Система контроля версий Git

Г. В. Кормаков
МГУ имени М. В. Ломоносова

e-mails: *egor2898@mail.ru*

9 ноября 2018 г.

План выступления

Существуют разные подходы

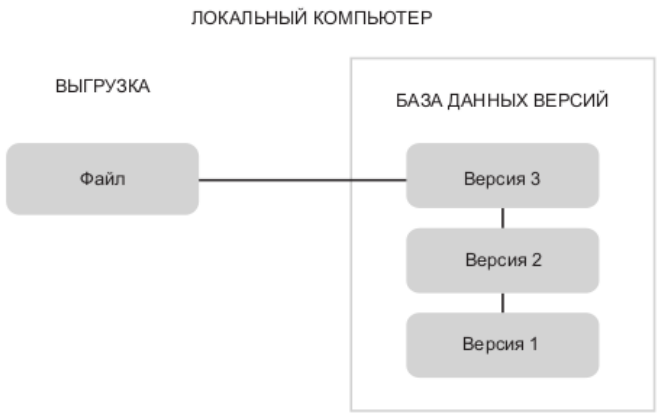

Рис. 1. Локальное управление версиями:

- \+ Очень просто
- – Легко забыть, где находится/потерять
- – Сохранить не тот файл
- – Единое хранилище

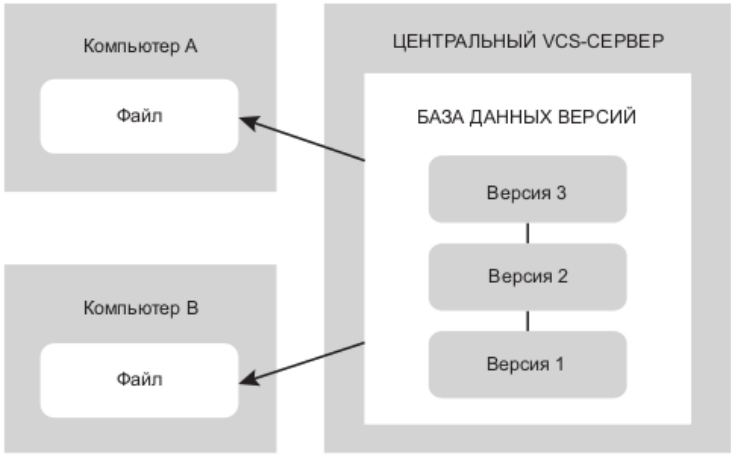

Рис. 2. Централизованное управление версиями:

- \+ Каждый участник проекта знает, что делают другие
- \+ Администрирование проще
- \+ Регулирование прав
- – Единое хранилище

Git – распределённая система контроля версий (Mercurial, Darcs)

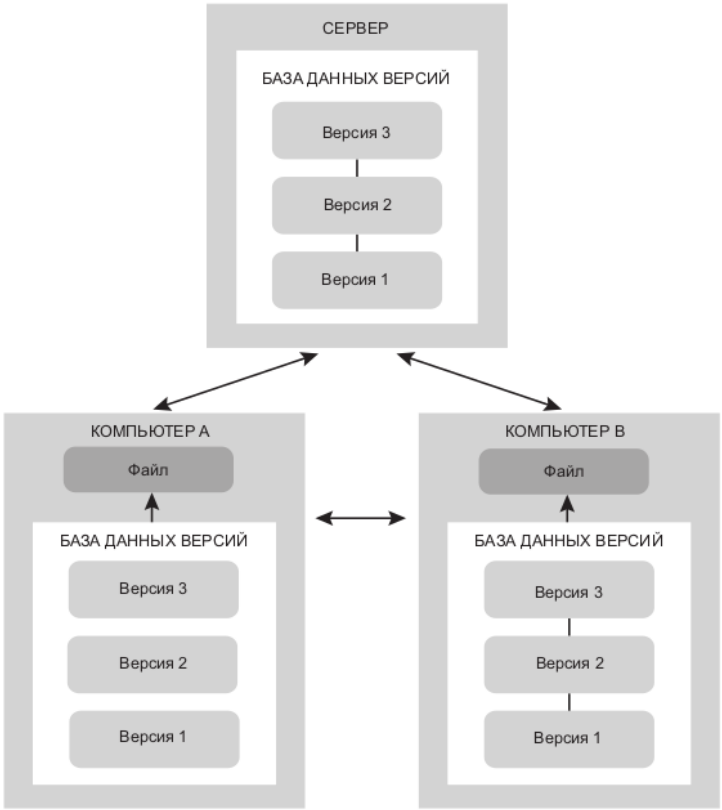

Рис. 3. Распределённое управление версиями

+ При падении основного сервера достаточно копии репозитория одного из клиентов для восстановления
+ Многие такие системы могут обладать несколькими удалёнными репозиториями. Это позволяет разным группам сотрудничать в рамках одного проекта

План выступления

Краткая предыстория git

В 2005 году отношения между разработчиками ядра Linux, и фирмой, создавшей BitKeeper, были разорваны и бесплатное использование этой системы контроля версий стало невозможным.

Сообщество разработчиков Linux начало работу над собственным инструментом, взяв за основу некоторые идеи BitKeeper. Основные цели для новой системы:

- быстродействие;
- простое проектное решение;
- мощная поддержка нелинейной разработки (тысячи параллельных ветвей);
- полностью распределенная система;
- возможность эффективной (в плане быстродействия и объема данных) работы с большими проектами, такими как ядро Linux.

Ключевые представления git: Поток снимков состояния

При создании новой версии (сохранении состояния проекта см. рис.5) в Git делается снимок всех файлов в конкретный момент времени и сохраняется ссылка на этот снимок. Вместо файлов, которые не претерпели изменений, сохраняется всего лишь ссылка на их ранее сохранённые версии.

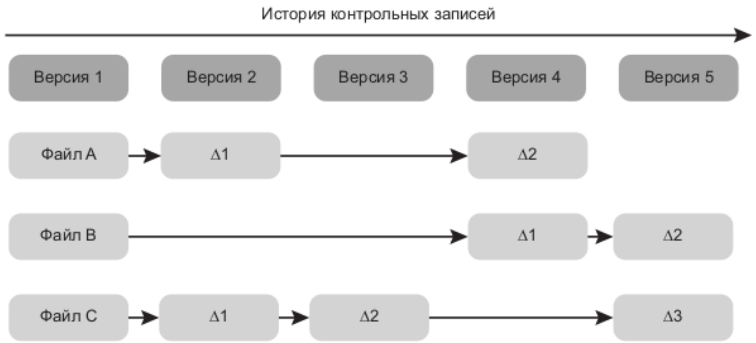

Рис. 4. Хранение данных в виде изменений, вносимых в базовую версию каждого файла

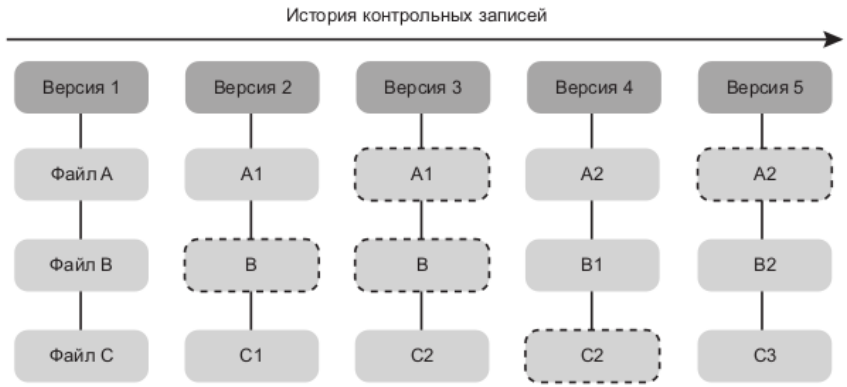

Рис. 5. Хранение данных в виде снимков состояния проекта

Ключевые представления git:

- Локальность операций: для осуществления практически всех операций системе Git требуются только локальные файлы и ресурсы
- Целостность: В системе Git для всех данных перед сохранением вычисляется контрольная сумма, по которой они впоследствии ищутся. Git всегда узнаёт о сохранении содержимого файла или папки
- Практически все операции в Git приводят к добавлению данных в базу. Систему сложно заставить выполнить неотменяемое действие или каким-то образом стереть данные.

3 основных состояния файлов в git

- *Зафиксированное (committed)* состояние означает, что данные надежно сохранены в локальной базе.
- *Модифицированное (modified)* состояние означает, что изменения уже внесены в файл, но пока не зафиксированы в базе данных.
- *Индексированное (staged)* состояние означает, что вы пометили текущую версию модифицированного файла как предназначенную для следующей фиксации.

В результате Git-проект разбивается на три основные области: *папка Git, рабочая папка* и *область индексирования* (рис. 6).

Области git-проекта:

Рис. 6. Области git-проекта

- *Папка Git* – место, где Git хранит метаданные и объектную базу данных проекта.
- *Рабочая папка* – место, куда выполняется выгрузка одной из версий проекта.
- *Область индексирования* – файл, хранящий информацию о следующей операции фиксации.

Основной процесс работы

Базовый рабочий процесс в Git выглядит так:

1. Вы редактируете файлы в рабочей папке.
2. Вы индексируете файлы, добавляя их снимки в область индексирования.
3. Вы выполняете фиксацию, беря файлы из области индексирования и сохраняя снимки в папке Git.

Соответственно, состояния файлов:

- **Committed** – сохранена конкретная версия файла в папке Git.
- **Staged** – после изменений файл перемещён в область индексирования
- **Modified** – отредактированный после выгрузки, но не проиндексированный

План выступления

Установка и первые настройки

- Установка в Debian/Ubuntu:

```
$ apt-get install git
```

- Установка на Mac:
  Xcode или попытаться запустить git через терминал.
- Установка на Windows: см. http://git-scm.com/download/win

Настройка окружения – **git config**. Инструмент позволяет задать переменные конфигурации для всех пользователей системы. Для получения списка всех параметров необходимо ввести:

```
$ git config –list
```

Первоначально необходимо задать имя пользователя, электронную почту и текстовый редактор по умолчанию (по умолчанию, обычно, Vim).

Для получения *справки* можно воспользоваться одним из 3 вариантов:

```
$ git help <команда>
$ git <команда> –help
$ man git-<команда>
```

Создание репозитория

- Инициализация репозитория в существующей папке:

```
$ git init
```

После выполнения этой команды в текущей папке создастся ваш конфигурационный файл .git. Но контроль за версиями файлов внутри рабочей папки пока отсутствует. Для того, чтобы начать отслеживать версии файла необходимо ввести, например, следующие команды:

```
$ git add *.c
$ git add LICENSE
$ git commit -m 'первоначальная версия проекта'
```

- Клонирование существующего репозитория:
  Получение копии существующего репозитория, например проекта, в котором вы хотите принять участие, выполняется командой git clone:

```
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

Жизненный цикл файла в рабочей папке

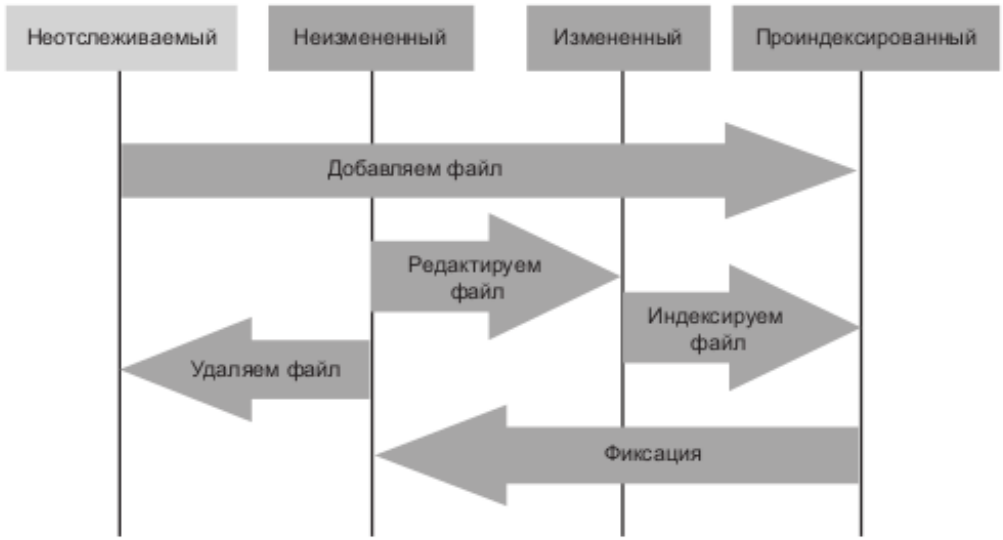

Рис. 7. Жизненный цикл файла в рабочей папке

- Индексацию проводят командой **git add**
- Фиксация - **git commit**
- Удаление файла происходит по команде **git rm** (Обычное удаление помещает файл в раздел не проиндексированных)
- Проверка статуса отслеживаемых файлов осуществляется командой **git status**
- Чтобы посмотреть все изменения пользуются **git log**

Заключение

- То, что было рассмотрено - лишь основы работы с локальным репозиторием. Здесь не был приведён пример работы с ветвлением и откатом на предыдущие версии
- Можно продолжить обучение git по книге С.Чакон, Б.Штрауб "Git для профессионального программиста". И, конечно, google :)

Спасибо за внимание!