Git

— мощная и сложная распределенная система контроля версий. Понимание всех возможностей git открывает для разработчика новые горизонты в управлении исходным кодом. Самый верный способ обучиться владению git — испытать его своими руками.
git (произн. «гит») — распределённая система управления версиями файлов. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux. На сегодняшний день поддерживается Джунио Хамано.

Система спроектирована как набор программ, специально разработанных с учётом их использования в скриптах. Это позволяет удобно создавать специализированные системы контроля версий на базе git или пользовательские интерфейсы. Например, Cogito является именно таким примером фронтенда к репозиториям Git, а StGit использует git для управления коллекцией патчей.

git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, SVK, Bazaar и Monotone, git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям git обеспечивается git-daemon, gitosis, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использование существующих конфигураций сетевых фильтров.

Возможности

Система спроектирована как набор программ, специально разработанных с учётом их использования в скриптах. Это позволяет удобно создавать специализированные системы контроля версий на базе git или пользовательские интерфейсы. Например, Cogito является именно таким примером фронтенда к репозиториям Git, а StGit использует git для управления коллекцией патчей.

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям git обеспечивается git-daemon, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Особенности, преимущества и недостатки

Преимущества и недостатки git по сравнению с централизованными системами управления версиями (такими как, например, Subversion) типичны для любой распределённой системы и описаны в статье «Система управления версиями». Если же сравнивать git с «родственными» ей распределёнными системами, можно отметить, что git изначально идеологически ориентирован на работу с изменениями, а не с файлами, «единицей обработки» для него является набор изменений, или патч. Эта особенность прослеживается как в структуре самой системы (в частности — в структуре репозитория), так и в принципах построения команд; она отражается на производительности системы в различных вариантах её использования и на достоинствах и недостатках git по сравнению с другими DVCS.

Часто называемые преимущества git перед другими DVCS:

- Высокая производительность.
- Развитые средства интеграции с другими VCS, в частности, с CVS, SVN и Mercurial. Помимо разнонаправленных конвертеров репозиториев, имеющиеся в комплекте программные средства позволяют разработчикам использовать git при размещении центрального репозитория в SVN или CVS, кроме того, git может имитировать cvs-сервер, обеспечивая работу через клиентские приложения и поддержку в средах разработки, специально не поддерживающих git.
- Продуманная система команд, позволяющая удобно встраивать git в скрипты.
- Качественный веб-интерфейс «из коробки».
- Репозитории git могут распространяться и обновляться общесистемными файловыми утилитами архивации и обновления, такими как rsync, благодаря тому, что фиксации изменений и синхронизации не меняют существующие файлы с данными, а только добавляют новые (за исключением некоторых служебных файлов, которые могут быть автоматически обновлены с помощью имеющихся в составе системы утилит). Для раздачи репозитория по сети достаточно любого веб-сервера.

В числе недостатков git обычно называют:

- Некоторое неудобство для пользователей, переходящих с других VCS. Команды git, ориентированные на наборы изменений, а не на файлы, могут вызвать недоумение у пользователей, привыкших к файл-ориентированным VCS, таким как SVN. Например, команда «add», которая в большинстве систем управления версиями производит добавление файла к проекту, в git делает совершенно другое: она находит и подготавливает к фиксации (commit'у) сделанные в проекте или его части изменения (то есть её название «добавить» относится не к файлам, а к сделанным

изменениям, которые добавляются в индекс для последующей фиксации).
- Использование для идентификации ревизий хэшей SHA1, что приводит к необходимости оперировать длинными строками вместо коротких номеров версий, как во многих других системах.
- Бóльшие накладные расходы при работе с проектами, в которых делаются многочисленные несвязанные между собой изменения файлов. При работе в таком режиме размеры наборов изменений становятся достаточно велики и происходит быстрый рост репозиториев.
- Бóльшие затраты времени, по сравнению с файл-ориентированными системами, на формирование истории конкретного файла, истории правок конкретного пользователя, поиска изменений, относящихся к заданному месту определённого файла.
- Отсутствие отдельной команды переименования/переноса файла, из-за чего подобная операция отражается в истории как удаление файла и создание его в другом месте, что требует специального анализа для определения, что в действительности файл был просто перенесён.
- Система не умеет отслеживать пустые каталоги.
- Некоторые команды работают неожиданно, в частности, могут приводить к неочевидным ошибкам или требовать для правильной работы указания специальных параметров, когда применяются к исходно пустому репозиторию или к репозиторию, в котором ещё не было сделано ни одного коммита.

В ряде публикаций, относящихся преимущественно к 2005—2008 годам можно встретить также нарекания в отношении документации git, отсутствия удобной windows-версии и удобных графических клиентов. В настоящее время эта критика неактуальна: существует версия git на основе MinGW («родная» сборка под Windows), и несколько высококачественных графических клиентов для различных операционных систем, в частности, под Windows имеется клиент TortoiseGit, идеологически очень близкий к широко распространённому TortoiseSVN — клиенту SVN, встраиваемому в оболочку Windows.

Слепки вместо патчей Главное отличие git от любых других СУВ (например, Subversion и ей подобных) это то, как git смотрит на данные. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion, Perforce, Bazaar и другие) относятся к хранимым данным как к набору файлов и изменений сделанных для каждого из этих файлов во времени, как показано на рисунке 1-4. Рисунок 1-4. Другие системы хранят данные как изменения к базовой версии для каждого файла. git не хранит свои данные в таком виде. Вместо этого git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл. То, как git подходит к хранению данных, похоже на рисунок 1-5.
Рисунок 1-5. git хранит данные как слепки состояний проекта во времени.

Это важное отличие git от практически всех других систем управления версиями. Из-за него git вынужден пересмотреть практически все аспекты управления версиями, которые другие системы взяли от своих предшественниц. git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх неё, чем на просто СУВ. В главе 3, коснувшись работы с ветвями в Git, мы узнаем, какие преимущества даёт такое понимание данных. Почти все операции – локальные

Для совершения большинства операций в git необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не нужна. Если вы пользовались централизованными системами, где практически на каждую операцию накладывается сетевая задержка, вы, возможно, подумаете, что боги наделили git неземной силой. Поскольку вся история проекта хранится локально у вас на диске, большинство операций выглядят практически мгновенными. К примеру, чтобы показать историю проекта, Git-у не нужно скачивать её с сервера, он просто читает её прямо из вашего локального репозитория. Поэтому историю вы увидите практически мгновенно. Если вам нужно просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, git может взять файл месячной давности и вычислить разницу на месте, вместо того чтобы запрашивать разницу у сервера СУВ или качать с него старую версию файла и делать локальное сравнение. Кроме того, работа локально означает, что мало чего нельзя сделать без доступа к Сети или VPN. Если вы в самолёте или в поезде и хотите немного поработать, можно спокойно делать коммиты а затем отправить их, как только станет доступна сеть. Если вы пришли домой, а VPN клиент не работает, всё равно можно продолжать работать. Во многих других системах это невозможно, или же крайне неудобно. Например, используя Perforce, вы мало что можете сделать без соединения с сервером. Работая с Subversion и CVS, вы можете редактировать файлы, но сохранить изменения в вашу базу данных нельзя (потому что она отключена от репозитория). Вроде ничего серьёзного, но потом вы удивитесь, насколько это меняет дело. git следит за целостностью данных

Перед сохранением любого файла git вычисляет контрольную сумму, и она становится индексом этого файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы git не узнал об этом. Эта функциональность встроена в сам фундамент git и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, git всегда это выявит. Механизм, используемый git для вычисления контрольных сумм, называется SHA-1 хеш. Это строка из 40 шестнадцатеричных знаков (0-9 и a-f), которая вычисляется на основе содержимого файла или структуры каталога, хранимого Git. SHA-1 хеш выглядит примерно так: 24b9da6552252987aa493b52f8696cd6d3b00373 Работая с Git, вы будете постоянно встречать эти хеши, поскольку они широко используются. Фактически, в своей базе данных git сохраняет всё не по именам файлов, а по хешам их содержимого. Чаще всего данные в git только добавляются

Практически все действия, которые вы совершаете в Git, только добавляют данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой СУВ, потерять данные, которые вы ещё не сохранили, но как только они зафиксированы, их очень сложно потерять, особенно если вы регулярно отправляете изменения в другой репозиторий.