Offline editing via git

Contents

Editing **3**
System user with SSH access 3
Provide a remote reposity 3

Naming convention in the archive tree **4**
Filenames 4
Directory 4
Images 4

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.

The procedure depends if you can access the server where Amusewiki is running with a system user, or if you need to use an external service like GitLab, GitHub, etc.

The instructions can be found at `/console/git`, where you will find the exact paths to use.

System user with SSH access

This is as simple as doing:

```
git clone ssh://amusewiki.org/var/lib/amusewiki/shared/repo/amw.git
cd amw
git pull
# edit, commit...
git push
```

The instance will pull the changes automatically.

Please ensure that your system user is member of the `amusewiki` group.

Provide a remote reposity

Create a public repository on GitLab, GitHub or similar services (or on your own server).

Please initialized it empty, without any README,license, .gitignore or similar.

Get the repository HTTPS URL, e.g. https://example.com/git/test and add it using the form you can find on the `/console/git` page of you Amusewiki instance. You can have one personal repository.

Now we are going to populate it and get it ready:

```
git clone https://amusewiki.org/git/amw
cd amw
git remote rename origin web
# Please replace the example URL in the next line with the real one with SSH protocol
git remote add origin git@example.com/git/test
git push -u origin master
git checkout -b upstream -t web/master
```

And the workflow is:

```
git checkout upstream
git pull
git checkout master
git merge upstream
# edit, commit...
git push
```

The tricky part is that you need to visit `/console/git` and press the ”fetch” button to pull the changes into the site. This doesn’t happen automatically.

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.

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. You need to use a **prefix** to make it clear to which subdirectory it belongs, following the naming convention described in the previous section.

E.g., a file attached to the normal page `my-test-file.muse` should have a name like `m-t-image-1.png` or `my-test-whatever.png`. Only PNG and JPG images should be attached.

E.g.:

```
m/mt/my-test-file.muse
m/mt/my-test-image-1.png
m/mt/my-test-image-2.png
```

or, using the same scheme the web application is using:

```
m/mt/my-test-file.muse
m/mt/m-t-image-1.png
m/mt/m-t-image-2.png
```

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