Git as Subversion

Руководство пользователя

Артём Навроцкий

Git as Subversion: Руководство пользователя

Артём Навроцкий

Дата публикации 2016

Содержание

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-репозиториев.

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.

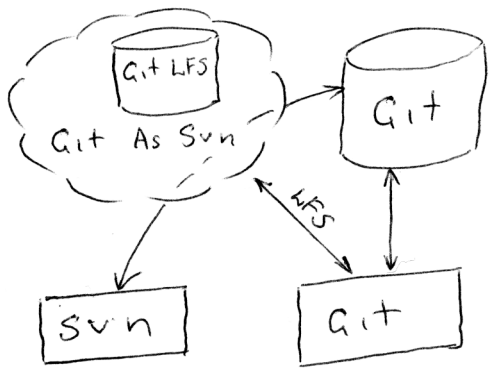
К сожалению, этот подход не работает в случае, когда изменяемые документы не поддерживают слияние (например, это характерно для двоичных файлов).

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-ревизии 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 файла кэша.

Как работает коммит?

Одна из самых важных деталей системы — сохранение изменений.

В общих чертах, алгоритм следующий:

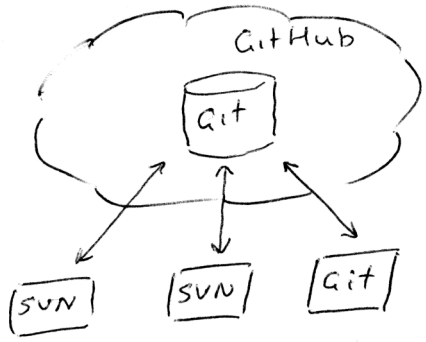
1. В момент команды **svn commit** клиент отправляет на сервер свои изменения. Сервер запоминает их. В этот же момент происходит первая проверка клиентских данных на актуальность.
2. Сервер берет голову ветки и начинает формировать новый коммит на базе полученных от клиента данных. В этот момент происходит 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-а:
   - 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-а, мы избегает 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

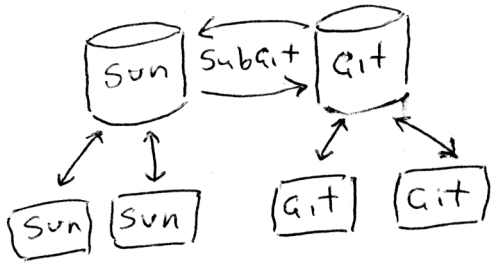


Это, наверное, самый близкий аналог.

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 репозиториев после внештатных ситуаций.

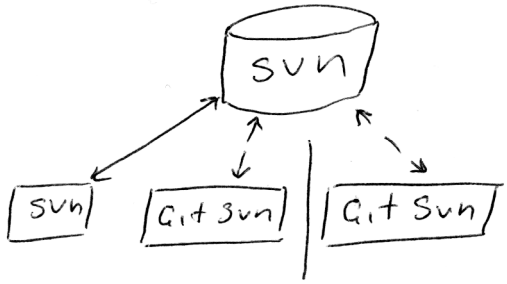
SubGit



Сайт: http://www.subgit.com/

Достаточно интересная реализация при которой поддерживаются Git и Subversion-репозитории в синхронном состоянии. За счет чего обеспечивается синхронность репозиториев — 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[1]
  - svn cat, ls
  - svn lock, unlock
  - svn replay (svnsync)
- Поддерживается Git LFS;
- Поддерживаются git submodules;[2]
- 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 подзний.

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 введённых 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 bintray GPG key
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 379CE192D401AB61
# Add package source
echo "deb https://dl.bintray.com/bozaro/git-as-svn debian main" | sudo tee /etc/apt/sou
# 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-репозиториев.

  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 Subverison 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 репозиториев

  Git as Subversion автоматически получает список репозиториев 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 репозиториев. 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.

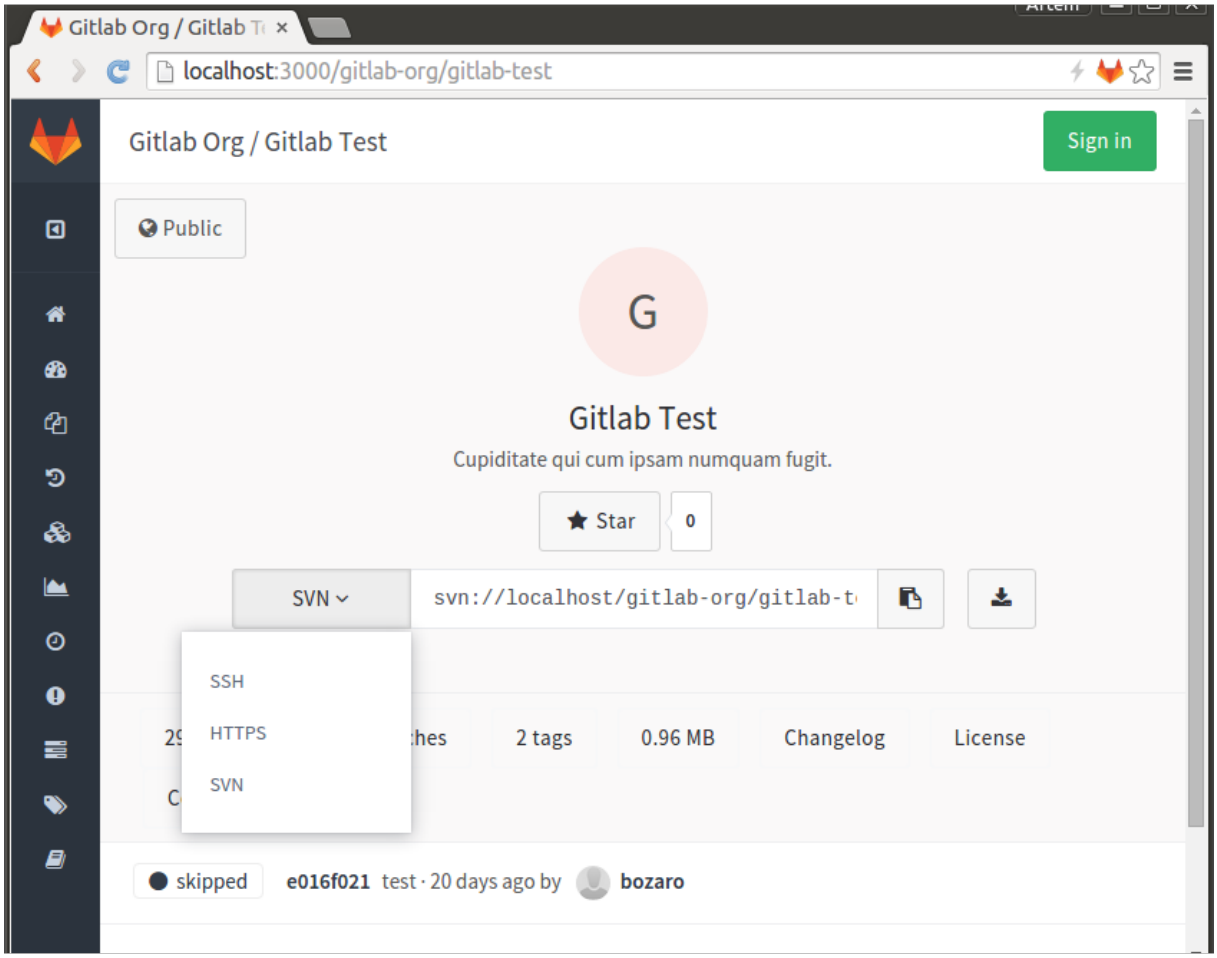
- Git LFS авторизация для SSH-пользователей

  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.

Пример SVN-ссылки в интерфейсе GitLab



Пример конфигурационного файла

```
!config:
realm: Example realm
compressionEnabled: true
parallelIndexing: true

# Use GitLab repositories
repositoryMapping: !gitlabMapping
  path: /var/opt/gitlab/git-data/repositories/
  # Uncomment following to only handle repositories with specified tags (add them t
  # repositoryTags:
  #   - git-as-svn
  template:
    branch: master
    renameDetection: true
```

```
15
16 # Use GitLab user database
17 userDB: !gitlabUsers {}
18
19 shared:
20   # Web server settings
21   # Used for:
22   #  * detecticting add/remove repositories via GitLab System Hook
23   #  * git-lfs-authenticate script (optionaly)
24   - !web
25     # baseUrl: http://git-as-svn.local/
26     listen:
27     - !http
28       host: localhost
29       port: 8123
30       # Use X-Forwarded-* headers
31       forwarded: true
32   # GitLab LFS Client
33   - !gitlabLfs {}
34   # GitLab server
35   - !gitlab
36     url: http://localhost:3000/
37     hookUrl: http://localhost:8123/
38     token: qytzQc6uYiQfsoqJxGuG
39
```

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 = navtive`
- 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. SVN+SSH

Rationale

The SVN protocol is totally unencrypted, and due to the way git-as-svn has to proxy authentication through to git servers, almost all authentication happens in plaintext.

Clearly this is undesirable, not only is potentially private code exposed over the svn protocol, but so are passwords and usernames.

Traditionally SVN has two ways of preventing this:

- Use HTTPS
- Use svn+ssh

The HTTP protocol is substantially different from the SVN protocol and is currently unimplemented in git-as-svn

Thus leaving the svn+ssh mechanism.

How does SVN+SSH work?

Normally when a client calls `svn <command> svn://host/path`, for an appropriate `<command>`, the subversion client will open a connection to the `host` server on port 3690. After an initial handshake as per the SVN protocol the server will ask the client to authenticate.

If possible the client will attempt to perform its actions anonymously, and if necessary the server will then ask for reauthentication.

If a client calls `svn <command> svn+ssh//username@host/path`, the subversion client will internally ask ssh to open connection using something equivalent to: `ssh username@host "svnserve -t"`.

If ssh succesfully connects, the SSH will run `svnserve -t` on the host, which will then proceed with the SVN protocol handshake over its `stdin` and `stdout`, and the client will use the `stdin` and `stdout` of the ssh connection.

When the server asks the client to authenticate, the server will offer the `EXTERNAL` authentication mechanism. (Possibly with the `ANONYMOUS` mechanism.)

If the client uses `EXTERNAL` mechanism, the server sets the user to be either the currently logged in user from the ssh, (or an optional tunnel-user parameter.)

Securing the `svnserve -t` call and protecting against semi-malicious uses of the `--tunnel-user` option or even the calling of other commands in cases of multiple users for a single repository requires some thought.

Often this is protected through the use of a suitable `command=""` parameter in the `authorized_keys` file, coupled with other options. e.g.

```
command="/usr/bin/svnserve -t --tunnel-user username",no-port-forwarding,no-
X11-forwarding,no-agent-forwarding,no-pty ssh-rsa ...
```

Of note, in this example the command provided by the client is ignored but it could be checked and managed as appropriately. In fact these techniques are used in the `authorized_keys` files of most `git` servers.

This provides a simple first way to handle `svn+ssh`, if we set `command="nc localhost 3690"` then whenever we connect by ssh we will be passed directly to the git-as-svn server. The downside being that the client will be asked to authenticate.

A better `git-as-svn-svnserve`

Handling the `EXTERNAL` authentication mechanism properly without creating a new port to listen on and a new adjusted SVN protocol is not possible.

However there is another way:

We can stand in the middle of the SVN protocol stream, catch the authentication handshake, proxy it before stepping back and letting the client and server talk to each other.

We can create a new authentication mechanism on the `git-as-svn` server that requires a secret token known only by us, to allow us to pass in the external username (or other identifier) as the user authentication using `sshKeyUsers` to proxy the `UserDB`

We can then use `git-as-svn-svnserve-tunnel SECRET EXTERNAL_USERNAME` as a replacement for `svnserve -t` or `nc localhost 3690` in the `command=""` option in authorized_keys.

Of course we need to keep the `authorized_keys` file up-to-date

Gitlab & git-as-svn-svnserve

There are two ways that Gitlab manages ssh access.

- Updating the git user's `authorized_keys` every time a SSH key is changed.
- The use of an SSH `AuthorizedKeysCommand`

First, let's look at the `authorized_keys` case.

Gitlab will update the `authorized_keys` file over time.

If you set the option: `gitlab_shell['auth_file']` in the `gitlab.rb` configuration file to a different location, you can catch changes to this file, and change the `command=""` option to something that will check whether we are trying to perform svn and handle it if so.

The suggested config, at least for Gitlab docker and assuming that git-as-svn has been installed in `/opt/git-as-svn` is:

- `/etc/gitlab/gitlab.rb`:

```
...
################################################################################
## gitlab-shell
################################################################################
...
# gitlab_shell['auth_file'] = "/var/opt/gitlab/.ssh/authorized_keys"
gitlab_shell['auth_file'] = "/var/opt/gitlab/ssh-shadow/authorized_keys"
...
```

- `/opt/git-as-svn/config.yaml`:

```
!config:
realm: Git-as-svn Realm
compressionEnabled: true
parallelIndexing: true
```

```
# Use GitLab repositories
repositoryMapping: !gitlabMapping
  path: /var/opt/gitlab/git-data/repositories/
  # Uncomment following to only handle repositories with specified tags (add them to r
  # repositoryTags:
  #   - git-as-svn
  template:
    branch: master
    renameDetection: true

# Wrap the Gitlab user database with sshKeyUsers
userDB:
  !sshKeyUsers
    userDB: !gitlabUsers {}
    sshKeysToken: CHANGE_THIS_TO_SOMETHING_SECRET

shared:
  # Web server settings
  # Used for:
  #  * detecticting add/remove repositories via GitLab System Hook
  #  * git-lfs-authenticate script (optionaly)
  - !web
    # baseUrl: http://git-as-svn.local/
    listen:
    - !http
      host: localhost
      port: 8123
      # Use X-Forwarded-* headers
      forwarded: true
  # GitLab LFS Client
  - !gitlabLfs {}
  # GitLab server
  - !gitlab
    url: http://localhost:3000/
    hookUrl: http://localhost:8123/
    token: qytzQc6uYiQfsoqJxGuG
  # Manage authorized_keys
  - !sshKeys
    shadowSSHDirectory: /var/opt/gitlab/ssh-shadow
    realSSHDirectory: /var/opt/gitlab/.ssh
    originalAppPath: /opt/gitlab/embedded/service/gitlab-shell/bin/gitlab-shell
    svnservePath: /opt/git-as-svn/bin/git-as-svn-svnserve
```

- `/opt/git-as-svn/bin/git-as-svn-svnserve`:

```
#!/bin/bash

###############################################################
# git-as-svn-svnserve
#
# Shadow the default gitlab/gitea shell and allow svnserve
###############################################################

SHADOW_SHELL_PATH="/opt/gitlab/embedded/service/gitlab-shell/bin/gitlab-shell"
TUNNEL_PATH="/opt/git-as-svn/bin/git-as-svn-svnserve-tunnel"
KEY="$1"
```

```
REAL_SHELL_PATH="$SHADOW_SHELL_PATH"
SECRET="CHANGE_THIS_TO_SOMETHING_SECRET"

SSH_ORIGINAL_COMMANDS=($SSH_ORIGINAL_COMMAND)

if [ -n "$SSH_ORIGINAL_COMMAND" ] && [ "${SSH_ORIGINAL_COMMANDS[0]}" = "svnserve" ] ;
    ## TUNNEL TO OUR SVNSERVER WITH MAGIC AUTHENTICATION ##
    exec "$TUNNEL_PATH" "$SECRET" "$KEY"
else
    exec -a "$REAL_SHELL_PATH" "$SHADOW_SHELL_PATH" "$@"
fi
```

- `/opt/git-as-svn/bin/git-as-svn-svnserve-tunnel`:

```
#!/bin/bash

###############################################################
# git-as-svn-svnserve-tunnel
#
# Use a bit of bash hackery to implement svnserve -t by
# pushing stdin to the svn port (3690) but hijack the
# authentication phase to pass in the ssh key id
###############################################################

SECRET="$1"
KEY="$2"
FAKE_AUTH="( success ( ( EXTERNAL ) 16:Git-as-svn Realm ) )"

function failed {
    echo "$0: Unable to connect to svn service! Is it running?" 1>&2
    exit
}
trap failed err

OUR_PID=$$
function finish {
    pkill -P $OUR_PID
    exec 3>&- 3<&-
}
trap finish EXIT

exec 3<>/dev/tcp/localhost/3690

trap finish err

function read_bracket {
    BEEN_IN=false
    NBRACK=0

    while ! $BEEN_IN || [ $NBRACK != 0 ]; do
        IFS= read -n1 -r -d '' FROM
        case $FROM in
            '('
            NBRACK=$(($NBRACK + 1))
            BEEN_IN=true
            ;;
```

```
            ')')
            NBRACK=$(($NBRACK - 1))
            ;;
            '')
            break
        esac
        echo -ne "$FROM"
    done
    IFS= read -n1 -r -d '' FROM
    echo -ne "$FROM"
    if [ "X$FROM" = "X" ]; then
        exec 0<&-
        exit
    fi
}

# Send server capabilities to client
read_bracket <&3 >&1

# Send client capabilities to server
read_bracket <&0 >&3

# Get the server authentication
AUTH_LIST_FROM_SERV=$(read_bracket <&3)

# Send the server our information
AUTHBODY=$(echo -ne "\0$SECRET\0$KEY" | base64)
AUTHBODY_LENGTH=${#AUTHBODY}
echo "( KEY-AUTHENTICATOR ( $AUTHBODY_LENGTH:$AUTHBODY ) )" >&3
if ! { command >&3; } 2>/dev/null; then
    exit
fi

# send the fake auth list to the client
echo "$FAKE_AUTH" >&1
if ! { command >&1; } 2>/dev/null; then
    exit
fi

# throwaway the client's response
read_bracket <&0 > /dev/null

# THEN PRETEND THAT THE REST OF IT WENT THAT WAY
(
    cat <&3 >&1 &
    CAT_PID=$!
    function on_exit {
        kill $CAT_PID
    }
    trap on_exit EXIT
    wait
    kill $OUR_PID
) &

cat <&0 >&3
pkill -P $OUR_PID
```

In the second case, if we proxy the `AuthorizedKeysCommand`, and just replace the `command=""` option as above then we have a working solution.

We have two main options, we can keep the same user, e.g. `git` for both subversion and git, or we could create another user.

The first option requires that we proxy the original app and replace it with our own. The second is similar but we leave the original response alone for git, just replacing it for svn

The first option is described below.

- `/assets/sshd_config`:

```
...
# AuthorizedKeysCommand /opt/gitlab/embedded/service/gitlab-shell/bin/gitlab-shell-aut
# AuthorizedKeysCommandUser git
AuthorizedKeysCommand /opt/git-as-svn/bin/git-as-svn-authorized-keys-command git %u %k
AuthorizedKeysCommandUser git
...
```

- `/opt/git-as-svn/bin/git-as-svn-authorized-keys-command`:

```
#!/bin/bash

################################################################
# git-as-svn-authorized-keys_command
#
# Shadow the default ssh AuthorizedKeysCommand and adjust its
# output to replace the original command with our svnserve
################################################################

################################################################
# For Gitlab Docker:
################################################################
ORIGINAL_AUTHORIZED_COMMAND="/opt/gitlab/embedded/service/gitlab-shell/bin/gitlab-shel
ORIGINAL_APP_PATH="/opt/gitlab/embedded/service/gitlab-shell/bin/gitlab-shell"
SVN_SERVE_PATH="/opt/git-as-svn/bin/git-as-svn-svnserve"

exec -a "$ORIGINAL_AUTHORIZED_COMMAND"  "$ORIGINAL_AUTHORIZED_COMMAND" "$@" | sed -e '
```

- `/opt/git-as-svn/config.yaml`:

```
!config:
realm: Git-as-svn Realm
compressionEnabled: true
parallelIndexing: true

# Use GitLab repositories
repositoryMapping: !gitlabMapping
  path: /var/opt/gitlab/git-data/repositories/
  # Uncomment following to only handle repositories with specified tags (add them to r
  # repositoryTags:
  #   - git-as-svn
  template:
    branch: master
```

```
    renameDetection: true

# Wrap the Gitlab user database with sshKeyUsers
userDB:
  !sshKeyUsers
    userDB: !gitlabUsers {}
    sshKeysToken: CHANGE_THIS_TO_SOMETHING_SECRET

shared:
  # Web server settings
  # Used for:
  #  * detecticting add/remove repositories via GitLab System Hook
  #  * git-lfs-authenticate script (optionaly)
  - !web
    # baseUrl: http://git-as-svn.local/
    listen:
    - !http
      host: localhost
      port: 8123
      # Use X-Forwarded-* headers
      forwarded: true
  # GitLab LFS Client
  - !gitlabLfs {}
  # GitLab server
  - !gitlab
    url: http://localhost:3000/
    hookUrl: http://localhost:8123/
    token: qytzQc6uYiQfsoqJxGuG
```

- `/opt/git-as-svn/bin/git-as-svn-svnserve` and `/opt/git-as-svn/bin/git-as-svn-svnserve-tunnel` same as above.

Gitea

There are two ways that Gitea manages ssh access.

- If Gitea is deferring to an external SSHD. It will update the git user's `authorized_keys` every time a SSH key is changed.
- If Gitea is using its own internal SSHD. It will run the `gitea serv` command each time.
- The use of an SSH `AuthorizedKeysCommand` in Gitea v1.7.0+

First, let's look at the `authorized_keys` case.

Gitea will update the `authorized_keys` file over time.

If you set the option: `SSH_ROOT_PATH` in the `[server]` of the gitea `app.ini` to a shadow location you can catch changes to this file, and change the `command=""` option to something that will check whether we are trying to perform svn and handle it if so.

The suggested config, at least for Gitea docker, and assuming that git-as-svn has been installed in `/app/git-as-svn` is:

- `/data/gitea/conf/app.ini`:

```
...
```

```
[server]
...
SSH_ROOT_PATH=/data/git/ssh-shadow
...
```

- /app/git-as-svn/config.yaml:

```
!config:
realm: Git-as-svn Realm
compressionEnabled: true
parallelIndexing: true

# Use Gitea repositories
repositoryMapping: !giteaMapping
  path: /data/git/repositories
  template:
    branch: master
    renameDetection: true

# Use Gitea user database
userDB:
  !sshKeyUsers
    userDB: !giteaUsers {}
    sshKeysToken: CHANGE_THIS_TO_SOMETHING_SECRET

shared:
  # Gitea LFS server - uses the GitLab layout
  - !localLfs
    path: /data/git/lfs
    saveMeta: false
    compress: false
    layout: GitLab
  # Gitea server
  - !gitea
    url: http://localhost:3000/api/v1
    #token: de0c16fdc2c2ec5bcb4917922900015d3bceb82b
    token: 90c68b84fb04e364c2ea3fc42a6a2193144bc07d
  - !giteaSSHKeys
  # Or if using Gitea v1.7.0 just: !sshKeys
    shadowSSHDirectory: /data/git/ssh-shadow
    realSSHDirectory: /data/git/.ssh
    originalAppPath: /app/gitea/gitea
    svnservePath: /app/gitea/git-as-svn-svnserve
```

- /app/git-as-svn/bin/git-as-svn-svnserve:

```
#!/bin/bash

################################################################
# git-as-svn-svnserve
#
# Shadow the default gitlab/gitea shell and allow svnserve
################################################################

SHADOW_SHELL_PATH="/app/gitea/gitea"
```

```
  TUNNEL_PATH="/app/git-as-svn/bin/git-as-svn-svnserve-tunnel"
  KEY="$2"
  SUBCOMMAND="$1"
  REAL_SHELL_PATH="$SHADOW_SHELL_PATH"

  if [ "$SUBCOMMAND" != "serv" ]; then
      exec -a "$REAL_SHELL_PATH" "$SHADOW_SHELL_PATH" "$@"
  fi

  SECRET="CHANGE_THIS_TO_SOMETHING_SECRET"

  SSH_ORIGINAL_COMMANDS=($SSH_ORIGINAL_COMMAND)

  if [ -n "$SSH_ORIGINAL_COMMAND" ] && [ "${SSH_ORIGINAL_COMMANDS[0]}" = "svnserve" ] ;
      ## TUNNEL TO OUR SVNSERVER WITH MAGIC AUTHENTICATION ##
      exec "$TUNNEL_PATH" "$SECRET" "$KEY"
  else
      exec -a "$REAL_SHELL_PATH" "$SHADOW_SHELL_PATH" "$@"
  fi
```

- `/app/git-as-svn/bin/git-as-svn-svnserve-tunnel` should be the same as in the gitlab case.

For the second case, we need to shadow the gitea binary

So we would need to move the original gitea from `/app/gitea/gitea` to `/app/gitea/gitea.shadow`

And either create `/app/gitea/gitea` as a symbolic link or just copy the below `/app/git-as-svn/bin/git-as-svn-svnserve` as it.

`/app/git-as-svn/bin/git-as-svn-svnserve`:

```
#!/bin/bash

################################################################
# git-as-svn-svnserve
#
# Shadow the default gitlab/gitea shell and allow svnserve
################################################################

SHADOW_SHELL_PATH="/app/gitea/gitea.shadow"
TUNNEL_PATH="/app/git-as-svn/bin/git-as-svn-svnserve-tunnel"
KEY="$2"
SUBCOMMAND="$1"
REAL_SHELL_PATH="/app/gitea/gitea"

if [ "$SUBCOMMAND" != "serv" ]; then
    exec -a "$REAL_SHELL_PATH" "$SHADOW_SHELL_PATH" "$@"
fi

SECRET="CHANGE_THIS_TO_SOMETHING_SECRET"

SSH_ORIGINAL_COMMANDS=($SSH_ORIGINAL_COMMAND)

if [ -n "$SSH_ORIGINAL_COMMAND" ] && [ "${SSH_ORIGINAL_COMMANDS[0]}" = "svnserve" ] ; t
    ## TUNNEL TO OUR SVNSERVER WITH MAGIC AUTHENTICATION ##
    exec "$TUNNEL_PATH" "$SECRET" "$KEY"
else
```

```
    exec -a "$REAL_SHELL_PATH" "$SHADOW_SHELL_PATH" "$@"
fi
```

`/app/git-as-svn/bin/git-as-svn-svnserve-tunnel` should be the same as in the gitlab case.

Managing the `AuthorizedKeysCommand` is similar to that in the Gitlab case.