


Тема: Системы контроля версий. Что такое контроль версий и для чего он необходим. Системы контроля версий CVS, SVN, GIT, Mercurial.

Урок 24.



План :

1. Что такое контроль версий и для чего он необходим.
2. Классификация VCS.
3. Ежедневный цикл работы.
4. Ветвления.
5. Разнообразие систем контроля версий.



1. Что такое контроль версий и для чего он необходим.


Система управления версиями
(Version Control System, VCS)

— 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.



Например:

Представьте себе ситуацию. Вы наняли разработчика, чтобы он добавил к Вашему интернет-магазину, функцию быстрого заказа. Так как сайт все время должен функционировать и приносить доход, разработчик решает вести работы на своем локальном сервере. Пока он работает, дизайнер прислал новый логотип, который Вы тут же сами заменили в шаблоне на основной версии сайта. Заодно вы уменьшили шрифт в названиях товаров, чтобы все влезло на экран нетбука одного из ваших клиентов. Потом Вы обновили пару картинок для товаров. В это время разработчик решил сделать вам услугу - удалить в своей версии вашего сайта откровенно ненужный функционал, который писал предыдущий разработчик. Никто не контролирует изменения и уже никто не понимает где, чья версия. Но Вы оба думаете, что ничего серьезного не делали. Вы ошибаетесь. Разработчик, заливая свою версию сайта, стёр результаты Вашей работы, ваяются ошибки. В чем проблема непонятно. Чтобы не попасть в такую ситуацию, необходимо пользоваться системой контроля версий.



Системы контроля версий необходимы для решения следующих задач:

Архивация и восстановление - ведётся история изменения файлов с возможностью обновления до указанного состояния;

Ведение истории – при каждом изменении пользователи вносят комментарии, где описывают, для чего были внесены изменения;

Создание веток (альтернативные реализации) – VCS позволяет создавать разные варианты одного документа, так называемые ветки, с общей историей изменений до точки ветвления и с разными - после неё.


Использование VCS даёт нам:

- Полную уверенность в том, что файлы, которые мы получаем из системы, являются актуальными всегда, в любой момент времени.

- Возможность получить требуемую версию с любого компьютера, который позволит подключиться к серверу.

- Сохраняя файл в VCS, не нужно думать о том, что кто-то, работающий с этим же файлом, пересохранит и удалит изменения.

- Для разработчиков программных продуктов использование системы также позволяет производить принятие/отклонение изменений, сделанных одним из разработчиков.



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. Другими словами, VCS позволяет изменять одни и те же файлы нескольким разработчикам одновременно и без создания локальных копий на их компьютерах. При этом все варианты изменений сохраняются отдельно, и можно сделать разные варианты одного и того же файла с учетом разных правок от разных людей. Если же несколько изменений затрагивают один и тот же фрагмент документа, то система предложит выбрать нужный вариант. Обычно для работы с системой контроля версий используется отдельный компьютер (сервер) или интернет-сервис, предоставляющий возможность аренды подобного сервера.

Пример: Если над одним Excel документом работает несколько человек, то для редактирования файл доступен только одному человеку, остальные получают доступ “только на чтение”. С использованием VCS Вы получаете возможность редактирования файла сразу и всеми. Единственным условием является только то, что после внесения изменений, файл нужно сохранить на сервер, а не на локальный компьютер.



2. Классификация VCS.

Все системы контроля версий можно подразделить на :

Централизованные / распределенные

- в централизованных системах контроля версий вся работа производится с центральным репозиторием, в распределённых- у каждого разработчика есть локальная копия репозитория.

Блокирующие/не блокирующие

– блокирующие системы контроля версий позволяют наложить запрет на изменение файла, пока один из разработчиков работает над ним, в не блокирующих один файл может одновременно изменяться несколькими разработчиками.

Для текстовых данных/для бинарных данных

– для VCS для текстовых данных очень важна поддержка слияния изменений, для VCS с бинарными данными важна возможность блокировки.



3. Ежедневный цикл работы.

Обычный цикл работы разработчика в течение рабочего дня, допуская некоторые вариации, 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. Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версии, для чего разработчик выполняет операцию обновления рабочей копии (update) насколько возможно.

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.



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. 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.




5. Разнообразие систем контроля версий.

Системы контроля версий стали неотъемлемой частью жизни не только разработчиков программного обеспечения, но и всех тех, кто столкнулся с проблемой управления интенсивно изменяющейся информацией.

Вследствие этого, появилось большое число различных продуктов, предлагающих широкие возможности и предоставляющих обширные инструменты для управления версиями.


Рассмотрим достоинства и недостатки наиболее популярных систем контроля версий: CVS, SVN, GIT, Mercurial.



Система управления параллельными версиями (Concurrent Versions System)

– это одна из систем контроля версий. Одна из наиболее старых систем контроля версий, её разработка началась в 1986 году. Эта система хранит историю изменений определённого набора файлов, как правило, исходного кода программного обеспечения, и облегчает совместную работу группы людей работающих над одним проектом.

CVS основана на технологии клиент-сервер, взаимодействующих по сети. Клиент и сервер также могут располагаться на одной машине, если над проектом работает только один человек, или требуется вести локальный контроль версий.




Работа CVS организована следующим образом.

Последняя версия и все сделанные изменения хранятся в репозитории сервера. Клиенты, подключаясь к серверу, проверяют отличия локальной версии от последней версии, сохраненной в репозитории, и, если есть отличия, загружают их в свой локальный проект.

При необходимости решают конфликты и вносят требуемые изменения в разрабатываемый продукт. После этого все изменения загружаются в репозиторий сервера. CVS, при необходимости, позволяет выбрать нужную версию разрабатываемого проекта и вести управление несколькими проектами одновременно.

Ниже приведены основные достоинства и недостатки системы управления параллельными версиями.



Достоинства:

1. Несколько клиентов могут одновременно работать над одним и тем же проектом.
2. Позволяет управлять не одним файлом, а целыми проектами.
3. Обладает огромным количеством удобных графических интерфейсов.
4. Широко распространена и поставляется по умолчанию с большинством операционных систем Linux.
5. При загрузке тестовых файлов из репозитория передаются только изменения, а не весь файл целиком.

Недостатки:


1. При перемещении или переименовании файла или директории теряются все, привязанные к этому файлу или директории, изменения.
2. Сложности при ведении нескольких параллельных веток одного и того же проекта.
3. Ограниченная поддержка шрифтов.
4. Для каждого изменения бинарного файла сохраняется вся версия файла, а не только внесенное изменение.
5. С клиента на сервер измененный файл всегда передается полностью.
6. Ресурсоемкие операции, так как требуют частого обращения к репозиторию, и сохраняемые копии имеют некоторую избыточность.



Но это достоинства только по сравнению с более ранними системами контроля версий.

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.



Subversion(SVN) — свободная централизованная система управления версиями, официально выпущенная в 2004 году и была призвана заменить собой распространенную на тот момент систему CVS.

Subversion реализует все основные функции CVS и свободна от ряда недостатков последней (переименование и перемещение файлов и каталогов, работа с двоичными файлами и т.д.). Принцип работы с Subversion очень походит на работу с CVS. Клиенты копируют изменения из репозитория и объединяют их с локальным проектом пользователя. Если возникают конфликты локальных изменений и изменений, сохраненных в репозитории, то такие ситуации разрешаются вручную. Затем в локальный проект вносятся изменения, и полученный результат сохраняется в репозитории.

При работе с файлами, не позволяющими объединять изменения, может использоваться следующий принцип:

1. Файл скачивается из репозитория и блокируется (запрещается его скачивание из репозитория).
2. Вносятся необходимые изменения.
3. Загружается файл в репозиторий и разблокируется (разрешается его скачивание из репозитория другим клиентам).

Из-за своей широкой функциональности а также простоты и схожести в управлении с CVS, Subversion с успехом вытесняет CVS. Рассмотрим её достоинства



Достоинства:


1. Система команд, схожая с CVS.
2. Поддерживается большинство возможностей CVS.
3. Разнообразные графические интерфейсы и удобная работа из консоли.
4. Отслеживается история изменения файлов и каталогов даже после их переименования и перемещения.
5. Высокая эффективность работы, как с текстовыми, так и с бинарными файлами.
6. Встроенная поддержка во многие интегрированные средства разработки, такие как KDevelop, Zend Studio и многие другие.
7. Возможность создания зеркальных копий репозитория.
8. Два типа репозитория – база данных или набор обычных файлов.
9. Возможность доступа к репозиторию через Apache с использованием протокола WebDAV.
10. Наличие удобного механизма создания меток и ветвей проектов.
11. Можно с каждым файлом и директорией связать определенный набор свойств, облегчающий взаимодействие с системой контроля версии.
12. Широкое распространение позволяет быстро решить большинство возникающих проблем, обратившись к данным, накопленным Интернет-сообществом.



Недостатки:


1. Полная копия репозитория хранится на локальном компьютере в скрытых файлах, что требует достаточно большого объема памяти.
2. Существуют проблемы с переименованием файлов, если переименованный локально файл одним клиентом был в это же время изменен другим клиентом и загружен в репозиторий.
3. Слабо поддерживаются операции слияния веток проекта.
4. 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.

В целом Subversion – современная система контроля версий, обладающая широким набором инструментов, позволяющих удовлетворить любые нужды для управления версиями проекта с помощью централизованной системы контроля. В Интернете множество ресурсов посвящено особенностям Subversion, что позволяет быстро и качественно решать все возникающие в ходе работы проблемы. Простота установки, подготовки к работе и широкие возможности позволяют ставить Subversion на одну из лидирующих позиций среди разнообразия систем контроля версий.



Git – это гибкая, распределенная (без единого сервера) система контроля версий, дающая массу возможностей не только разработчикам программных продуктов, но и писателям для изменения, дополнения и отслеживания изменения «рукописей» и сюжетных линий, и учителям для корректировки и развития курса лекций, и администраторам для ведения документации, и для многих других направлений, требующих управления историей изменений. У каждого разработчика, использующего Git, есть свой локальный репозиторий, позволяющий локально управлять версиями. Затем, сохраненными в локальный репозиторий данными, можно обмениваться с другими пользователями. Часто при работе с Git создают центральный репозиторий, с которым остальные разработчики синхронизируются.

Пример организации системы с центральным репозиторием – это проект разработки ядра Linux (<http://www.kernel.org>). В этом случае все участники проекта ведут свои локальные разработки и беспрепятственно скачивают обновления из центрального репозитория. Когда необходимые работы отдельными участниками проекта выполнены и отлажены, они, после удостоверения владельцем центрального репозитория в корректности и актуальности проделанной работы, загружают свои изменения в центральный репозиторий. Наличие локальных репозиториями значительно повышает надежность хранения данных, так как, если один из репозитория выйдет из строя, данные могут быть легко восстановлены из других репозиториях. Работа над версиями проекта в Git может вестись в нескольких ветках, которые затем могут с легкостью полностью или частично объединяться, уничтожаться, откатываться и разрастаться во все новые и новые ветки проекта. Рассмотрим основные достоинства и недостатки этой системы управления версиями.



Достоинства:


1. Гибкая система ветвления проектов и слияния веток между собой.
2. Наличие локального репозитория, содержащего полную информацию обо всех изменениях, позволяет вести полноценный локальный контроль версий и заливать в главный репозиторий только полностью прошедшие проверку изменения.
3. Высокая производительность и скорость работы.
4. Удобный и интуитивно понятный набор команд.
5. Множество графических оболочек, позволяющих быстро и качественно вести работы с Git.
6. Возможность делать контрольные точки, в которых данные сохраняются без дельта компрессии, а полностью. Это позволяет уменьшить скорость восстановления данных, так как за основу берется ближайшая контрольная точка, и восстановление идет от нее. Если бы контрольные точки отсутствовали, то восстановление больших проектов могло бы занимать часы.
7. Широкая распространенность, легкая доступность и качественная документация.
8. Гибкость системы позволяет удобно ее настраивать.
9. Универсальный сетевой доступ с использованием протоколов http, ftp, rsync, ssh и др.



Недостатки:

1. Unix – ориентированность. На данный момент отсутствует зрелая реализация Git, совместимая с другими операционными системами.
2. Использование для идентификации ревизий хешей SHA1, что приводит к необходимости оперировать длинными строками вместо коротких номеров версий, как во многих других системах (хотя в командах допускается использование неполных хеш-строк).
3. Не отслеживается изменение отдельных файлов, а только всего проекта целиком, что может быть неудобно при работе с большими проектами, содержащими множество несвязных файлов.
4. При начальном создании репозитория и синхронизации его с другими разработчиками, потребуется достаточно длительное время для скачивания данных, особенно, если проект большой, так как требуется скопировать на локальный компьютер весь репозиторий.

Git – гибкая, удобная и мощная система контроля версий, способная удовлетворить большинство пользователей. Существующие недостатки постепенно удаляются и не приносят серьезных проблем пользователям. Если ваша команда работает над большим проектом, территориально удалена, и тем более, если часто приходится разрабатывать программное обеспечение, не имея доступа к другим разработчикам (например, вы не хотите терять время при перелете из страны в страну), можно делать любые изменения и сохранять их в локальном репозитории, откатываться, переключаться между ветками и т.



Mercurial - кроссплатформенная распределённая система управления версиями, разработанная для эффективной работы с очень большими репозиториями кода. Распределенная система контроля версий Mercurial разрабатывалась параллельно с системой контроля версий Git. Первоначально, она была создана для эффективного управления большими проектами под Linux, а поэтому была ориентирована на быструю и надежную работу с большими репозиториями. На данный момент mercurial адаптирован для работы под Windows, Mac OS X и большинство Unix систем.

Большая часть системы контроля версий написана на языке Python, и только отдельные участки программы, требующие наибольшего быстродействия, написаны на языке C. Mercurial поддерживает полностью децентрализованную работу (отсутствует понятие основного хранилища кода), ветвление (возможно вести несколько веток одного проекта и копировать изменения между ветками), слияние репозиториев (чем и достигается «распределённость» работы). Идентификация ревизий происходит на основе алгоритма хеширования SHA1 (Secure Hash Algorithm 1), однако, также предусмотрена возможность присвоения ревизиям индивидуальных номеров. Для взаимодействия между клиентами используются протоколы HTTP, HTTPS или SSH.

Набор команд - простой и интуитивно понятный, во многом схожий с командами subversion. Так же имеется ряд графических оболочек и доступ к репозиторию через веб-интерфейс. Немаловажным является и наличие утилит, позволяющих импортировать репозитории многих других систем контроля версий. Рассмотрим основные достоинства и недостатки Mercurial.




Достоинства:

1. Быстрая обработка данных.
2. Кроссплатформенная поддержка.
3. Возможность работы с несколькими ветками проекта.
4. Простота в обращении.
5. Возможность конвертирования репозитория в других системах поддержки версий, таких как CVS, Subversion, Git, Bazaar и др.

Недостатки:

1. Возможные (но чрезвычайно низкие) совпадения хеш - кода отличных по содержанию ревизий.
2. Ориентирован на работу в консоли.

Простой и отточенный интерфейс, и набор команд, возможность импортировать репозитории с других систем контроля версий, позволяют перейти на Mercurial и быстро обучиться основным его особенностям. Это не занимает больше нескольких дней. Надежность и скорость работы позволяют использовать его для контроля версий огромных проектов. Все это делает Mercurial достойным конкурентом Git.



Большой выбор систем контроля версий позволяет удовлетворить любые требования и организовать работу так, как вам необходимо. Однако, среди всего многообразия систем есть явные лидеры.

Так, если необходимо управлять огромным проектом, состоящим из десятков тысяч файлов и над которым работу ведут тысячи человек, то лучше всего выбор остановить на Git или Mercurial.

Для программистов одиночек или небольших проектов, не требующих ветвления и создания множества версий, лучше всего подойдет Subversion.