Git

О контроле версий

Что такое контроль версий, и зачем он вам нужен? Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Под версионный контроль можно поместить файлы практически любого типа.

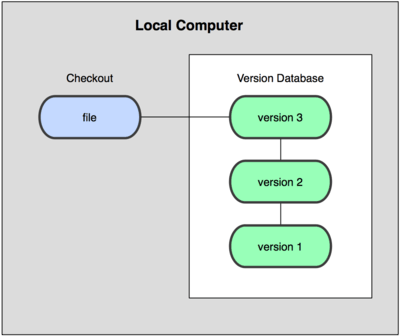
Если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить.

Локальные системы контроля версий

Многие предпочитают контролировать версии, просто копируя файлы в другой каталог. Такой подход очень распространён, потому что прост, но он и чаще даёт сбои. Очень легко забыть, что ты не в том каталоге, и случайно изменить не тот файл, либо скопировать файлы не туда, куда хотел, и затереть нужные файлы.

Чтобы решить эту проблему, программисты уже давно разработали локальные СКВ с простой базой данных, в которой хранятся все изменения нужных файлов.

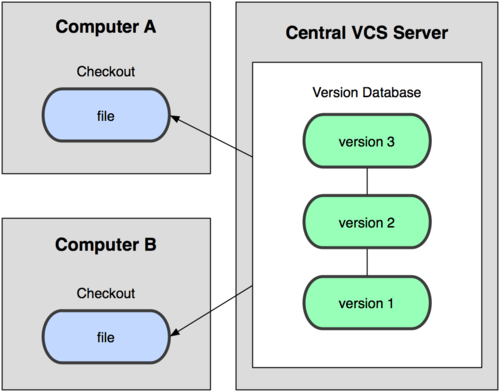
Пример: rcs



Централизованные системы контроля версий

Следующей основной проблемой оказалась необходимость сотрудничать с разработчиками за другими компьютерами. Чтобы решить её, были созданы централизованные системы контроля версий (ЦСКВ).

В таких системах, например, CVS или Subversion, 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.



Такой подход имеет множество преимуществ, особенно над локальными СКВ. К примеру, все знают, кто и чем занимается в проекте. У администраторов есть чёткий контроль над тем, кто и что может делать, и, конечно, администрировать ЦСКВ намного легче, чем локальные базы на каждом клиенте.

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. Если же повреждается диск с центральной базой данных и нет резервной копии, вы теряете абсолютно всё — всю историю проекта, разве что за исключением нескольких рабочих версий, сохранившихся на рабочих машинах пользователей. Локальные системы контроля версий подвержены той же проблеме: если вся история проекта хранится в одном месте, вы рискуете потерять всё.

Распределённые системы контроля версий

И в этой ситуации в игру вступают распределённые системы контроля версий (РСКВ). В таких системах как Git, Mercurial, Bazaar или Darcs 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.



Кроме того, в большей части этих систем можно работать с несколькими удалёнными репозиториями, таким образом, можно одновременно работать по-разному с разными группами людей в рамках одного проекта. Так, в одном проекте можно одновременно вести несколько типов рабочих процессов, что невозможно в централизованных системах.

Почему Git

1. Скорость
2. Простота дизайна
3. Поддержка нелинейной разработки (тысячи параллельных веток)
4. Полная распределённость
5. Возможность эффективной работы с такими большими проектами (как по скорости, так и по размеру данных)

Он невероятно быстр, очень эффективен для больших проектов, а также обладает превосходной системой ветвления для нелинейной разработки.

Основы Git

Главное отличие Git'а от любых других СКВ (например, Subversion и ей подобных) — это то, как Git смотрит на свои данные. В принципе, большинство других систем хранит информацию как список изменений (патчей) для файлов. Эти системы (CVS, Subversion и другие) относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени.



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.



Это важное отличие Git'а от практически всех других систем контроля версий. Из-за него Git вынужден пересмотреть практически все аспекты контроля версий, которые другие системы переняли от своих предшественниц. Git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх неё, чем на просто СКВ. В главе 3, коснувшись работы с ветвями в Git'е, мы узнаем, какие преимущества даёт такое понимание данных.

Работая с Subversion и CVS, вы можете редактировать файлы, но сохранить изменения в вашу базу данных нельзя (потому что она отключена от репозитория).

Три состояния

Теперь внимание. Это самое важное, что нужно помнить про Git, если вы хотите, чтобы дальше изучение шло гладко. В Git'е 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.



Каталог Git'а — это место, где Git хранит метаданные и базу данных объектов вашего проекта. Это наиболее важная часть Git'а, 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'а и помещаются на диск для того, чтобы вы их просматривали и редактировали.

Область подготовленных файлов — это обычный файл, обычно хранящийся в каталоге Git'а, 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).

Эта область может пригодится в том случае, если вы не хотите фиксировать все измененные файлы в данный момент, а хотите сохранить только какую-то их часть. Также она позволяет еще раз подумать перед тем, как сохранить внесенные изменения.

Стандартный рабочий процесс с использованием Git'а 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.

Установка Git в Windows

Просто скачиваете .exe файл и устанавливаете.

Первоначальная настройка Git

Первое, что вам следует сделать после установки Git'а, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git'е содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена:

git config --global user.name "John Doe"

git config --global user.email johndoe@example.com

git config –list – проверка настроек

git config user.name – или так

Если вы забыли, как это работает

git help <команда>

git <команда> --help

Работа с Git

git init – создание git репозитория.

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git содержащий все необходимые файлы репозитория — основу Git-репозитория. На этом этапе ваш проект ещё не находится под версионным контролем.

Жизненный цикл git:



git status – определение состояния файлов

git add – добавление файлов под версионный контроль

Файл .gitignore позволяет игнорировать файлы

git commit -m "first commit" – сохранение изменений

git commit -am 'added new file' – обход индексации

git rm <filename> - удаление файлов

git rm --cached <filename> - удаление файлов из отслеживаемых

git log – просмотр истории коммитов

git log --pretty=oneline – история коммитов в одну строчку

git commit –amend – изменение последнего коммита

git reset HEAD <filename> - отмена индексации файлов

git checkout -- benchmarks.rb – отмена последних изменений в файле

git remote – отображение удаленных репозиториев (только названия)

git remote –v – отображение удаленных репозиториев (название + url)

git remote add <name> <url> – добавление удаленного репозитория

git fetch [имя удал. сервера] – получение данных из удаленного сервера (те данные, которых у вас нет и которые не противоречат вашим)

git pull <имя удаленного сервера> <имя ветки> - забирает данные и сливает их с соответствующей веткой

git push <имя сервера> <имя ветки> - отправка данных на сервер

git config --global alias.co checkout – алиасы (более простые названия команд)

И еще куча всего полезного…