмента, либо о неправильной организации работы (когда два или более разработчиков делают одно и то же).
- Блокировки создают административные проблемы. Типичный пример: разработчик может забыть снять блокировку с занятых им файлов, уходя в отпуск. Для разрешения подобных проблем приходится применять административные меры, в том числе включать в систему технические средства для сброса неверных блокировок, но и при их наличии на приведение системы в порядок расходуется время.

С другой стороны, в некоторых случаях использование блокировок вполне оправданно. Очевидным примером является организация работы с бинарными файлами, для которых нет инструментальных средств слияния изменений либо такое слияние принципиально невозможно (как, например, для файлов изображений). Если автоматическое слияние невозможно, то при обычном порядке работы любое параллельное изменение подобных файлов будет приводить к конфликту. В данном случае гораздо удобнее сделать такой файл блокируемым, чтобы гарантировать, что любые изменения в него будут вноситься только последовательно.

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 двояко.

Одни системы поддерживают версионность *файлов*. Это означает, что любой файл, появляющийся в проекте, получает собственный номер версии (обычно — номер 1, условной «нулевой» версией файла считается пустой файл с тем же именем). При каждой

фиксации разработчиком изменений, затрагивающих файл, соответствующая часть фиксируемых изменений применяется к файлу и файл получает новый, обычно следующий по порядку, номер версии. Поскольку фиксации обычно затрагивают только часть файлов в репозитории, номера версий файлов, имеющиеся на один и тот же момент времени, со временем расходятся, и проект в целом (то есть весь набор файлов репозитория), фактически, никакого «номера версии» не имеет, поскольку состоит из множества файлов с различными номерами версий. Подобным образом работает, например, система управления версиями CVS.

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 (то есть даже при изменении одного файла на один байт весь репозиторий считается изменённым и получает новый номер версии). Таким способом трактует номера версий, например, система Subversion. Номера версии отдельного файла здесь, фактически, не существует, условно можно считать таковым текущий номер версии репозитория (то есть считать, что при каждом изменении, внесённом в репозиторий, все его файлы меняют номер версии, даже те, которые не менялись). Иногда, говоря о «версии файла» в таких системах, имеют в виду ту версию репозитория, в которой файл был последний раз (до интересующего нас момента) изменён.

Для практических целей обычно имеет значение не отдельный файл, а весь проект целиком. В системах, поддерживающих версионность отдельных файлов, для идентификации определённой версии проекта можно использовать дату и время — тогда версия проекта будет состоять из тех версий входящих в него файлов, которые имелись в репозитории на указанный момент времени. Если поддерживается версионность репозитория в целом, номером версии проекта может выступать номер версии репозитория. Однако оба варианта не слишком удобны, так как ни дата, ни номер версии репозитория обычно не несут информации о значимых изменениях в проекте, о том, насколько долго и интенсивно над ним работали. Для более удобной пометки версий проекта (или его частей) системы управления версиями поддерживают понятие **тегов**.

Тег (tag) — 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 (например, входящим в головную версию главной ветви проекта на определённый момент времени) может быть присвоена заданная метка. Таким образом можно идентифицировать версию проекта (версия «XX.XXX.XXX» — это набор версий файлов репозитория, имеющих тег «XX.XXX.XXX»), зафиксировав таким образом 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. С точки зрения пользователя системы пометка тегами может выглядеть по-разному. В некоторых системах она изображается именно как пометка (тег можно создать, применить к определённым версиям файлов и каталогов, снять). В других системах (например, Subversion) тег представляет собой просто отдельный каталог на файловом дереве репозитория, куда из ствола и ветвей проекта с помощью команды копирования делаются копии нужных версий файлов. Так что визуально тег — это просто вынесенная в отдельный каталог копия определённых версий файлов репозитория. По соглашению в дерево каталогов, соответствующее тегу, запрещена фиксация изменений (то есть версия проекта, представляемая тегом, является неизменной).

Базовые принципы разработки ПО в VCS

Порядок использования системы управления версиями в каждом конкретном случае определяется техническими регламентами и правилами, принятыми в конкретной фирме или организации, разрабатывающей проект. Тем не менее, общие принципы правильного использования VCS немногочисленны и едины для любых разработок и систем управления версиями.

1. Любые рабочие, тестовые или демонстрационные версии проекта собираются только из репозитория системы. «Персональные» сборки, включающие ещё незафиксированные изменения, могут делать только разработчики для целей промежуточного тестирования.

Таким образом, гарантируется, что репозиторий содержит всё необходимое для создания рабочей версии проекта.
2. Текущая версия главной ветви всегда корректна. Не допускается фиксация в главной ветви неполных или не прошедших хотя бы предварительное тестирование изменений. В любой момент сборка проекта, проведённая из текущей версии, должна быть успешной.
3. Любое значимое изменение должно оформляться как отдельная ветвь. Промежуточные результаты работы разработчика фиксируются в эту ветвь. После завершения работы над изменением ветвь объединяется со стволом. Исключения допускаются только для мелких изменений, работа над которыми ведётся одним разработчиком в течение не более чем одного рабочего дня.
4. Версии проекта помечаются тегами. Выделенная и помеченная тегом версия более никогда не изменяется.

Распределённые системы управления версиями

Также известны как *Distributed Version Control System*, DVCS. 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. В некоторых таких системах локальное хранилище располагается непосредственно в каталогах рабочей копии.

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 изменений* (англ. *change sets*) между пользователями.

Описанная модель логически близка созданию отдельной ветки для каждого разработчика в классической системе управления версиями (в некоторых распределённых системах перед началом работы с локальным хранилищем нужно создать новую ветвь). 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 (pull) и передачи своего репозитория на удалённый компьютер (push). 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 (push) завершается удачно только при условии отсутствия конфликтов. Если конфликты возникают, пользователь должен сначала слить версии в своём репозитории (выполнить pull), 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 (как техническими, так и административными мерами) желаемый порядок синхронизации. При этом каждый разработчик может вести работу независимо, так, как ему удобно, изменяя и сохраняя промежуточные версии документов, пользуясь всеми возможностями системы (в том числе доступом к истории изменений) даже в отсутствие сетевого соединения с сервером. Связь с сервером или другими разработчиками требуется исключительно для проведения синхронизации, при этом обмен наборами изменений может осуществляться по различным схемам.

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 (изменения файлов происходят на разных серверах, слияния и выделения ветвей происходят локально, об изменениях становится известно только при синхронизации, причём не всем разработчикам, а только тем, кто в данной синхронизации участвует).
- Единая сквозная нумерация версий системы и/или файлов, в которой номер версии монотонно возрастает (такая нумерация также требует наличия главного сервера, задающего номера версий для всех остальных). 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 (рабочего компьютера, домашнего компьютера, ноутбука и так далее). Использование распределённой системы избавляет от необходимости выделять один из компьютеров в качестве сервера, а синхронизация выполняется по необходимости, обычно при «пересадке» разработчика с одного устройства на другое.
- 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. Возможны и более сложные варианты — например, с созданием групп для работы по отдельным направлениям внутри более крупного проекта. При этом могут быть выделены отдельные «групповые» серверы для синхронизации работы групп, тогда процесс окончательного слияния изменения становится древовидным: сначала отдельные разработчики синхронизируют изменения на групповых серверах, затем обновлённые репозитории групп синхронизируются с главным сервером. Возможна работа и без «групповых» серверов, тогда разработчики одной группы синхронизируют изменения между собой, после чего любой из них (например, руководитель группы) передаёт изменения на центральный сервер.

В традиционной «офисной» разработке проектов, когда группа разработчиков относительно невелика и целиком находится на одной территории, в пределах единой локальной компьютерной сети, с постоянно доступными серверами, централизованная система может оказаться лучшим выбором из-за своей более жёсткой структуры и наличия функциональности, отсутствующей в распределённых системах (например, уже упомянутой блокировки). Возможность фиксировать изменения без их слияния в центральную ветвь в таких условиях легко реализуется путём выделения незавершённых работ в отдельные ветви разработки.

Проприетарные системы контроля версий

- Azure DevOps Server
- IBM Configuration Management Version Control (CMVC)
- IBM Rational MultiVersion File System
- Microsoft Visual SourceSafe
- Perforce
- QVCS
- Rational ClearCase
- Sun WorkShop TeamWare
- Vault (система управления версиями)
- Visual Studio Team System

Распределённые системы управления версиями

- Bazaar
- Darcs
- Fossil
- Git
- Mercurial
- TortoiseHg

Свободные системы управления версиями»

- Bazaar
- Bonsai CVS code management system
- CVS
- Darcs
- Fossil
- Git
- GitLab
- Mercurial
- Subversion
- TortoiseHg

Словарь

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.

amend
Внести изменения, не создавая новой версии — обычно когда разработчик ошибочно зафиксировал (*commit*) версию, но не залил (*push*) её на сервер.

blame
Понять, кто внёс изменение.

branch
Ветвь — 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. Документы в разных ветвях имеют одинаковую историю до точки ветвления и разные — после неё.

changeset, changelist, 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. В некоторых СУВ (*Subversion*) — новая версия автоматически переносится в хранилище документов.

check-out, clone
Извлечение документа из хранилища и создание рабочей копии.

conflict
Конфликт — ситуация, когда несколько пользователей сделали изменения одного и того же участка документа. Конфликт обнаруживается, когда один пользователь зафиксировал свои изменения, а второй пытается зафиксировать и система сама не может корректно слить конфликтующие изменения. Поскольку программа может быть недостаточно разумна для того, чтобы определить, какое изменение является «корректным», второму пользователю нужно самому разрешить конфликт (**resolve**).

graft, backport, cherry-picking, transplant
Использовать встроенный в СУВ алгоритм слияния, чтобы перенести отдельные изменения в другую ветвь, не сливая их. Например, исправили ошибку в экспериментальной ветви — вносим эти же изменения в стабильный ствол.

head
Основная версия — самая свежая версия для ветви/ствола, находящаяся в хранилище. Сколько ветвей, столько основных версий.

merge, integration
Слияние — 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.

pull, update
Получить новые версии из хранилища. В некоторых СУВ (*Subversion*) — происходит и *pull*, и *switch*, то есть загружаются изменения, а потом рабочая копия доводится до последнего состояния. **Будьте внимательны**, понятие *update* двусмысленно и в *Subversion* и *Mercurial* значит разное.

push
Залить новые версии в хранилище. Многие распределённые СУВ (*Git*, *Mercurial*) предполагают, что *commit* надо давать каждый раз, когда программист выполнил какую-то законченную

функцию. А залить — когда есть интернет и другие хотят ваши изменения. *Commit* обычно не требует ввода имени и пароля, а *push* — требует.

rebase

Использовать встроенный в СУВ алгоритм слияния, чтобы перенести точку ветвления (версию, от которой начинается ветвь) на более позднюю версию ствола. Чаще всего применяется в таком сценарии: Борис внёс изменения и обнаруживает, что не может залить (*push*) их, поскольку Анна ранее изменила совершенно другое место кода. Можно просто слить их (*merge*). Но дерево будет линейным и более читабельным, если отказаться от своей редакции, но внести те же изменения в редакцию Анны — это и есть *rebase*. Если Анна и Борис работают над одним и тем же местом кода, мешая друг другу и разрешая конфликты вручную, *rebase* не рекомендуется.

repository, depot

Хранилище документов — место, где система управления версиями хранит все документы вместе с историей их изменения и другой служебной информацией.

revision

Версия документа. Системы управления версиями различают версии по номерам, которые назначаются автоматически.

shelving

Откладывание изменений. Предоставляемая некоторыми системами возможность создать набор изменений (changeset) и сохранить его на сервере без фиксации (commit'а). Отложенный набор изменений доступен на чтение другим участникам проекта, но до специальной команды не входит в основную ветвь. Поддержка откладывания изменений даёт возможность пользователям сохранять незавершённые работы на сервере, не создавая для этого отдельных ветвей.

strip

Удалить целую ветвь из хранилища.

tag, label

Метка, которую можно присвоить определённой версии документа. Метка представляет собой символическое имя для группы документов, причём метка описывает не только набор имён файлов, но и версию каждого файла. Версии включённых в метку документов могут принадлежать разным моментам времени.

trunk, mainline, master

Ствол — основная ветвь разработки проекта. Политика работы со стволом может отличаться от проекта к проекту, но в целом она такова: большинство изменений вносится в ствол; если требуется серьёзное изменение, способное привести к нестабильности, создаётся ветвь, которая сливается со стволом, когда нововведение будет в достаточной мере испытано; перед выпуском очередной версии создаётся ветвь для последующего выпуска, в которую вносятся только исправления.

update, sync, switch

Синхронизация рабочей копии до некоторого заданного состояния хранилища. Чаще всего это действие означает обновление рабочей копии до самого свежего состояния хранилища. Однако при необходимости можно синхронизировать рабочую копию и к более старому состоянию, чем текущее.

working copy

Рабочая (локальная) копия документов.