**GIT**

**ОСНОВЫ КОНТРОЛЯ ВЕРСИЙ**

Система контроля версий (СКВ) (Version Control System или Revision Control System) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Обычно СКВ используются для контроля версий исходных кодов программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости, возвращаться к более ранним его версиям, определять, кто и когда сделал то или иное изменение и др.

СКВ можно разделить на следующие виды:

* Локальные системы контроля версий (ЛСКВ)
* Централизованные системы контроля версий (ЦСКВ) например, CVS, Subversion и Perforce;
* [Распределённые системы контроля версий](http://git-scm.com/book/ru/%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9E-%D0%BA%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%B5-%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D0%B9#%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B-%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) (РСКВ) например, Git, Mercurial, Bazaar или Darcs.

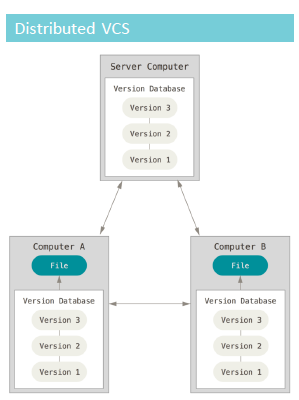
ЦСКВ – есть центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые получают копии файлов из него. Много лет это было стандартом для систем контроля версий

Однако при таком подходе есть и несколько серьёзных недостатков. Наиболее очевидный — централизованный сервер является уязвимым местом всей системы. Если сервер выключается на час, то в течение часа разработчики не могут взаимодействовать, и никто не может сохранить новой версии своей работы.

Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

РСКВ – клиенты не просто выгружают последние версии файлов, а полностью копируют весь репозиторий. Поэтому в случае, когда "умирает" сервер, через который шла работа, любой клиентский репозиторий может быть скопирован обратно на сервер, чтобы восстановить базу данных. Каждый раз, когда клиент забирает свежую версию файлов, он создаёт себе полную копию всех данных.





**Основные понятия**

**Репозиторий (repository)** – центральное хранилище, которое содержит версии файлов. Очень часто репозиторий организуется средствами какой-нибудь СУБД.

**Версия файла (revision)** – состояние файла в определенный момент времени. Репозиторий предоставляет возможность хранить неограниченное число версий одного и того же файла.

**Актуальная версия файла** – обычно это самая последняя версия файла, размещенного в репозитории.

**Рабочая версия файла (workingcopy)** – версия файла, с которой в текущий момент ведется работа, и которая не загружена в репозиторий.

**Загрузка (Upload)** – размещение файла в репозитории. В процессе загрузки в репозиторий помещается рабочая версия файла.

**Выгрузка (Checkout)** – получение файла из репозитория. В процессе выгрузки осуществляется получение из репозитория необходимой версии файла.

**Ветвь (branch)** — 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.

****

**ОСНОВЫ GIT**

**1.1 Слепки файловой системы**

Основным отличием Git от любых других систем управления версиями (например, Subversion и ей подобных) является то, каким образом в Git организованы данные. Большинство других систем управления версиями хранит данные в виде списка изменений (патчей) для файлов. Такие системы (CVS, Subversion, Perforce, Bazaar и другие) представляют хранимые данные в виде набора файлов и изменений, сделанных для каждого из этих файлов во времени (рисунок 1.1).



Рисунок 1.1

Вместо того чтобы хранить данные в виде, представленном на рисунке 1.1, Git представляет хранимые данные в виде набора ***слепков*** небольшой файловой системы. Каждый раз, когда пользователь фиксирует текущую версию проекта, система управления версиями Git сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Для повышения эффективности, в случае, если файл не был изменен, Git не сохраняет файл снова, а создает ссылку на сохраненный ранее файл. Данный подход изображен на рисунке 1.2.



Рисунок 1.2

Данная особенность отличает Git практически от всех других систем управления версиями. Вследствие чего, создание Git потребовало пересмотра практические всех аспектов управления версиями, которые другие системы переняли от своих предшественниц. Таким образом, Git напоминает небольшую файловую систему с мощными инструментами, работающими поверх нее, чем просто систему управления версиями.

**1.2 Локальные операции**

Для большинства операций в системе Git необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не требуется. Поскольку вся история проекта хранится локально на диске пользовательского компьютера, большинство операций выполняются практически мгновенно.

Для демонстрации истории проекта Git не загружает ее с сервера, а просто читает ее напрямую из локального репозитория конкретного пользователя, который запросил демонстрацию истории проекта. Если необходимо просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может вычислить разницу локально, вместо того, чтобы запрашивать разницу у сервера системы управления версиями или загружать старую версию файла и только затем осуществлять локальное сравнение.

Локальное выполнение операций означает, что лишь малую часть операций нельзя выполнить без доступа к сети или VPN. В случае если пользователь хочет поработать, не имея доступа к сети, например, находясь в самолете или поезде, он может продолжать делать ***коммиты*** (фиксировать изменения проекта), а затем отправить их на сервер, как только станет доступна сеть. Аналогично, если VPN-клиент не работает, все равно можно продолжать работу.

Во многих других системах управления версиями полноценная локальная работа невозможна или крайне неудобна. Например, используя Perforce, практически ничего нельзя сделать без соединения с сервером. Работая с Subversion и CVS, пользователь может редактировать файлы, но сохранить изменения в локальную базу данных невозможно, поскольку она отключена от центрального репозитория.

**1.3 Контроль целостности данных**

Перед тем, как любой файл будет сохранен, Git вычисляет его контрольную сумму, которая используется в качестве ***индекса*** данного файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы изменения не были обнаружены системой Git. Данная функциональность является важной составляющей Git. Если информация будет потеряна или повреждена при передаче, Git всегда это выявит.

Механизм, который используется в Git для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит приблизительно следующим образом:

24b9da6552252987aa493b52f8696cd6d3b00373

При работе с Git, такие хеши встречаются повсюду, поскольку они очень широко используются в системе Git. Фактически, в своей базе данных Git сохраняет все не по именам файлов, а по хешам их содержимого.

**1.4 Данные только добавляются**

Практические все действия, совершаемые пользователем в Git, только ***добавляют*** данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой системе управления версиями, потерять данные, которые еще не были сохранены, но как только они будут зафиксированы, их очень сложно потерять, особенно если изменения регулярно отправляются в центральный репозиторий. Поэтому, при использовании системы Git, можно экспериментировать, не боясь что-то серьезно поломать в проекте.

## 

В git файлы могут находиться в одном из трех состояний:

1) «новый» (Untracked) – файл только что создан;

2) «зафиксированный» (Committed) – файл уже сохранен в локальном репозитории;

3) «измененный» (Modified)– файл, который был изменен, но еще не был зафиксирован;

4) «подготовленный» (Staged) – измененный файл, отмеченный для включения в следующий коммит.

Таким образом, в проектах, использующих Git, есть три области (рисунок 1.3):

1) каталог Git (Gitdirectory) – место, где Git хранит метаданные и базу данных объектов пользовательского проекта; это наиболее важная часть Git и именно она копируется, когда выполняется ***клонирование*** репозитория с сервера;

2) рабочий каталог (workingdirectory) – извлеченная из базы копия определенной версии проекта; эти файлы извлекаются из сжатой базы данных в каталоге Git и помещаются на диск для того, чтобы их можно было просматривать и редактировать;

3) область подготовленных файлов (stagingarea) – это файл, обычно хранящийся в каталоге Git, который содержит информацию о том, что должно войти в следующий коммит; иногда его называют индексом.

Стандартный рабочий процесс с использованием системы управления версиями Git выглядит примерно следующим образом (рисунок 1.3):



Рисунок 1.3

1. Пользователь вносит изменения в файлы в своем рабочем каталоге.

2. Пользователь подготавливает файлы, добавляя их слепки в область подготовленных файлов.

3. Пользователь делает коммит, который берет подготовленные файлы из области подготовленных фалов (индекса) и помещает их в каталог Git на постоянное хранение.

Если рабочая версия файла совпадает с версией в каталоге Git, файл считается зафиксированным. Если файл изменен, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из базы, но не был подготовлен, то он считается измененным.

Когда определен репозиторий Git и рабочая копия файлов для проекта,то после этого необходимо делать некоторые изменения и фиксировать ***снимки*** (или слепки) состояния (snapshots) этих изменений в репозитории каждый раз, когда проект достигает состояния, которое бы хотелось сохранить.

Необходимо помнить, что каждый файл в рабочем каталоге может находиться в одном из двух состояний:

1) «отслеживаемый» – файл, который был в последнем снимке состояния проекта (находящийся под версионным контролем); он может быть неизмененным, измененным или подготовленным к коммиту;

2) «неотслеживаемый» – любой файл в рабочем каталоге, который не входил в последний снимок состояния и не подготовлен к коммиту.

Когда репозиторий клонирован, все файлы являются отслеживаемыми и неизмененными, потому что они только были клонированы (checkedout), но не были отредактированы.

Как только файлы будут отредактированы, Git будет рассматривать их как измененные. Изменения необходимо индексировать (подготавливать к коммиту) и затем фиксировать, после чего цикл повторяется. Данный жизненный цикл изображен на рисунке.

