Tiki Manager

**Executive Summary:** "Tiki Manager" is a TRIM 2.0

The "Tiki Remote Instance Manager (TRIM)" will go into maintenance mode, and the code will be forked and revamped to become its replacement, now known as the "Tiki Manager". This is in conjunction with work on Best practices for a local Git repository and interacting with the Tiki community.

Below is the plan for 2019.

**Details:**
Over 10 years ago, the Tiki Remote Instance Manager (TRIM) was first released: https://sourceforge.net/p/tikiwiki/code/12467/

It has quite a few features:

- TRIM
- TRIM use cases
- TRIM Test Plan

It empowers some pretty cool things: http://wikisuite.org/TRIM-and-Syncthing-for-automated-offsite-backups

We have been talking about revamping TRIM for a while:

- TRIM Revamp

Jorge (Xorti), Ricardo and Marc have discussed and agreed on a general plan going forward:

At a high level:
1- TRIM goes in maintenance mode. Just security and bug fixes.
2- A new project starts: "Tiki Manager". In reality, "Tiki Manager" is pretty much a TRIM 2.0

TRIM source remains here: https://sourceforge.net/p/tikiwiki/code/HEAD/log/?path=/trim

Tiki Manager will live here: https://gitlab.com/tikiwiki/tiki-manager

Sources from SVN are copied, and thus history is kept, and major changes will be done there.

It will based on all the most modern tech:

- Symfony 4.1: https://packagist.org/packages/symfony/console
  - And thus, minimum version: PHP 7.1, but some parts of the code will still be PHP 5.3 because they need to connect to a remote server and run some old code to manage older Tiki instances.
- The GitLab Continuous Integration (CI) to test under diverse platforms.

No changes

- Data format remains SQLite. Just move your data directory to Tiki Manager and it will work.
- Backup format remains the same. Backups from TRIM will be restorable via Tiki Manager.
Stuff that will disappear
- GNU Make related code
- WordPress installer (wasn't maintained)
- CVS to SVN converter

Documentation
https://doc.tiki.org/TRIM will remain as is, and a new page will be made: https://doc.tiki.org/Manager and relevant docs will be copied over.

Many commands will be similar. Ex.:
- make upgrade -> tiki-manager.php upgrade
- make backup -> tiki-manager.php backup
- make editinstance -> tiki-manager.php editinstance

But some will change. Ex.:
- make instance -> tiki-manager.php makeinstance or perhaps tiki-manager.php addinstance

Developer coordination and community support
We will start an XMPP chat room: tiki-manager@conference.wikisuite.chat

Upcoming features
- Running on Windows. This didn't work very well: TRIM on Cygwin
- Run any tiki-console.php command
  - Install / upgrade Tiki Packages
  - themes
  - etc.
- Be the base of a future ClearOS app. So Tiki Manager will installs Tiki instances, and also send instructions to the ClearOS API (ex.: desired PHP version, SSL certificate, etc.). We'll make this generic so it can send ad hoc commands to other systems.
- Manage dev-staging-production workflow
- REST API
- Bootstrap 4 web interface: tiki-download_wiki_attachment.php?attId=210
If easy, Activity indicator when doing long operations

Long term roadmap

- Alerts for failure of an unattended operation (via email or to monitoring system)
- item5930-TRIM-make-compare
- Packaged as a phar: Phar
- Be ported to other distros: Tiki Manager Packaging
- To become the base for a next generation show server:
  - [https://tiki.org/Presentation-Fosdem-2014-show.tiki.org](https://tiki.org/Presentation-Fosdem-2014-show.tiki.org)
    - For the Tiki community and to permit community members to run their own
    - And later, to be able to evolve into [http://wikisuite.org/Orchestrator](http://wikisuite.org/Orchestrator)
- See "To embed in a Tiki (so a Tiki can become a management tool for other Tikis)" section on TRIM Revamp

Questions with answers

If TRIM works so well, why make these architectural changes?

1. TRIM works well on GNU/Linux. But we want it to be more portable and work on just about any platform
2. TRIM is based on a design from a decade ago. The new infrastructure takes advantage of today's technology, and makes it easier to deploy, test, add features, etc. Please see: [https://en.wikipedia.org/wiki/Technical_debt](https://en.wikipedia.org/wiki/Technical_debt)

Why not use and improve an existing generic solution?

We did take a good look at the alternatives: Generic PHP app deployment tools which are written in PHP
Of course, they have some nice things, but they are also missing things TRIM does well. The fact that TRIM was built for Tiki gives it a definite advantage.

Why not a full rewrite?

"Each of these bugs took weeks of real-world usage before they were found. The programmer might have spent a couple of days reproducing the bug in the lab and fixing it. If it's like a lot of bugs, the
fix might be one line of code, or it might even be a couple of characters, but a lot of work and time went into those two characters.

When you throw away code and start from scratch, you are throwing away all that knowledge. All those collected bug fixes. Years of programming work.


Why all the disruption? Why not just create an SVN branch and use trunk for evolution?

- A new name (Tiki Manager)
- A new base (Symfony Console)
- A new SCM too (Git)
- To be able to start with latest and greatest, to be able to innovate and deprecate things, without disrupting existing users.

Why change the name?

To be more descriptive. An acronym is not great. https://tiki.org/forumthread68945-Possible-rename-of-TRIM-which-stands-for-Tiki-Remote-Instance-Manager-to-just-Tiki-Manager

Why Tiki Manager and not something else?

We considered many options: Tiki Manager Packaging

Why based on Symfony Console?

- It's great
- It's popular. 100M installs on https://packagist.org/packages/symfony/console
- We already use in Tiki, and we are very happy with it: Console

Will it be Git only?

Yes, managed on Git. The Git and SVN combined workflow will not be available. Why? Since we are starting fresh, and it only affects a smaller number of devs.

When?

The first version of Tiki Manager will be released in December 2018 or January 2019

- Work had already been done to convert to Symfony but with the idea of more an evolution than a revolution. We now need to review that code, and pull out some stuff.

How long will TRIM be maintained?

At least one year after Tiki Manager is stable, to permit a smooth transition

PHP 5.6 and 7.0 support ends in December 2018:
http://php.net/supported-versions.php

After copying TRIM to GIT as a seed to the new Tiki Manager, the changes related with the console introduction, etc. will be reverted, so that the version in SVN will go back to work in PHP 5.3 / pre-Symfony console. Ex.:

- http://sourceforge.net/p/tikiwiki/code/66751
Questions without answers yet

What will be the lifecycle of Tiki Manager?
Context: TRIM has evolved for a decade as one single version, which you got directly from source control. No branches, no releases. Just svn up to get latest version.

For Tiki Manager, we could continue with the same model or change.

The people who get involved in the project (hint hint, you are invited) will discuss such topics and shape the future, and answer questions such as:

- Will there be regular releases?
- Will there be Long Term Support (LTS) versions?
- When will requirements change? Ex.: to require PHP 7.2?

Will Tiki Manager fetch sources directly from Git?
Answer: Most probably

The idea is to fetch code (for example) from https://gitlab.com/tikiwiki/tiki/tree/19.x instead of https://sourceforge.net/p/tikiwiki/code/HEAD/tree/branches/19.x/

Since Tiki Manager will be managed on Git, and the initial way to get it will be via Git, this would make sense. It would avoid having to have SVN as a dependency of Tiki Manager. It would also make it easier to get your Tiki from a local Git repo.

We have an old CVS to SVN converter: https://doc.tiki.org/TRIM#make_convert
Maybe we need to create a SVN to GIT one now. Not clear yet if and how we'll do this.


What about projects with their own Git repository?
Good question. We are thinking about that:

- Best practices for a local Git repository and interacting with the Tiki community

How integrated will Tiki Manager be with Tiki?
We don't know yet. We'll see how things evolve in coming years. We don't need to decide now.

It's important that Tiki Manager always be able to run in standalone mode, like Tiki Check. But as time goes by, there should be more and more code sharing as they'll both use Symfony Console

Tiki Manager could be bundled in Tiki, or available via Packages

Will atomic upgrades be supported?
Answer: Maybe one day

See atomic upgrade

How will the various Tiki instances be connected?
This is something that is being improved in Tiki, but not specifically connected to Tiki Manager. Basically, let's continue to improve this, but it's not dependent or a blocker to Tiki Manager's roadmap.
Sync dev/prod Tiki instances - admin tool to show differences between configuration, trackers, fields and wiki pages
http://sourceforge.net/p/tikiwiki/code/67442

Could we / should we run some tests?
The vanilla Tiki is tested by the community Continuous Integration system, but we should run tests on what Tiki Manager is deploying (which could be different, with patches and all)

Could we / should we request that the monitoring system check that all is OK?
After an upgrade, we typically should check that all is OK. Since it's OK most of the time, we have verification fatigue. We could automate this.
We (Fabio, Rodrigo and Marc) are testing Zabbix.

Todo
A method to combine / deploy specific branches with patches, a theme, system configurations (tiki.ini), etc.
Let's brainstorm and then find a technical solution for Tiki Manager to handle the following use case.

I want a specific Tiki branch / fork: could be standard upstream Tiki, or my development fork (ex.: a feature fork)

- With n patches/merge requests/backports of specific commits
  - Ex.: https://gitlab.com/tikiwiki/tiki/merge_requests which I may want to test, or use in production until it's merged in
  - Could be "permanent" patches we need to maintain because of the specific infrastructure of the project. Similar to what we do to patch Composer dependencies: https://github.com/cweagans/composer-patches
  - Specific commits in other branches (or later in same branch). Ex.: I want to test a fix from trunk

- A theme which is managed in another Git repo
- Other adjustments which could be outside the web root (ex.: tiki.ini)
  - Although here, we may want to unify the code base and use environment variables
- Some profiles applied
  - Ex.: to test some common profiles with a proposed merge request or to set up a demo site (ex.: Tiki for Intranets demo): item1513-OpenSourceCMS-type-demo-to-test-develop-and-show-off-profiles

And to run the usual tests (unit tests, syntax validation for target platform, etc.) on all this. Combining merge requests to trunk with a base branch 15..x is risky (PHP syntax may have changed) and folks that would use this are supposed to be aware of the challenges.

In TRIM make update/upgrade, we already have the concept of "do a dry run and abort if there is a conflict". This is needed here as well.

alias
- Tiki Manager