Git and SVN combined workflow

How to contribute to Tiki using Git, please see Git Workflow instead.

About
This page documents the Git and Subversion (SVN) interoperability.

Status
To check if SVN repo is in sync with the latest from GitLab:

- https://sourceforge.net/p/tikiwiki/code/HEAD/log/?path=/trunk
- https://sourceforge.net/p/tikiwiki/code/HEAD/log/?path=/branches/24.x
- https://sourceforge.net/p/tikiwiki/code/HEAD/log/?path=/branches/23.x
- https://sourceforge.net/p/tikiwiki/code/HEAD/log/?path=/branches/21.x

Workflow
Examples

Let’s assume someone makes a merge request on our GitLab mirror (which is synced auto-magically from sf.net SVN via git.tiki.org). Here are some steps and examples of commands which are necessary to make:

1.1. Review the merge request

The open merge requests are by default listed on https://gitlab.com/tikiwiki/tiki/merge_requests (you can see all here).

If you are listed on https://gitlab.com/tikiwiki/tiki/project_members as "Owner", "Maintainer" or "Developer" you should be able to handle that.

Please review if the commit message is clear and in the good format (you can edit that, if not), check the CI / CD pipelines passed (e.g. if the PHPCS check did not fail) and the code looks good to you. Also beware if the source branch (the fork) of the merge request is not too much behind the upstream. If it is more than 10-20 commits behind it will be hard (if not impossible) to review and merge due to many conflicts. Ask the participant who requested the merge to update (rebase) and re-test. (See the discussion at https://gitlab.com/tikiwiki/tiki/merge_requests/9 for example.)

Best is if you can test the fork on your local copy before merging but if it is only a simple change and the code (diff) is easy to understand you can proceed to the next step.

1.2. Approve and/or merge the request directly

If everything looks great you can use "Add approval" button (in case you are just a reviewer but e.g. someone else should handle the merge and sync with git.tiki.org) or the "Merge" button directly.

For example this merge request was reviewed and merged by Fabio: https://gitlab.com/tikiwiki/tiki/merge_requests/6

1.3. When not suitable to merge, Close and/or discuss

For example this merge request was rejected and closed by Marc: https://gitlab.com/tikiwiki/tiki/merge_requests/10

1.4. When the request is merged

The merge request was approved and merged to our Gitlab repository. All good. But to sync that to our SVN now some command line manual actions are needed to sync the merged code from Gitlab to git.tiki.org and sf.net (SVN).

1.4.1. Prerequisites

The next steps assume you are familiar to work with terminal command line, you have git installed locally and they require git push access on the git.tiki.org server (using your public ssh-rsa key) and your user e-mail record in the tiki.git/subgit/authors.txt file to map your user between Git and SVN for SubGit. It is also good if there is the contributor's username and email mapping added to that file, so the authors get properly credited - otherwise the commit will go to a "shelve" and will be committed under your own username to the svn.
1.4.1.1. Clone git.tiki.org

If you haven't done that yet, you must clone git.tiki.org to your local. Cloning whole history and branches would take lot of time and space (more than 3GB!) however. To save time and disk space you can get a "shallow clone" with the `--depth` parameter:

```
$ git clone --depth 10 --branch master git@git.tiki.org:tiki.git
```

For more details and options see Git Usage!

Now `cd` to your freshly cloned directory...

1.4.1.2. Add gitlab mirror as remote

**Add a remote to track GitLab** (this needs to be done only once) on your local git repository (cloned from `git@git.tiki.org:tiki.git`):

```
$ git remote add gitlab git@gitlab.com:tikiwiki/tiki.git
```

1.4.2. Pull it from GitLab.com

```
$ git pull gitlab master
```

Example output:

```
~/Documents/Devel/Tiki/git.tiki.org/tiki (master)
$ git pull gitlab master
Warning: Permanently added 'gitlab.com,35.231.145.151' (ECDSA) to the list of known hosts.
remote: Enumerating objects: 46, done.
remote: Counting objects: 100% (37/37), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 19 (delta 15), reused 6 (delta 3)
Unpacking objects: 100% (19/19), done.
From gitlab.com:tikiwiki/tiki
 * branch               master     -> FETCH_HEAD
   cebffa7d259..ecde867d498  master     -> gitlab/master
Updating d1e367fb3e3..ecde867d498
Fast-forward
   lib/importer/tikiimporter_blog_wordpress.php | 8 +++++++++-
   tiki-importer.php                      | 1 +
2 files changed, 8 insertions(+), 1 deletion(-)
```

1.4.3. Push it to git.tiki.org

```
$ git push origin master
```

Example output:
That should do it! You can now check and see the Git commits merged and synced to the SVN repository.

Username mapping problems

Sometimes, we have users pushing on git.tiki.org that is not a registered user on SourceForge. It may broke subgit synchronization. As a support of it, a helper script is providing fallback username to commits. It tries to create a fallback username based on commiter email or it creates an user and email based on SVN username.

This script is available at [https://gitlab.com/tikiwiki/systools/blob/master/subgit/authors.py](https://gitlab.com/tikiwiki/systools/blob/master/subgit/authors.py). Instead providing a TXT file mapping on SubGit configuration, this script should be set and the TXT mapping will be argument for this script.

The other problem is that it is common for feedback to be sent to the SVN Mailing List, and if the commit isn't associated to properly set up SourceForge account, the mail will bounce. Ex.: for r69454, I got:

```
Your message wasn't delivered to "guill.bout"@users.sourceforge.net because the address couldn't be found, or is unable to receive mail.
```

Discussion

Below is discussion that led to the implementation of this workflow.
Goals

- Allow developers without SVN commit access to send pull requests using Git and Github GitLab
- Keep current SVN workflow working as is, without the need to update all release scripts or change developer habits (Avoid the need of all the work involved in What ToDo When Migrating To Git)
- Allow developers to migrate to Git as they feel comfortable
- Allow work on security issues to remain secret until release
- Be implemented immediately, having a minimum of extra operational work to keep everything synced
- Take into account upcoming Continuous Integration system, which is based on Git and is running behind the scenes and being refined as of 2018-02-01
- Allow a gradual migration from SVN to Git

Once implemented the proposed workflow will be This workflow is in use until we are able to abandon SVN. Once we have a fully Git-based workflow, we can change the development workflow and choose the best place for our central code repository.

Ideas on a full Git workflow are being discussed at Choosing the Git Workflow

Implementation steps

- Using Subgit, we setup a Git Repo in our own server (eg: git.tiki.org).
- For every user who already have SVN commit access and prefers to commit to the Git Repo, we give write access to the Git Repo.
- We reuse our github.com/tikiorg/tiki project and accept pull requests from non-trusted committers there.
- Subgit will keep SVN Repo and Git Repo synchronized and will push commits from both sources to Github automatically, but won't handle commits coming from pull requests at Github.
- Developers who accept pull requests will have to fetch changes from Github and push to Git Repo manually. This is simple standard operation, unless a conflicting change happens between accepting the pull request and running a shell command.
- Contributor authorship is preserved in both Git and SVN logs.
- We also setup a Security Git Repo, that is a clone of our Git Repo, and is synced manually by developers. This repository is private, and for each security issue we create a new branch. Developers who work on security issues will have to work with Git. Once branch is ready, we merge to master at Security Repo and push changes to Git Repo. This way we keep code secret until ready to be released.

Constraints

- We need Subgit, which is not Free Software. But we have got a free license, since our project is Open Source.
- Our main Git repository has to be installed on a server in which we have access to the filesystem and permission to install Subgit. So, we'll need to manage developer permissions there, maybe manually asking for SSH keys. This has to be like this until we abandon SVN.
- Any changes made directly to GitLab repository (eg: Pull Requests) have to be manually pushed to main Git repository by an authorized developer.
Roundtable meeting about GIT and follow up in the devlist


Questions

- Should we direct developers to use Git branches for Experimental branches
- What about email notifications for work on Git branches?
  - Once changes are merged in, they will be in SVN and be in email notifications
- Could this help with Continuous upstream?
- Do we want to create documentation for installation on different environment (tricks and tips page) ?

IE: Under certain environment where you don't control your server specific commands are required to install a Tiki clone. Like a domain or subdomain folder is needed and not empty - Tiki clone create a directory and doesn't work on existing non-empty directory.

- You need to create your clone in a new directory
- To move your git files in the domain/subdomain directory
- Enter the directory and apply a few git command

  mv tikiclone/.git subdomain.domain.com/.git

  git remote show origin //to be sure

  git checkout 20.x //to be sure

  git reset --hard

  *may be different depending of the hosting setup*

Todo

- Document a Git workflow for the average project, which will now use version control for a theme and a few patches. With SVN, maintaining a few local patches was easy as svn up would handle merging fine, but local theme was unversioned, which has been sub-optimal, especially when dealing with multiple instances, like development, fail-over, staging and development.

Opinions

- I think this is a very sensible approach. Thank you Luis.
Related links

- https://gitlab.com/esr/reposurgeon
- Backport changes using Git