Московский Авиационный Институт

(Национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Реферат**

**по курсу «Сетевое программирование»**

**на тему**

**«Система контроля версий Git»**

X семестр

Студент: Рябов В.В.

группа 08-508

Москва, 2015

**Оглавление**

Понятие «системы контроля версий» 3

Распределённые системы управления версиями 5

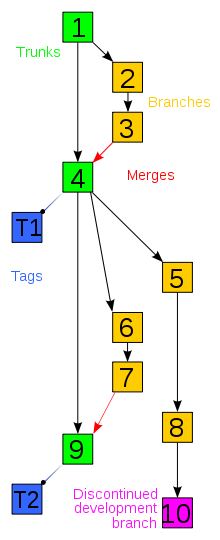
**Система управления версиями Git 6**

Выводы 10

Использованные материалы 10

**Понятие «системы контроля версий»**

Система управления версиями (Version Control System или Revision Control System) представляют собой программное обеспечение для облегчения деятельности с быстро меняющейся информацией. Система контроля версий предоставляет возможность хранить несколько вариантов одного и того же документа. При необходимости можно вернуться к старым версиям, можно узнать, кем были сделаны те или иные изменения и т.д. Такого рода системы в большинстве своем используются при разработке программного обеспечения, чтобы можно было хранить исходные коды разрабатываемых программ. Система контроля версий 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.

Внутри репозитория могут быть несколько параллельных линий разработки, обычно называемых ветвями. Это может быть полезно для хранения стабильной или выпущенной версии ветви, одновременно продолжая работу над рабочей версией. Другой вариант – это открыть выделенную ветвь для работы над экспериментальной возможностью.

Система контроля версий также позволяет пользователям дать ярлык снимку ветви (часто называемых как тэги) для легкого доставания. Это полезно для обозначения индивидуальных релизов или самых свежих рабочих версий, предназначенных для использования.

Использование системы контроля версий безусловно обязательно для разработчика, если проект больше нескольких сот строк или если для проекта совместно работают несколько разработчиков. Использование хорошей системы контроля версий определенно лучше, нежели использование узконаправленных специальных методов, которые используют некоторые разработчики для управления различными ревизиями своего кода.

Большинство систем управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из репозитория; обычно создаётся локальная копия документа, т. н. «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 репозиторий. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в репозитории 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.

Очень часто над одним и тем же проектом трудится несколько человек. Если один из них будет изменять исходный файл, и одновременно с этим другой человек будет выполнять аналогичную операцию, то возможна такая ситуация, что какие-то изменения могут не сохраниться. Системы контроля версий работают с такого рода проблемами и имеют определенный перечень их решения. В большинстве своем эти системы могут автоматически объединять такого рода изменения, которые делают разные члены команды разработчиков. Но стоит отметить, что такого рода объединения чаще всего выполняется для текстовых файлов и с определенным условием: изменения происходили в разных частях файла. Данное ограничение имеет место, поскольку в большинстве своем системы контроля версий направлены на поддержку процесса разработки программных продуктов, а изначальные коды находятся в текстовых файлах. В случае если автоматически выполнить объединение не получилось, то система предлагает исправить ситуацию вручную.

Зачастую осуществить объединение невозможно ни с помощью системы, ни вручную. Ярким примером является ситуация, когда формат файла очень сложный или неизвестен. Отдельные системы контроля версий предоставляют возможность пользователю заблокировать файл в репозитории. Данная операция не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла 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.

**Распределённые системы управления версиями**

Существуют системы управления версиями, которые, вместо традиционной клиент-серверной, используют распределённую модель. Такие системы, в общем случае, не нуждаются в централизованном репозитории: вся история изменения документов хранится на каждом компьютере. Фактически, каждый компьютер, помимо рабочей копии, хранит локальную копию всего репозитория. В некоторых системах рабочая копия сама является репозиторием.  
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. По мере внесения изменений, копии, принадлежащие разным разработчикам, начинают различаться и возникает необходимость в их синхронизации. Такая синхронизация может осуществляться с помощью обмена патчами или так называемыми наборами изменений между пользователями.

Описанная модель аналогична созданию отдельной ветки для каждого разработчика в классической системе управления версиями (в некоторых распределённых системах перед началом работы с локальным репозиторием нужно создать новую ветвь). Пока разработчик изменяет только свою ветвь, его работа не влияет на других участников проекта и наоборот. Однако при необходимости выполнить слияние ветвей (или синхронизацию локальных репозиториев в распределённой модели) могут возникнуть конфликты.  
Основное преимущество распределённых систем заключается в их гибкости. Каждый разработчик может вести работу независимо, так, как ему удобно, сохраняя промежуточные варианты документов и передавая результаты другим участникам, когда посчитает нужным. При этом обмен наборами изменений может осуществляться по различным схемам. В небольших коллективах участники работы могут обмениваться изменениями по принципу «каждый с каждым», за счет чего отпадает необходимость в создании выделенного сервера. Крупное сообщество, наоборот, 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. Распределенная система контроля версий позволяет клонировать удаленный репозиторий, производя точную копию. Она также позволяет распространять изменения из одного репозитория на другой. В нераспределенных VCS разработчику нужен репозиторий для того, чтобы зафиксировать изменения в нем. Это делает разработчика без доступа к репозиторию человеком второго сорта. С распределенной VCS каждый разработчик может клонировать главный репозиторий, поработать над ним и потом распространить свои изменения на главный репозиторий.

**Система управления версиями Git**

Разрабатывать проект масштаба Linux без мощной и надежной системы контроля версий – невозможно. Одним из кандидатов и наиболее подходящим проектом оказалась система контроля версий Monotone, но Линуса Торвальдса не устроила ее скорость работы. Так как особенности организации СКВ Monotone не позволяли значительно увеличить скорость обработки данных, то 3 апреля 2005 года Торвальдс приступил к разработке собственной системы контроля версий – Git.

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’a. В этом случае все участники проекта ведут свои локальны 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 репозиториев также значительно повышает надежность хранения данных, так как, если один из репозиториев выйдет из строя, данные могут быть легко восстановлены из других репозиториев.

В Git файлы могут находиться в одном из трёх состояний: зафиксированном, изменённом и подготовленном. "Зафиксированный" значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

Таким образом, в проектах, использующих Git, есть три части: каталог Git (Git directory), рабочий каталог (working directory) и область подготовленных файлов (staging area).

Каталог Git — это место, где Git хранит метаданные и базу данных объектов вашего проекта. Это наиболее важная часть Git, и именно она копируется, когда вы клонируете репозиторий с другого компьютера.

Рабочий каталог — это извлечённая из базы копия определённой версии проекта. Эти файлы достаются из сжатой базы данных в каталоге Git и помещаются на диск для того, чтобы вы их просматривали и редактировали.

Область подготовленных файлов — это обычный файл, обычно хранящийся в каталоге Git, который содержит информацию о том, что должно войти в следующий коммит. Иногда его называют индексом (index), но в последнее время становится стандартом называть его областью подготовленных файлов (staging area).

Стандартный рабочий процесс с использованием Git выглядит примерно так:

1. Вы вносите изменения в файлы в своём рабочем каталоге.
2. Подготавливаете файлы, добавляя их слепки в область подготовленных файлов.
3. Делаете коммит, который берёт подготовленные файлы из индекса и помещает их в каталог Git'а на постоянное хранение.

Если рабочая версия файла совпадает с версией в каталоге Git, файл считается зафиксированным. Если файл изменён, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из БД, но не был подготовлен, то он считается изменённым. В главе 2 вы узнаете больше об этих трёх состояниях и как можно либо воспользоваться этим, либо пропустить стадию подготовки.

Для обобщения итогов приведем основные достоинства и недостатки СКВ Git.

Достоинства:  
⎯ Надежная система сравнения ревизий и проверки корректности данных, основанные на алгоритме хеширования SHA1 (Secure Hash Algorithm 1).  
⎯ Гибкая система ветвления проектов и слияния веток между собой.  
⎯ 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.  
⎯ Высокая производительность и скорость работы.  
⎯ Удобный и интуитивно понятный набор команд.  
⎯ Множество графических оболочек, позволяющих быстро и качественно вести работы с Git’ом.  
⎯ 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.  
⎯ Широкая распространенность, легкая доступность и качественная документация.  
⎯ Гибкость системы позволяет удобно ее настраивать и даже создавать специализированные контроля системы или пользовательские интерфейсы на базе Git.  
⎯ Универсальный сетевой доступ с использованием протоколов http, ftp, rsync, SSH и др.

Недостатки:  
⎯ Unix – ориентированность.  
⎯ Возможные (но чрезвычайно низкие) совпадения хеш-кода отличных по содержанию ревизий.  
⎯ Не отслеживается изменение отдельных файлов, а только всего проекта целиком, что может быть неудобно при работе с большими проектами, содержащими множество несвязных файлов.  
⎯ При начальном (первом) 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.

**Выводы**

Git – гибкая, удобная и мощная система контроля версий. Если вы ведете большой проект, территориально удаленный, и тем более, если часто приходится разрабатывать программное обеспечение, не имея доступа к другим разработчикам, можно делать любые изменения и сохранять их в локальном репозитории, откатываться, переключаться между ветками и т.д. Git – один из лидеров систем контроля версий.

**Использованные материалы**

материалы сайта http://git-scm.com

материалы из статьи http://en.wikipedia.org/wiki/Version\_control

материалы из статьи http://en.wikipedia.org/wiki/Git\_(software)

http://guides.beanstalkapp.com/version-control/intro-to-version-control.html

http://biz30.timedoctor.com/git-mecurial-and-cvs-comparison-of-svn-software/