Вопрос 4. Работа с системой контроля версий

1. Основные понятия

Система контроля версий *(Version Control System, VCS*) представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п.

Система контроля версий записывает изменения в файл или набор файлов в течение времени и позволяет вернуться позже к определённой версии. Контроль версий может применяться для работы с исходным кодом программного обеспечения, а также практически для любых типов файлов.

Наверное, всем знакома ситуация, когда при работе над проектом, возникает необходимость внести изменения, но при этом нужно сохранить работоспособный вариант. В таком случае, как правило, создается новая папка с дополнением в виде даты или небольшой пометки, в нее копируется рабочая версия проекта и уже с ним производится работа. Со временем количество таких папок может значительно возрасти, что создает трудности в вопросе отката на предыдущие версии, отслеживании изменений и т.п. Эта ситуация значительно ухудшается, когда над проектом работает несколько человек.

Для решения таких проблем как раз и используется система контроля версий, она позволяет комфортно работать над проектом как индивидуально, так в коллективе. *VCS* отслеживает изменения в файлах, предоставляет возможности для создания новых и слияние существующих ветвей проекта, производит контроль доступа пользователей к проекту, позволяет откатывать исправления и определять кто, когда и какие изменения вносил в проект.

Пример: *Если вы графический или web-дизайнер и хотите сохранить каждую версию изображения или макета (скорее всего, захотите), система контроля версий — как раз то, что нужно. Она позволяет вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда, и многое другое. Использование VCS также значит в целом, что, если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить. В дополнение ко всему вы получите всё это без каких-либо дополнительных усилий.*

Основным понятием *VCS* является **репозиторий** (*repository*) – специальное хранилище файлов и папок проекта, изменения в которых отслеживаются. В распоряжении разработчика имеется так называемая **“рабочая копия”** (*working copy*) проекта, с которой он непосредственно работает.

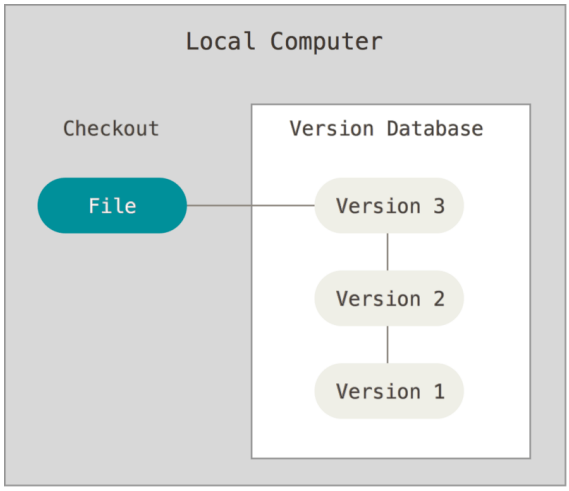
Рабочую копию необходимо периодически синхронизировать с репозиторием, эта операция предполагает отправку в него изменений, которые пользователь внес в свою рабочую копию (такая операция называется *commit*) и актуализацию рабочей копии, в процессе которой к пользователю загружается последняя версия из репозитория (этот процесс носит название *update*).

2. Виды систем контроля версий

1) Локальные системы контроля версий разработаны достаточно давно и представляют собой простую базу данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

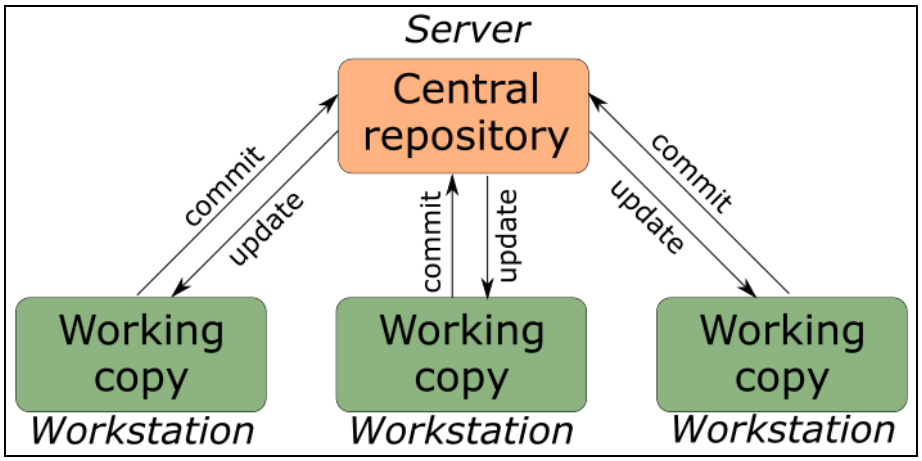
Одна из популярных локальных систем — система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различий

между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.



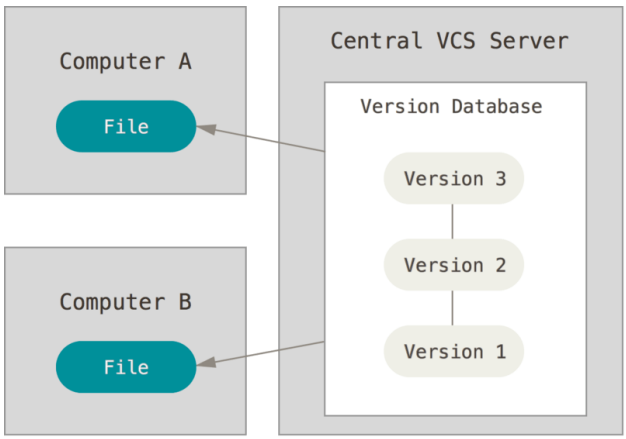
Многие люди в качестве метода контроля версий применяют копирование файлов в отдельный каталог (возможно даже, каталог с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко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

2) Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществлялся через специальное клиентское приложение.



Такие системы решает проблему необходимости взаимодействия с другими разработчиками. Они используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которым доступны любые файлы из этого централизованного хранилища.

Этот подход имеет множество преимуществ, особенно перед локальными системами контроля. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них, а администраторы имеют полный контроль над тем, кто и что делает.



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 сохранятся на локальных машинах разработчиков. Локальные системы контроля тоже страдают от этой проблемы: когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

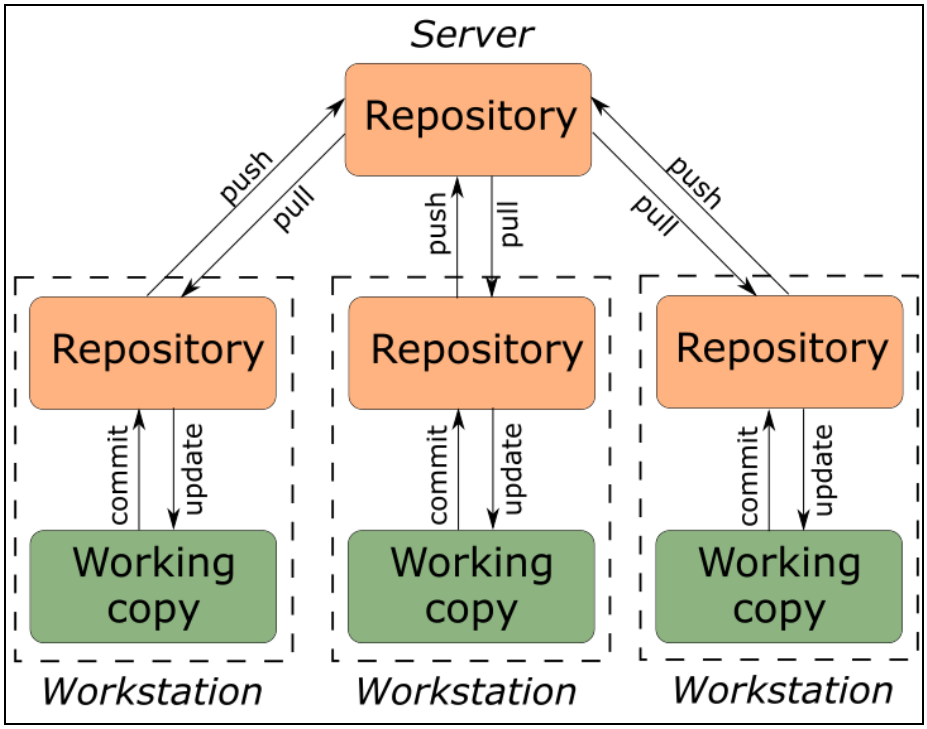
В качестве примеров таких программных продуктов можно привести: *CVS, Subversion, Perforce.*

CVS (*Concurrent Versions System,* Система одновременных версий) — одна из первых систем получивших широкое распространение среди разработчиков, она возникла в конце 80-х годов прошлого века. настоящее время этот продукт не развивается, это в первую очередь связано с рядом ключевых недостатков, таких как невозможность переименования файлов, неэффективное их хранение, практически полное отсутствие контроля целостности.

Subversion (*SVN*) – система контроля версий, созданная на замену *CVS*. *SVN* была разработана в 2004 году и до сих пор используется. Несмотря на многие преимущества по сравнению с *CVS* у *SVN* все-таки есть недостатки, такие как проблемы с переименованием, невозможность удаления данных из хранилища, проблемы в операции слияния ветвей и т.д. В целом *SVN* был (и остается) значительном шагом вперед по сравнению с *CVS*.

3) Распределенные системы контроля версий (*Distributed Version Control System, DVCS*) позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой. При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться. При работе с такой системой, пользователи периодически синхронизируют свои локальные репозитории с центральным и работают непосредственно со своей локальной копией. После внесения достаточного количества

изменений в локальную копию они (изменения) отправляются на сервер. При этом сервер, чаще всего, выбирается условно, т.к. в большинстве *DVCS* нет такого понятия как “выделенный сервер с центральным репозиторием”.



Большое преимущество такого подхода заключается в автономии разработчика при работе над проектом, гибкости общей системы и повышение надежности, благодаря тому, что каждый разработчик имеет локальную копию центрального репозитория.

Здесь 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 из таких систем 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.

Две наиболее известные *DVCS* – это *Git* и *Mercurial*.

Mercurial — свободная *DVCS,* которая построена таким образом, что в ней отсутствует понятие центрального репозитория, для работы с этой *VCS* используется (как правило) консольная утилита *hg*. *Mercurial* обладает всеми возможностями системы контроля версий, такими как ветвление, слияние, синхронизация с другими репозиториями. Данный проект используют и поддерживают большое количество крупных разработчиков, среди них *Mozilla, OpenOffice, OpenJDK* и многие другие. Сам продукт написан на языке *Python* и доступен на большинстве современных

операционных систем (*Windows, Mac OS, Linux*), также существует значительное количество утилит с графическим интерфейсом для работы с *Mercurial*.

Лидер этого рынка на сегодня, это *Git* – распределенная система контроля версий, разработанная Линусом Торвальдсем для работы над ядром операционной системы *Linux*. Среди крупных проектов, в рамках которых используется *git*, можно выделить ядро *Linux*, *Qt*, *Android*.

Git свободно распространяется под лицензией *GNU GPL* 2 и, также как *Mercurial*, доступен практически на всех операционных системах. По своим базовым возможностям *git* схож с *Mercurial* (и другими *DVCS*), но благодаря ряду достоинств (высокая скорость работы, возможность интеграции с другими *VCS*, удобный интерфейс) и очень активному сообществу, сформировавшемуся вокруг этой системы, *git* вышел в лидеры рынка распределенных систем контроля версий. Необходимо отметить, что несмотря на большую популярность таких систем как *git*, крупные корпорации, подобные *Google*, используют свои *VCS*.

3. Особенности работы в системах контроля версий

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.**

Предполагается, что проект, каким бы он ни был, уже существует и на сервере размещён его репозиторий, к которому разработчик получает доступ.

1) **Копии и дубли.**

Первое, что делает разработчик, это извлекает 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, такая 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 т.п.

2) **Постоянное обновление.**

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.

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.

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). 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.

4) **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. 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, но чаще всего, по выполнению работы, 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.

5) **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).

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.

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.

6) **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.

7) **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.

Разработчик, желающий изменить файл, вызывает специальную команду блокировки (**lock**) с указанием имени этого файла, сервер проверяет, не заблокирован ли уже файл другим разработчиком и помечает его флагом «заблокированный», с сохранением идентификатора разработчика и времени блокировки;

Разработчик работает с заблокированным файлом. Если в процессе работы выясняется, что файл изменять не нужно, он может вызвать команду снятия блокировки (**unlock, release lock**). По завершении работы с блокируемым файлом разработчик фиксирует изменения. Обычно блокировка при этом снимается автоматически,

8) **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 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.

Литература:

1. Система управления версиями: статья Википедии / https://dic.academic.ru/dic.nsf/ruwiki/44659
2. Введение – О системе контроля версий / https://git-scm.com/book/ru/v2/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B5-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D1%8F-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9