Offline editing via git

Offline editing via git

amusewiki.org

Contents

If the filename has is composed by more words (separated by a dash), the target subdirectory will be the first character of the filename and the first character of the second word of the filename. The rationale for this is that usually you want URIs in the form firstname-lastname-title, so there is an high chance to put all the files from the same author in the same directory.

E.g., john-doe-the-title have to go into the `j/jd` directory, as `j/jd/john-doe-the-title.muse`

If the filename has only one word, then the second character used for the subdirectory will be the *last* character of the filename. E.g. `t/tg/testing.muse`

In all the files, the `#title` header line is mandatory.

Images

Images belongs to the same directory as the corresponding text. It's highly recommended to use a prefix to make it clear to which subdirectory it belongs, to reduce the chance of clashes when inserting them manually.

E.g., a file attached to the normal page `testing.muse` could have a name like `t-g-image-1.png` or `t-g-image-2.jpg`. Only PNG and JPG should be attached.

PDFs attached to the page (via `#ATTACH filename.pdf` header), must be placed into the `uploads` directory.

Naming convention in the archive tree

Creating new pages while via git is fully supported, but you have to pay attention to the naming convention, otherwise the texts will not show up.

Anyway, the recommended way to create files and attachments, it's to use the web interface, because the site is able to avoid conflicts and it's supposed to know where to place files. Also, if you're importing, you can use the HTML converter and save quite a few effort.

So the local editing is more useful for *editing* rather than uploading new texts. You could, e.g., import, add the images (with some placeholders in the text), place a `#DELETED wip` header line, commit, and resume the editing locally.

Filenames

The maximum length is 95 characters in the range, ASCII letters and digits only, separated by a single dash. No leading or trailing dashes. The extension must be `.muse`. E.g. `my-new-page.muse`. The filename is arbitrary and doesn't have to (even if it's recommended) map to the real title set in the header via #title.

Directory

Special page's files go in the `specials` directory, from the root of the archive (1 level down).

E.g. `specials/index.muse`

Normal page's files go in a two level deep directory.

The top-level directory is a single character or digit, using the *first* character of the filename.

Server-side setup for public accessible sites.

Obviously, don't export the private sites.

Make the repositories read-only available via git-daemon.

```
apt-get install git-daemon-sysinit
```

Change /etc/default/git-daemon to read:

```
GIT_DAEMON_ENABLE=true
```

Start the service

```
service git-daemon start
```

Export the repositories

Link the repositories in `/var/lib/git` (as per package documentation).

Something like this should do, assuming the application root is `/var/lib/amusewiki`

```
root@localhost:/var/lib/git# for i in /var/lib/amusewiki/repo/*;
     do if [ -d ''$i/.git'' ]; then echo $(basename $i); \
     ln -s $i/.git $(basename $i).git ; fi ; done
```

And in your application directory, mark the repository for export:

```
$ for i in repo/*/.git ; do touch $i/git-daemon-export-ok; done
```

Editing

Amusewiki fully supports offline editing. This document describes how you can do it. It assumes you have some knowledge about how Git works. The internet is full of tutorials and documentation on this matter.

This assumes that the site's archive is public and accessible. The place to find the git clone URL is usually the CGIT interface, accessible from each text via the recent changes features, and navigating to the root of the repository, e.g. `http://www.amusewiki.org/git/amw/`, where you can find the "Clone" URL.

On your machine, clone it, so you can work on it (this step can also be done via SSH if the site is private):

```
git clone git://git.amusewiki.org/git/amw.git
# or, for private sites
git clone ssh://www.amusewiki.org/var/lib/amusewiki/repo/am
```

Now we have the `amw` repository locally, but we don't know where to push our changes, yet, because we just cloned the working tree of the site, so we cannot and we should not push there directly. Instead, we have to provide another one, let's call it "bridge", where we can push and where amusewiki can pull.

Usually it's enough to set it up on GitLab or GitHub or a normal repository on the same machine where amusewiki resides and where amusewiki can read.

We are going to initialize it and set it to the new origin, keeping the read-only on a branch.

```
# on the server, create the bridge repo
mkdir -p /var/cache/git/marco
cd /var/cache/git/marco
git init --bare amw.git

# on your machine, initialize it
```



```
git remote rename origin web
git remote add origin \
    ssh://www.amusewiki.org/var/cache/git/marco/amw.git
git push -u origin master
# track the upstream on your machine in a branch
git checkout -b upstream -t web/master
```

The site will refuse to accept changes which don't result in a fast-forward, so you have to keep the upstream in a branch (as per last command).

This way, if the site refuses the changes, you have to do:

```
git checkout upstream
git pull
git checkout master
git merge upstream
git push
```

It's recommended to do so each time you start editing locally.

Visit `/console/git` to add your personal repository. If it's a GitLab/GitHub one, you can use the web interface. Otherwise you have to add it manually. Refresh `/console/git`.

```
# as the amusewiki user, add our bridge repository
cd /var/lib/amusewiki/repo/amw
git remote add marco /var/cache/git/marco/amw.git
```

With the "Fetch" button you will pull your changes.

The page itself contains some documentation about the workflow.

After the setup, the workflow strips down to:

```
# locally
git checkout upstream
git pull
git checkout master
git merge upstream
# edit, add, commit, hack, etc.
git push
```

Visiting `/console/git` and fetching from your repo. Merge the upstream again if the fetching fails.