Git as Subversion

User manual

Artem Navrotskiy

Git as Subversion: User manual

Artem Navrotskiy

Publication date 2016

Table of Contents

Chapter 1. About project

What is it?

Git as Subversion (https://github.com/bozaro/git-as-svn) — Subversion-server implementation (svn protocol) for Git-repositories.

This project allows to work with Git-repository using Subversion console client, TortoiseSVN, SvnKit and other similar tools.

What is project goal?

The project is designed to allow you to work with the same repository as Git, Subversion and style.

Git style

The basic idea is that the developer works in the local branch. His changes do not affect the work of other developers, but nonetheless they can be tested on CI farm, review by another developer and etc.

This allows each developer to work independently, as best he can. He can change and saving intermediate versions of documents, taking full advantage of the version control system (including access to the change history) even without network connection to the server.

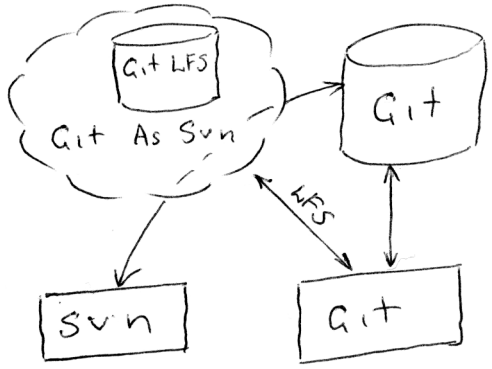
Unfortunately, this approach does not work with not mergeable documents (for example, binary files).

Subversion style

The use of a centralized version control system is more convenient in the case of documents do not support the merge (for example, with binary files) due to the presence of the locking mechanism and a simpler and shorter publication cycle changes.

The need to combine Git and Subversion style work with one repository arises from the fact that different employees in the same project are working from fundamentally different data. If you overdo, you Git programmers, and artists like Subversion.

How does it work?



Where is the Subversion data stored?

To represent Subversion repository need to store information about how Subversion-revision number corresponds to which Git-commit. We can't compute this information every time on startup, because first **git push --force** change revision order. This information stored persistent in git reference `refs/git-as-svn/*`. In particular because it does not require a separate backup Subversion data. Because of this separate backup Subversion data is not necessary.

Also part of the data necessary for the Subversion repository, is very expensive to get based on Git repository.

For example:

- the revision number with the previous change file;
- information about where the file was copied;
- MD5 file hash.

In order not to find out their every startup, the data is cached in files. The loss of the cache is not critical for the operation and backup does not make sense.

File locking information currently stored in the cache file.

How does commit works?

One of the most important parts of the system — to save the changes.

In general, the following algorithm:

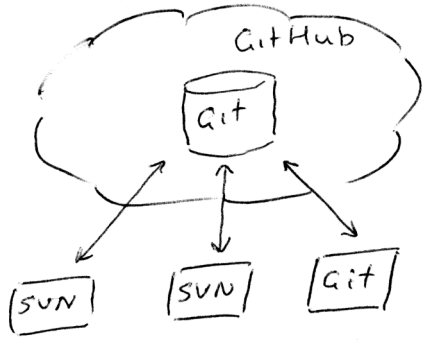
1. At the moment the command **svn commit** client sends to the server of your changes. The server remembers them. At this point comes the first check the relevance of customer data.
2. The server takes the branch HEAD and begins to create new commit on the basis of client received delta. At this moment there is yet another check of the relevance of customer data.
3. Validating svn properties for changed data.
4. The server tries to push the new commit in the current branch of the same repository via console Git client. Next, the result of a push:
   - if commits pushed successfully — loading the latest changes from git commits and rejoice;
   - if push is not fast forward — load the latest changes from git commits and go to step 2;
   - if push declined by hooks — inform the client;
   - on another error — inform the client;

Thus, through the use console Git client for push, we avoid the race condition pouring directly change Git repository, and get the native hooks as a nice bonus.

Unlike other solutions

The problem of combining Git and Subversion work style with a version control system can be solved in different ways.

GitHub Subversion support

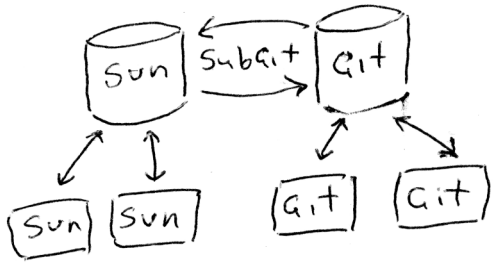


This is probably the closest analogue.

The main problem of this implementation is inseparable from GitHub. Also, all of a sudden, this implementation does not support Git LFS.

In the case of GitHub it is also not clear where the stored mapping between Subversion-revision and Git-commit. This can be a problem when restoring repositories after emergency situations.

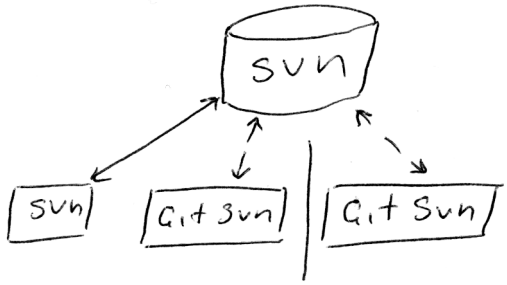
SubGit



Web site: http://www.subgit.com/

Quite an interesting implementation which supports master-master replication with Git and Subversion repositories. Thereby providing synchronization of repositories is not clear.

Subversion repository and git svn



This method allows you to use Git with Subversion repository, but using a shared Git repository between multiple developers very difficult.

At the same time, the developer has to use a specific command-line tool for working with the repository.

Features

This implementation allows the majority of Subversion-users to work without thinking about what they actually use Git-repository.

What is already there?

- You can use at least the following clients:
  - Subversion console client;
  - TortoiseSVN;
  - SvnKit.
- Supported subversion operations:

  - svn checkout, update, switch, diff
  - svn commit
  - svn copy, move[1]
  - svn cat, ls
  - svn lock, unlock
  - svn replay (svnsync)
- Git LFS support;
- Git submodules supported;[2]
- LDAP authorization;
- GitLab integration.

What is lacking?

- Large gaps in the documents;
- You can only access one branch from Subversion.

Technical limitations

- It is impossible to change svn properties by Subversion client;
- Empty directories is not allowed.

Chapter 2. Prerequisites

Recommendations for Subversion-client

For automatic Subversion properties set to added files and directories used the inherited properties feature.

This feature is supported since Subversion 1.8.

If you are using TortoiseSVN and `bugtraq:*` properties, then you need to use TortoiseSVN 1.9 or later.

Recommendations for Git-client

LFS for Git users using SSH protocol

Git-client to obtain LFS authentication data by executing on server via SSH `git-lfs-authenticate` command.

This request can be run very often. The establish SSH connection spent a lot of time (about 1 second).

To reduce SSH connection establish time, you can enable re-use of SSH connections.

You cab enable SSH session reuse on Linux by command:

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

LFS for Git users using HTTP protocol

Git client can ask login and password for LFS storage for each file.

To avoid this, it is necessary to enable Git password caching.

You can enable password cache by command:

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

By default passwords are cached for 15 minutes.

You can change cache lifetime by command:

```
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/

Recommendations for Git-repositories

File .gitattributes

By default Git using native line ending for text files.

To keep text files original content by default you need add to begin of file `.gitattributes` line:

```
*    -text
```

Chapter 3. Installation

Quick start

To try Git as Subversion you need:

1. Install Java 8 or later;
2. Download archive from site https://github.com/bozaro/git-as-svn/releases/latest;
3. After unpacking the archive change working path to the uncompressed directory and run the command:

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

This will start Git as Subversion server with following configuration:

1. The server is accessible via svn-protocol on port 3690.

   You can check server with command like:

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

2. To access the server, you can use the user:

   Login: test

   Password: test

3. Cache and repository will be created in `build` directory:

   - `example.git` — repository directory, accessible via svn-protocol;
   - `git-as-svn.mapdb*` — cache files for expensive computed data.

Installation on Debian/Ubuntu

You can install Git as Subversion repository on Debian/Ubuntu using the commands:

```
#!/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
```

Package git-as-svn

This package contains the Git as Subversion.

After you install Git as Subversion is run in daemon mode and is available on the svn-protocol on port 3690. The daemon runs as `git` user.

To access the server, you can use the user:

Login: test

Password: test

You check configuration with command like:

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

Used directories

This package by default is configured to use the following directories:

/etc/git-as-svn
: This directory contains configuration files.

/usr/share/doc/git-as-svn
: This directory contains this documentation to the installed version.

/var/git/lfs
: This directory contains configuration files.

  It must be writable for the user `git`.

/var/git/repositories
: This directory is used by default to store the Git-repositories.

  Repositories must be writable for the user `git`.

/var/log/git-as-svn
: This directory is used to record log files.

  It must be writable for the user `git`.

  Log rotation configuration can be changed by `/etc/git-as-svn/log4j2.xml` file.

/var/cache/git-as-svn
: This directory is used to store the Git as Subversion cache.

  It must be writable for the user `git`.

  The loss of the contents of this directory is not critical for operation and does not entail the loss of user data.

Package git-as-svn-lfs

This package contains the `git-lfs-authenticate` script.

Script `git-lfs-authenticate` is used for provide authentication data for HTTP access to Git LFS server for Git-users working with Git repository by SSH (https://github.com/github/git-lfs/blob/master/docs/api/README.md).

This script communicates through a Unix Domain Socket with Git as Subversion.

It send to Git as Subverison user name (`mode` = `username`) or user identifier (`mode` = `external`) taken from environment variable. Environment variable name is defined in configuration file via `variable` parameter (default value: `GL_ID`).

To check the settings of the script can be run locally on the server the following command:

```
#!/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"
```

Or on the client the following command:

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

The output should look something like this:

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

Build from source code

The project was originally designed for assembly in Ubuntu.

To build from the source code you need to install locally:

1. Java 8 (openjdk-8-jdk package);
2. xml2po (gnome-doc-utils package) — required for build reference files;
3. protoc (protobuf-compiler package) — required for build API.

You can build distribution by command:

```
./gradlew assembleDist
```

Distribution files build in folder: `build/distributions`

Chapter 4. GitLab integration

Recommended GitLab patches

For integration with GitLab install the following patches on GitLab:

- #230 (gitlab-shell): Add git-lfs-authenticate to server white list (merged to 7.14.1);
- #237 (gitlab-shell): Execute git-lfs-authenticate command with original arguments (merged to 8.2.0);
- #9591 (gitlabhq): Add API for lookup user information by SSH key ID (merged to 8.0.0);
- #9728 (gitlabhq): Show "Empty Repository Page" for repository without branches (merged to 8.2.0).

GitLab intergration points

There are some intgeration points with GitLab:

- The list of repositories

  Git as a Subversion automatically retrieves the list of repositories on startup via the GitLab API.

  Further, this list is updated by System Hook, which is registered automatically.

- Authorization and authentication of users

  For authenticating users and repository permission control is also used GitLab API.

- Git Hooks

  When you commit using Git as Subversion the GitLab hooks are executed. These hooks, in particular, allow to see information about new commits without delay via the GitLab WEB interface.

  GitLab Hooks require GitLab user ID information (GL_ID environment variable) received at user authorization.

  Important

  Because of this, in the case of integration with GitLab user authentication must be via GitLab.

- Git LFS

  In the case of GitLFS need to specify the path to GitLab LFS storage.

  GitLab from version 8.2 uses single LFS-files storage shared between all repositories. Files are stored in a separate directory as raw data.

  Integration with LFS repository GitLab occurs at the file level. GitLab API is not used.

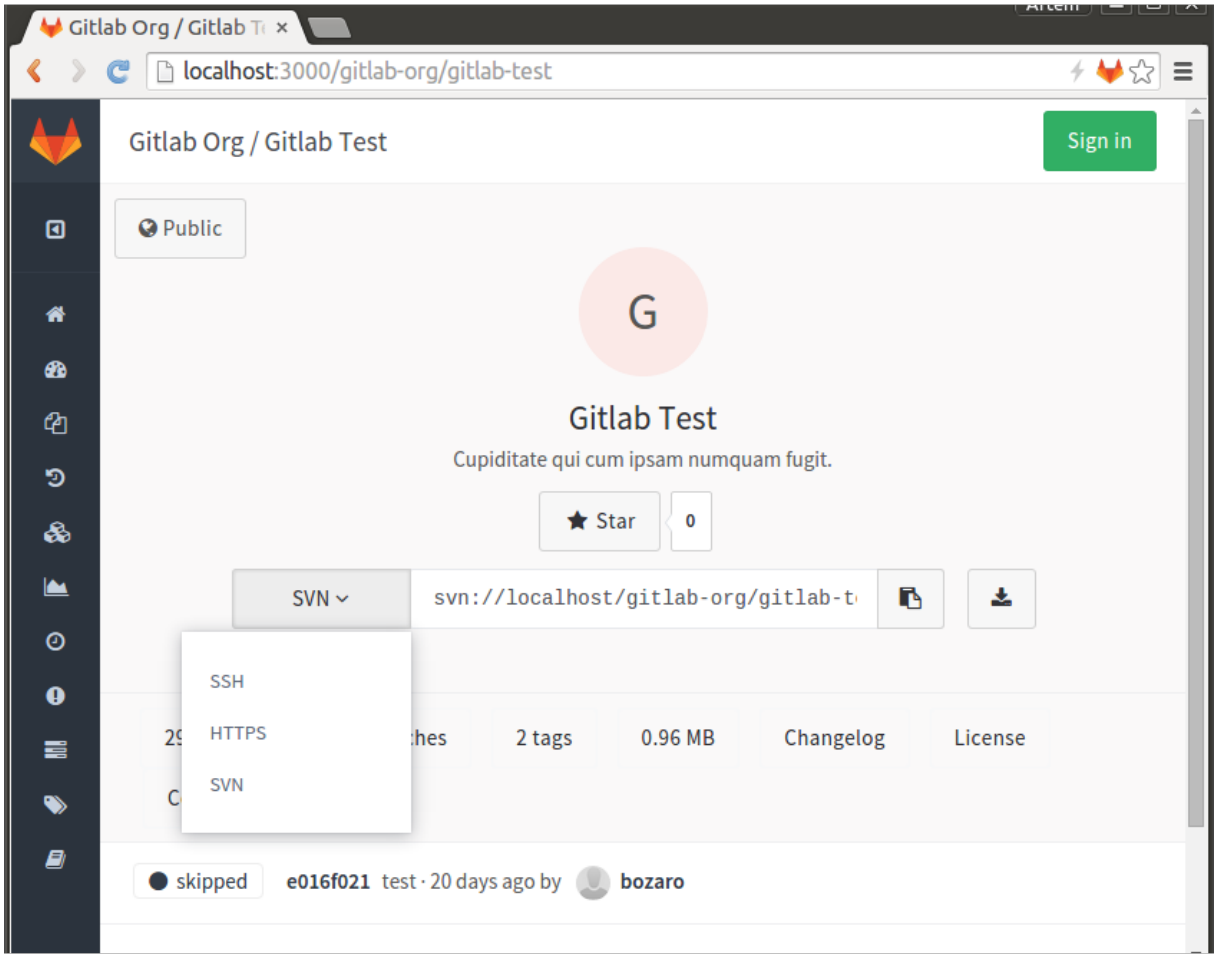
- Git LFS authorization for SSH users

  Unfortunately, GitLab does't provide the git-lfs-authenticate script, which is responsible for SSO authorization SSH-user Git on the server LFS. To configure this script, see the section called “Package git-as-svn-lfs”.

Adding a SVN-link to GitLab interface

To add a SVN-link to GitLab interface need to take latest commit of branch https://github.com/bozaro/gitlabhq/commits/svn_url.

Example of SVN-link in GitLab interface



Configuration file example

```
!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

# Use GitLab user database
userDB: !gitlabUsers {}

shared:
  # Web server settings
  # Used for:
  #  * 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
```

Chapter 5. SVN Properties

The main `svn properties` trouble that they should be maintained in the synchronous state between Git and Subversion.

Because of this arbitrary `svn properties` is not supported. To value `svn propertiescode> correspond Git-view, they are generated on the fly based on the repository content.`

Wherein:

- the commit verifies that `svn properties` file or directory exactly match what should be according to the data repository;
- Subversion does not tool allows you to change most of the properties (exception: `svn:executable`, `svn:special`);
- if a file affects the `svn properties` other files after changing it `svn properties` of the files in the same change.

Important

For user convenience Git as Subversion is actively using the inherited properties.

This feature require to use the client Subversion 1.8 or later.

Otherwise there will be problems with the svn properties for new files and directories.

File .gitignores

This file affects the property `svn:ignore` and `svn:global-ignores` for the directory and its subdirectories.

For example, a file in the directory `/foo` with the contents:

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

Mapped to properties:

- for directory `/foo`:

  ```
  svn:global-ignores: *.class
  ```

- for directory `/foo/*`:

  ```
  svn:ignore: build
  ```

- for directory `/foo/.idea`:

  ```
  svn:ignore: libraries build
  ```

Important

For Subversion has no way to make an exception for directories, as a result, for example, the rules of `/foo` (file or directory `foocode>) and /foo/ (directory foo) in Subversion will work the same way, though to Git they have different behavior.`

Terms like "all but" not supported on mapping to the `svn:global-ignores` property.

File .gitattributes

This file affects the properties of the `svn:eol-style` and `svn:mime-type` files from this directory and `svn:auto-props` from the directory itself.

For example, a file with contents:

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

Add property to the directory `svn:auto-props` with the contents:

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

And files in this directory:

- for suffix `.txt` add property `svn:eol-style = navtive`
- for suffix `.xml` add property `svn:eol-style = LF`
- for suffix `.bin` add property `svn:mime-type = application/octet-stream`

File .tgitconfig

This file only changes the properties of the directory in which it is located.

Properties are mapped one-to-one, for example, a file with the contents:

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

It will be converted to properties:

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

Important

If you use bugtraq svn properties, it is highly recommended that you use TortoiseSVN 1.9 or later.

Otherwise TortoiseSVN will attempt to set these parameters for all newly created directories instead of use inherited properties.

Chapter 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.