

# **Git as Subversion**

**Руководство пользователя**

**Артём Навроцкий**

---

# **Git as Subversion: Руководство пользователя**

Артём Навроцкий

Дата публикации 2016

Авторские права © 2014-2016 Артём Навроцкий

---

# Содержание

1. О проекте .....	1
Что это? .....	1
Какова цель проекта? .....	1
Как оно работает? .....	1
Где хранятся Subversion-данные репозитория? .....	2
Как работает коммит? .....	2
Отличие от других решений .....	2
Поддержка Subversion у GitHub .....	3
SubGit .....	3
Subversion репозиторий и git svn .....	3
Функционал .....	4
Что уже есть? .....	4
Чего еще нет? .....	4
Технические ограничения .....	4
2. Предварительные требования .....	5
Рекомендации для Subversion-клиента .....	5
Рекомендации для Git-клиента .....	5
LFS для Git-клиентов использующих протокол SSH .....	5
LFS для Git-клиентов использующих протокол HTTP .....	5
Рекомендации для Git-репозитория .....	6
Файл .gitattributes .....	6
3. Установка .....	7
Быстрый старт .....	7
Установка на Debian/Ubuntu .....	7
Пакет git-as-svn .....	7
Используемые директории .....	8
Пакет git-as-svn-lfs .....	8
Сборка из исходного кода .....	9
4. Интеграция с GitLab .....	10
Список исправлений для GitLab .....	10
Точки стыка интеграции с GitLab .....	10
Добавление SVN-ссылки в интерфейс GitLab .....	11
Пример конфигурационного файла .....	11
5. SVN Properties .....	13
Файл .gitignores .....	13
Файл .gitattributes .....	14
Файл .tgitconfig .....	14
6. API .....	15
Описание интерфейса .....	15

---

# Глава 1. О проекте

## Что это?

Git as Subversion (<https://github.com/bozaro/git-as-svn>) — это реализация Subversion-сервера (по протоколу svn) для Git-репозитория.

Он позволяет работать с Git-репозиторием, используя консольный Subversion-клиент, TortoiseSVN, SvnKit и подобный инструментарий.

## Какова цель проекта?

Проект создан для того, чтобы позволить работать с одним и тем же репозиторием как в Git-стиле, так и в Subversion-стиле.

### Git-стиль

Основная идея сводится к тому, что разработчик производит все изменения в локальной ветке. Эти изменения никак не влияют на работу остальных разработчиков, но тем не менее их можно протестировать на сборочной ферме, передать другому разработчику на проверку и т.д.

Это позволяет каждому разработчику вести работу независимо, так, как ему удобно, изменяя и сохраняя промежуточные версии документов, пользуясь всеми возможностями системы (в том числе доступом к истории изменений) даже в отсутствие сетевого соединения с сервером.

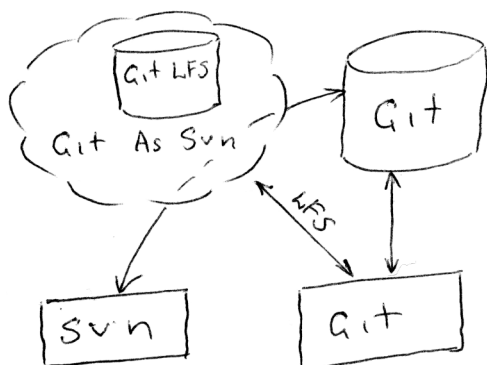
К сожалению, этот подход не работает в случае, когда изменяемые документы не поддерживают слияние (например, это характерно для двоичных файлов).

### Subversion-стиль

Использование централизованной системы контроля версий более удобно в случае использования документов не поддерживающих слияние (например, это характерно для двоичных файлов) за счет наличия механизма блокировок и более простого и короткого цикла публикации изменений.

Необходимость совместить Git-стиль и Subversion-стиль работы с одним репозиторием возникает из-за того, что разные сотрудники в рамках одного проекта работают с принципиально разными данными. Если утрировать, то программисты предпочитают Git, а художники любят Subversion.

## Как оно работает?



## Где хранятся Subversion-данные репозитория?

Для представления Subversion репозитория нужно хранить информацию о том, какой номер Subversion-реvisions соответствует какому Git-коммиту. Вычислять эту информацию каждый раз при запуске нельзя, так как тогда первый же **git push --force** нарушит порядок ревизий. Эти данные лежат в ветках `refs/git-as-svn/*`. В частности из-за этого не требуется отдельного резервного копирования для Subversion-данных.

Также часть данных, необходимых для Subversion репозитория, очень дорого получить на основании Git-репозитория.

Например:

- номер ревизии с предыдущим изменением файла;
- данные о том, откуда был скопирован файл;
- MD5-хэш файла.

Чтобы не заниматься их вычислением каждый запуск, эти данные кэшируются в файлах. Потеря данного кэша не критична для работы и его резервное копирование не имеет смысла.

Данные о блокировках файлов в данный момент также хранятся в файлах кэша.

## Как работает коммит?

Одна из самых важных деталей системы — сохранение изменений.

В общих чертах, алгоритм следующий:

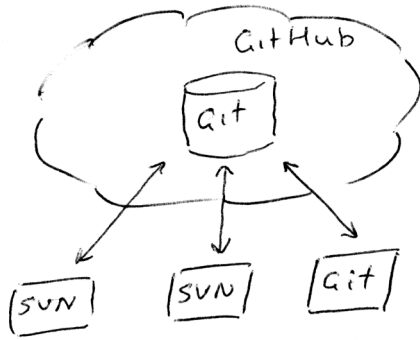
1. В момент команды **svn commit** клиент отправляет на сервер свои изменения. Сервер запоминает их. В этот же момент происходит первая проверка клиентских данных на актуальность.
2. Сервер берет голову ветки и начинает формировать новый коммит на базе полученных от клиента данных. В этот момент происходит ещё одна проверка на актуальность клиентских данных.
3. Проверяется целостность `svn properties` для заливаемых данных.
4. Сервер пытается консольным Git-клиентом сделать `push` нового коммита в текущую ветку этого же репозитория. Далее по результату `push-a`:
  - если все хорошо — загружаем последние изменения из git-коммитов и радуемся;
  - если не `fast forward` — загружаем последние изменения из git-коммитов и идём к шагу 2;
  - если отбили хуки — сообщаем клиенту;
  - если другая ошибка — сообщаем клиенту.

Таким образом, за счёт использования в данной операции консольного Git-a, мы избегаем гонки с заливкой напрямую через Git и получаем хуки в качестве приятного бонуса.

## Отличие от других решений

Проблему совмещения Git и Subversion стиля работы с системой контроля версий можно решить разными способами.

## Поддержка Subversion у GitHub

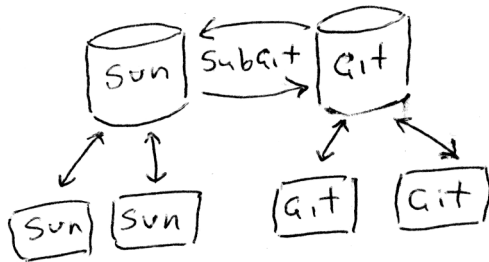


Это, наверное, самый близкий аналог.

Основная проблема данной реализации — неотделимость от GitHub. Также, внезапно, данная реализация не поддерживает Git LFS.

В случае с GitHub также не понятно, где хранится соответствие между Subversion-ревизиями и Git-коммитами. Это может быть проблемой при восстановлении репозитория после внештатных ситуаций.

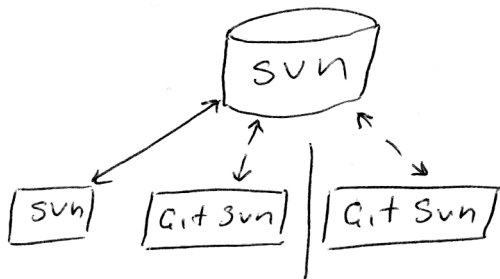
## SubGit



Сайт: <http://www.subgit.com/>

Достаточно интересная реализация при которой поддерживаются Git и Subversion-репозитории в синхронном состоянии. За счет чего обеспечивается синхронность репозитория — непонятно.

## Subversion репозиторий и git svn



Данный способ позволяет использовать Git при работе с Subversion-репозиторием, но использование общего Git-репозитория между несколькими разработчиками сильно затруднено.

Также надо учесть, что разработчику приходится использовать специфический инструмент командной строки для работы с репозиторием.

# Функционал

Данная реализация позволяет большинству Subversion-пользователей работать не задумываясь о том, что они на самом деле используют Git-репозиторий.

## Что уже есть?

- Работают, как минимум, следующие клиенты:
  - Консольный Subversion-клиент;
  - TortoiseSVN;
  - SvnKit.
- Работают, как минимум, следующие операции:
  - `svn checkout`, `update`, `switch`, `diff`
  - `svn commit`
  - `svn copy`, `move`<sup>1</sup>
  - `svn cat`, `ls`
  - `svn lock`, `unlock`
  - `svn replay` (`svnsync`)
- Поддерживается Git LFS;
- Поддерживаются `git submodules`;<sup>2</sup>
- Авторизация через LDAP;
- Интеграция с GitLab.

## Чего еще нет?

- Большие пробелы в документации;
- Из Subversion доступна только одна ветка.

## Технические ограничения

- Нельзя средствами Subversion менять `svn properties`;
- Нельзя создавать пустые директории.

---

# Глава 2. Предварительные требования

## Рекомендации для Subversion-клиента

Для автоматической расстановки Subversion свойств на добавляемых файлах и директориях используются наследуемые свойства.

Данная функция поддерживается начиная с Subversion 1.8.

Если вы используете TortoiseSVN и свойства `bugtraq: *`, то необходимо использовать TortoiseSVN 1.9 или более поздний.

## Рекомендации для Git-клиента

### LFS для Git-клиентов использующих протокол SSH

Git-клиент для получения реквизитов доступа к LFS-хранилищу выполняет на сервере через SSH команду `git-lfs-authenticate`.

Данный запрос может выполняться очень часто. На установку каждого SSH соединения тратится довольно много времени (порядка 1-ой секунды).

Для сокращения времени установки SSH соединений можно включить повторное использование SSH соединений.

Включение повторного использования сессий под Linux можно сделать командой:

```
#!/bin/sh
echo "Host *"
    ControlMaster auto
    ControlPath ~/.ssh/controlmasters/%r@%h:%p
    ControlPersist 10m
" > ~/.ssh/config
mkdir ~/.ssh/controlmasters
chmod 700 ~/.ssh/controlmasters
```

### LFS для Git-клиентов использующих протокол HTTP

Git-клиент может запрашивать реквизиты для доступа к LFS-хранилищу для каждого файла по отдельности.

Чтобы этого не происходило, необходимо включить в Git кэширование введенных паролей.

Включить кэширование паролей можно командой:

```
git config --global credential.helper cache
```

По-умолчанию пароли кэшируются в течение 15 минут.

Изменить времени жизни кэша можно командой:

```
git config --global credential.helper 'cache --timeout=3600'
```



Более подробная информация доступна по адресу: <https://help.github.com/articles/caching-your-github-password-in-git/>

## Рекомендации для Git-репозитория

### Файл .gitattributes

По-умолчанию Git изменяет окончание текстовых файлов в зависимости от текущей системы.

Для того, чтобы Git не изменял окончания файлов по-умолчанию нужно добавить в начало .gitattributes следующую строку:

```
* -text
```

---

# Глава 3. Установка

## Быстрый старт

Для того, чтобы посмотреть, как работает Git as Subversion нужно:

1. Установить Java 8 или более позднюю;
2. Скачать архив с сайта <https://github.com/bozaro/git-as-svn/releases/latest>;
3. После распаковки архива перейти в распакованный каталог и запустить команду:

```
bin/git-as-svn --config doc/config-local.example --show-config
```

В результате будет запущен Git as Subversion сервер в следующей конфигурации:

1. Сервер доступен через svn-протокол по порту 3690.

Для его проверки можно выполнить команду:

```
svn ls svn://localhost/example
```

2. Для доступа к серверу необходимо использовать пользователя:

Имя пользователя: test

Пароль: test

3. Репозиторий и кэш будут созданы в каталоге build:

- example.git — каталог с репозиторием, доступным через svn-протокол;
- git-as-svn.mapdb\* — файлы с кэшем дорого вычисляемых данных.

## Установка на Debian/Ubuntu

Вы можете установить Git as Subversion на Debian/Ubuntu репозиторий при помощи команд:

```
#!/bin/bash
# Add package source
echo "deb https://dist.bozaro.ru/ debian/" | sudo tee /etc/apt/sources.list.d/dist.boza
curl -s https://dist.bozaro.ru/signature.gpg | sudo apt-key add -
# Install package
sudo apt-get update
sudo apt-get install git-as-svn
sudo apt-get install git-as-svn-lfs
```

## Пакет git-as-svn

Данный пакет содержит Git as Subversion.

После его установки Git as Subversion запускается в режиме демона и доступен по svn-протоколу на порту 3690. Демон запускается от имени пользователя git.

Для доступа к серверу необходимо использовать пользователя:

Имя пользователя: test

Пароль: test

Для проверки можно выполнить команду вида:

```
svn ls --username test --password test svn://localhost/example/
```

## Используемые директории

Данный пакет по умолчанию настроен на использование следующих директорий:

`/etc/git-as-svn`

Это директория с конфигурационными файлами.

`/usr/share/doc/git-as-svn`

Данная директория содержит в себе данную документацию на установленную версию.

`/var/git/lfs`

Данная директория по умолчанию используется для хранения Git LFS файлов.

Она должна быть доступна на запись для пользователя `git`.

`/var/git/repositories`

Эта директория по умолчанию используется для хранения Git-репозитория.

Репозитории должны быть доступны на запись для пользователя `git`.

`/var/log/git-as-svn`

Эта директория используется для записи логов.

Она должна быть доступна на запись для пользователя `git`.

Параметры ротации логов задаются через конфигурационный файл `/etc/git-as-svn/log4j2.xml`.

`/var/cache/git-as-svn`

Эта директория используется для хранения кэша Git as Subversion.

Она должна быть доступна на запись для пользователя `git`.

Потеря содержимого данной директории не является критичной для работы и не влечёт за собой потерю пользовательских данных.

## Пакет git-as-svn-lfs

Данный пакет содержит в себе скрипт `git-lfs-authenticate`.

Скрипт `git-lfs-authenticate` используется для предоставления реквизитов доступа к Git LFS серверу по HTTP протоколу для пользователей, использующих SSH для работы с Git-репозиторием (<https://github.com/github/git-lfs/blob/master/docs/api/README.md>).

Данный скрипт общается через Unix Domain Socket с Git as Subversion.

В Git as Subversion отправляет имя (`mode = username`) или идентификатор (`mode = external`) пользователя, полученное из определённой параметром `variable` переменной окружения (по умолчанию: `GL_ID`).

Для проверки настройки скрипта можно выполнить локально на сервере команду вида:

```
#!/bin/bash
# Set environment variable defined in configuration file
export GL_ID=key-1
```

```
# Check access to repository
sudo su git -c "git-lfs-authenticate example download"
```

Или на клиенте команду вида:

```
#!/bin/bash
ssh git@remote -C "git-lfs-authenticate example download"
```

Результат выполнения команды должен выглядеть примерно следующим образом:

```
{
  "href": "https://api.github.com/lfs/bozaro/git-as-svn",
  "header": {
    "Authorization": "Bearer SOME-SECRET-TOKEN"
  },
  "expires_at": "2016-02-19T18:56:59Z"
}
```

## Сборка из исходного кода

Данный проект изначально рассчитан на сборку в Ubuntu.

Для сборки из исходного кода необходимо локально установить:

1. Java 8 (пакет `openjdk-8-jdk`);
2. xml2po (пакет `gnome-doc-utils`) — необходимо для сборки документации;
3. protoc (пакет `protobuf-compiler`) — необходимо для сборки API.

Полностью собрать дистрибутив можно командой:

```
./gradlew assembleDist
```

Комплект установочных файлов будет располагаться в директории: `build/distributions`

---

# Глава 4. Интеграция с GitLab

## Список исправлений для GitLab

Для интеграции с GitLab нужно установить следующие исправления на GitLab:

- [#230 \(gitlab-shell\)](#): Добавлена команда `git-lfs-authenticate` в белый список (включено в 7.14.1);
- [#237 \(gitlab-shell\)](#): Выполнение команды `git-lfs-authenticate` в оригинальными аргументами (включено в 8.2.0);
- [#9591 \(gitlabhq\)](#): Добавлено API для получения информации о пользователе по идентификатору SSH-ключа (включено в 8.0.0);
- [#9728 \(gitlabhq\)](#): Исправлена ошибка во время отображения репозитория без веток (включено в 8.2.0).

## Точки стыка интеграции с GitLab

В случае с GitLab есть несколько точек стыка:

- Список репозитория

Git as Subversion автоматически получает список репозитория в момент старта через GitLab API.

Далее этот список поддерживается в актуальном состоянии при помощи System Hook, который так же регистрируется автоматически.

- Авторизация и аутентификация пользователей

Для аутентификации пользователей и определения наличия прав на репозиторий так же используется GitLab API.

- Git Hooks

При коммите через Git as Subversion выполняются хуки от GitLab. Эти хуки, в частности, позволяют видеть информацию о новых коммитах без задержки через WEB-интерфейс GitLab.

Для того, чтобы этот функционал работал, Git as Subversion должен передать идентификатор GitLab-пользователя (GL\_ID), полученный при его авторизации.



### Важно

Из-за этого в случае интеграции с GitLab авторизация пользователей должна так же проходить через GitLab.

- Git LFS

В случае использования Git LFS надо так же указать путь до GitLab LFS хранилища.

GitLab с версии 8.2 использует общее хранилище LFS-файлов для всех репозитория. Файлы хранятся в отдельном каталоге в сыром виде.

Интеграция с LFS хранилищем GitLab происходит на уровне файлов. Никакое API от GitLab при этом не используется.

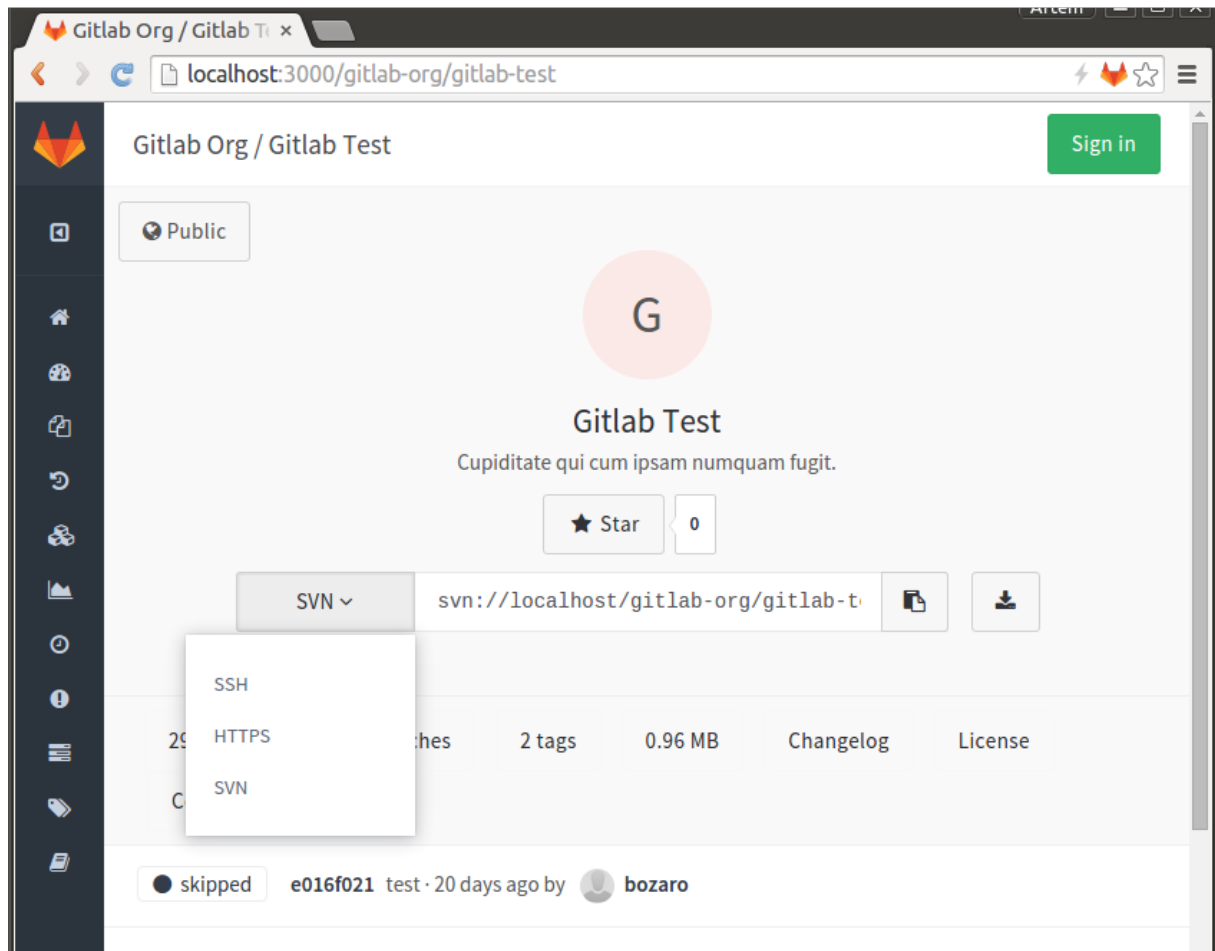
- Git LFS авторизация для SSH-пользователей

К сожалению, GitLab не предоставляет скрипт `git-lfs-authenticate`, который отвечает за SSO авторизацию SSH-пользователей на Git LFS сервере. Для настройки данного скрипта см. «Пакет `git-as-svn-lfs`».

# Добавление SVN-ссылки в интерфейс GitLab

Для того, чтобы добавить SVN-ссылку в интерфейс GitLab нужно взять последний коммит из ветки [https://github.com/bozaro/gitlabhq/commits/svn\\_url](https://github.com/bozaro/gitlabhq/commits/svn_url).

## Пример SVN-ссылки в интерфейсе GitLab



## Пример конфигурационного файла

```
1 !config:
2 realm: Example realm
3 compressionEnabled: true
4
5 # Use GitLab repositories
6 repositoryMapping: !gitlabMapping
7   path: /var/opt/gitlab/git-data/repositories/
8   template:
9     branch: master
10    renameDetection: true
11
12 # Use GitLab user database
13 userDB: !gitlabUsers {}
14
```

```
15 shared:
16   # Web server settings
17   # Used for:
18   # * detecticting add/remove repositories via GitLab System Hook
19   # * git-lfs-authenticate script (optionaly)
20   - !web
21     baseUrl: http://git-as-svn.local/
22     listen:
23     - !http
24       host: localhost
25       port: 8123
26       # Use X-Forwarded-* headers
27       forwarded: true
28   # GitLab LFS server
29   - !lfs
30     # Secret token for git-lfs-authenticate script
31     # token: secret
32     path: /mnt/storage/lfs-objects
33     saveMeta: false
34     compress: false
35     layout: GitLab
36   # GitLab server
37   - !gitlab
38     url: http://localhost:3000/
39     hookUrl: http://localhost:8123/
40     token: qytzQc6uYiQfsoqJxGuG
41
```

---

# Глава 5. SVN Properties

Основная беда `svn properties` в том, что их надо поддерживать в синхронном состоянии между Git и Subversion.

Из-за этого произвольные `svn properties` не поддерживаются. Чтобы значения `svn properties` соответствовали Git-представлению, они генерируются на лету на основании содержимого репозитория.

При этом:

- при коммите проверяется, что `svn properties` файла/директории точно соответствуют тому, что должно быть по данным репозитория;
- средствами Subversion большую часть свойств изменить нельзя (исключения: `svn:executable`, `svn:special`);
- если какой-либо файл влияет на `svn properties` других файлов, то после его изменения `svn properties` этих файлов так же поменяются.



## Важно

Для удобства пользователей Git as Subversion активно использует наследуемые свойства.

Для того, чтобы они работали необходимо использовать клиент Subversion 1.8 или более поздний.

В противном случае будут проблемы с `svn properties` для новых файлов и директорий.

## Файл .gitignores

Данный файл влияет на свойство `svn:ignore` и `svn:global-ignores` для директории и её поддиректорий.

Например, файл в каталоге `/foo` с содержимым:

```
.idea/libraries
*.class
*/build
```

Проецируется на свойства:

- для каталога `/foo`:  
`svn:global-ignores: *.class`
- для каталогов `/foo/*`:  
`svn:ignore: build`
- для каталога `/foo/.idea`:  
`svn:ignore: libraries build`



## Важно

Для Subversion нет способа сделать исключения для директорий, в результате, к примеру, правила `/foo` (файл или директория `foo`) и `/foo/` (директория `foo`) в Subversion будут работать одинаково, хотя в Git у них поведение разное.



Правила вида "все кроме" не поддерживаются при проецировании на `svn:global-ignores`.

## Файл `.gitattributes`

Данный файл влияет на свойства `svn:eol-style` и `svn:mime-type` файлов от данной директории и `svn:auto-props` у самой директории.

Например, файл с содержимым:

```
*.txt          text eol=native
*.xml          eol=lf
*.bin          binary
```

Добавит к директории свойство `svn:auto-props` с содержимым:

```
*.txt = svn:eol-style=native
*.xml = svn:eol-style=LF
*.bin = svn:mime-type=application/octet-stream
```

И файлам в данной директории:

- с суффиксом `.txt` свойство `svn:eol-style` = `native`
- с суффиксом `.xml` свойство `svn:eol-style` = `LF`
- с суффиксом `.bin` свойство `svn:mime-type` = `application/octet-stream`

## Файл `.tgitconfig`

Данный файл меняет только свойства директории, в которой он расположен.

Свойства проецируются один-к-одному, например, файл с содержимым:

```
[bugtraq]
url = https://github.com/bozaro/git-as-svn/issues/%BUGID%
logregex = #(\d+)
warnifnoissue = false
```

Будет преобразован в свойства:

- `bugtraq:url` = `https://github.com/bozaro/git-as-svn/issues/%BUGID%`
- `bugtraq:logregex` = `#(\d+)`
- `bugtraq:warnifnoissue` = `false`



### Важно

Если вы используете данные `svn properties`, то крайне рекомендуется использовать TortoiseSVN 1.9 или более поздний.

В противном случае TortoiseSVN будет пытаться установить данные параметры для всех вновь создаваемых каталогов.

## Глава 6. API



<http://xkcd.ru/927/>

## Описание интерфейса

При реализации интерфейса API предъявлялись следующие требования:

- Схема должна быть первична.

Это позволяет использовать схему как документацию и, в некоторых случаях, генерировать код для работы с API.

- Реализация клиента должна быть тривиальной на любом языке программирования.

В результате, для схемы был выбран Protocol Buffers (<https://developers.google.com/protocol-buffers/>).

Так как Protocol Buffers не имеет штатного RPC, общение ведется по протоколу HTTP.

Для вызова метода, отправляется запрос на URL вида:

`http://somehost/<repository>/<service>/<method>.<format>`

При этом:

`<repository>`

Имя репозитория, для которого вызывается метод API.

Не указывается для общих API.

`<service>`

Имя сервиса внутри .proto-схемы в нижнем регистре.

`<method>`

Имя метода внутри .proto-схемы в нижнем регистре.

`<format>`

Формат сериализации сообщений.

Поддерживаются следующие форматы:

- bin (application/x-protobuf)

Бинарная сериализация Protocol Buffers.

- json (application/json)

Сериализация в JSON. Позволяет использовать API без привязки к Protocol Buffers.

- xml (application/xml)

Сериализация в XML. Позволяет использовать API без привязки к Protocol Buffers.

- txt (text/plain)

Текстовая сериализация. В основном полезна для вызова некоторых методов из браузера.

Аргумент RPC-вызова может быть передан через тело POST-запроса.

Скалярные атрибуты верхнего уровня можно так же передать через параметры URL.