Git workflow for versioning local changes

Table of contents

- General concept
- Set up the whole environment
  - Create empty local repository
  - Clone project for DEV
  - Rename repos
    - Fill local repo (origin)
    - Set local GIT repo as default repo
    - Set up production environment: PROD
- Make local changes in DEV and get them in PROD
  - Make a sample change in DEV
    - Update PROD with the new change
- Get new improvements from upstream
  - Update DEV
  - Update PROD
  - TODO

General concept

This is not the simplest use-case but it is very common for people who want a reasonable guarantee that their production Tiki will work with no regression, and who want to be able to use versioning for rolling back in case anything is wrong.

We have two environments (or more).

- DEV is a Tiki server for improving the Tiki site. It may break sometimes before it gets better. It gets updates from the official Tiki repo (+upstream+) and it pushes updates and local changes to the local repo (+origin+).
- PROD is the production Tiki which should only be updated from origin after everything was tested on DEV or on another server like a preprod or acceptance Tiki instance. preprod or acceptance are out of the scope of this document and will not be mentioned again.

Let's imagine a Tiki site running Tiki 22.x

Set up the whole environment

Create empty local repository

This will hold a full Tiki repo, plus local changes such as custom theme, custom templates, logos and favicons, local improvements not yet contributed to the community or contributed in another branch (like contributed in master while running 22.x)

For instance let's create a local_repository repo on gitlab with options:
x Public
x Not initialised

The URL is: [git@gitlab.com:xxxx/local_repository.git](git@gitlab.com:xxxx/local_repository.git)

Clone project for DEV

Get a full repo (yes, it's big) from official sources.
dev:~$ git clone https://gitlab.com/tikiwiki/tiki.git dev_tiki22

We now have a full repo in folder `dev_tiki22` (place folder as suits your local environment)

**Rename repos**

```
dev:~/dev_tiki22$ git remote
origin

dev:~/dev_tiki22$ git remote rename origin upstream

dev:~/dev_tiki22$ git remote add origin git@gitlab.com:xxxxx/local_repository.git

dev:~/dev_tiki22$ git remote -v
origin  git@gitlab.com:xxxxx/local_repository.git (fetch)
origin  git@gitlab.com:xxxxx/local_repository.git (push)
upstream        https://gitlab.com/tikiwiki/tiki.git (fetch)
upstream        https://gitlab.com/tikiwiki/tiki.git (push)
```

**Fill local repo (origin)**

```
dev:~/dev_tiki22$ git fetch --all
Fetching upstream
Fetching origin

dev:~/dev_tiki22$ git push -u origin --all
Counting objects: 549383, done.
Delta compression using up to 12 threads.
Writing objects: 100% (549383/549383), 1.32 GiB | 7.90 MiB/s, done.
remote: Resolving deltas: 100% (430804/430804), done.
remote: Checking connectivity: 549383, done.
To gitlab.com:xxxxx/local_repository.git
  * [new branch] 22.x -> 22.x
Branch '22.x' set up to track remote branch '22.x' from 'origin'.
```

**Set local GIT repo as default repo**

This section is unfinished. Help would be appreciated.

We want that checkouts and pull and push default to `origin` instead of `upstream`

```
This is wrong

dev:~/dev_tiki22$ git checkout 21.x
Branch '21.x' set up to track remote branch '21.x' from 'upstream'.
Switched to a new branch '21.x'

dev:~/dev_tiki22$ git push -u origin --all
Branch '22.x' set up to track remote branch '22.x' from 'origin'.
Branch '21.x' set up to track remote branch '22.x' from 'origin'.
```
All is fixed now but creating a new branch sets it to upstream again instead of origin. This is an accident waiting to happen (pushing local stuff on the official Tiki repository). any idea?

<table>
<thead>
<tr>
<th>dev:$/dev_tiki22$ git checkout 19.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch '19.x' set up to track remote branch '19.x' from 'upstream'.</td>
</tr>
<tr>
<td>Switched to a new branch '19.x'</td>
</tr>
</tbody>
</table>

We need to remember to run

```
git push -u origin --all
```

Set up production environment: PROD

We can save space by installing a light source version with no history.

```
prod$ git clone --depth=1 --branch=22.x git@gitlab.com:xxxxx/local_repository.git tiki_prod_22
```

Make local changes in DEV and get them in PROD

Make a sample change in DEV

```
dev:$/dev_tiki22$ cp templates/tiki-listpages.tpl themes/templates
dev:$/dev_tiki22$ git status
On branch 22.x
Your branch is up to date with 'origin/22.x'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        themes/templates/tiki-listpages.tpl

dev:$/dev_tiki22$ git commit themes/templates/tiki-listpages.tpl -m "First local modification"
dev:$/dev_tiki22$ git push
```

Update PROD with the new change

```
development_tiki22$ git pull
```

Get new improvements from upstream

Update DEV

```
development_tiki22$ git pull --rebase --autostash
```

(needs testing and confirmation)
Update PROD

After some testing in DEV, update PROD

```
prod:~/repo_prod_22$ git pull
```

TODO

Use tags for updating PROD. That would make rolling back PROD easier.