blohg Documentation

Release 0.12

Rafael G. Martins

November 30, 2013

Contents

Author Rafael Goncalves Martins

Website http://blohg.org/

Source code http://hg.rafaelmartins.eng.br/blohg/

Mailing list blohg@librelist.com

Version 0.12

This is blohg, a full-featured blogging engine, that uses Mercurial (or alternatively Git) as the storage backend.

Here you will find a complete documentation about the usage and the concepts behind blohg.

Have fun, and feel free to contact us, using the mailing list, if you have any questions.

Enjoy!

CHAPTER 1

User's Guide

1.1 About blohg

blohg is a Mercurial (or alternatively Git) based blogging engine written in Python, built on the top of the Flask micro-framework and some of its extensions. All the content of the blogs are stored inside a repository, and its history is used to build the posts and pages.

1.1.1 Motivation

Everybody knows that we have a large number of blogging engines lying around the blogosphere, but there are not a many choices for programmers, who are used to working daily with source-code editors and version control systems and may be more productive when blogging using these tools, instead of the fancy WYSIWYG editors and administration interfaces.

Actually this isn't the first project trying to implement a VCS-based blogging engine, but most of the existing projects aren't tied to a VCS and are just using text files that can be versioned with a VCS, without taking full advantage of the VCS's revision history.

This project uses Mercurial as the VCS, reStructuredText as the markup language and Jinja2 as the template engine. All of these are pretty popular within the Python ecosystem and easy to use.

1.1.2 Basic concepts

Here are some of the basic concepts needed to understand how blohg works: In addition, you should know how Mercurial works, the reStructuredText syntax and the Jinja2 syntax.

Pages

Pages are static content, such as an "About me" page. They aren't listed in atom feeds or in the home view. You may want to create a menu entry manually in the template to them. Pages are `.rst` files stored in any directory inside `content/`, excluding `content/post/`. Sub-directories are allowed.

Posts

Posts are the dynamic content of a blog. They are shown on the home page and atom feeds, ordered by publication date, descendant. Posts should be stored inside the directory `content/post/`. Sub-directories are allowed.

Tags

Tags are identifiers that are used to classify posts by topic. Each tag generates a HTML page and an atom feed with related posts. Tags aren't allowed in pages, only posts.

1.1.3 Main features

These are some of the cool features of blohg:

- Support for static pages and posts.
- Support for tags for posts.
- Support for aliases, making it easy to migrate from some other blogging engine.
- Support for building a static version of the blog, to host it in restricted environments.
- Post/page metadata grabbed from the Mercurial repository.
- Plenty of reStructuredText directives available, to make the blogging experience as smooth as possible.
- Easily customizable by Jinja2 templates.
- Can be used as a full-featured CMS.
- Support for pagination for posts.
- Atom feeds for posts and tags.

1.2 Installing blohg

This section will guide you through the alternatives for setting up blohg in your operating system. blohg is currently tested on Linux and Windows, but should works in any operating system where Flask and Mercurial (and/or Git) run properly.

blohg works on Python 2.7.

blohg is available at the *Python Package Index* (PyPI):

http://pypi.python.org/pypi/blohg

Warning: Before installing blohg manually or using `pip`, make sure that you have a C compiler and the usual build tools (e.g the `build-essential` package for Debian/Ubuntu) installed. You can work around these dependencies if you need (e.g when running Windows), installing Mercurial with `--pure` argument, running the following command inside of a directory with the Mercurial sources:

```
# python setup.py --pure install
```

There's no way to install Git bindings without a compiler, unfortunately.

Warning: libgit2 is experimental and breaks the ABI every minor release. You may have some trouble when trying to get pygit2 working.

1.2.1 Manually

Download the latest tarball from PyPI, extract it and run:

```
# python setup.py install
```

1.2.2 Using `pip`

To install blohg using `pip`, type:

```
# pip install blohg
```

If you want to use Git repositories, install a recent version of libgit2 (yeah, setuptools/distutils don't know how to handle non-python dependencies. see the official documentation for instructions about how to install it on your operating system), and type:

```
# pip install blohg[git]
```

You should be careful about the version of libgit2 installed on your system. The major and minor versions of pygit2 should match the major and minor versions of libgit2, e.g. if the required version of pygit2 is 0.19.1, you need libgit2-0.19.*.

1.2.3 Gentoo Linux

There's a Gentoo ebuild available in the main tree. Install it using:

```
# emerge -av www-apps/blohg
```

1.3 Creating a new blog

This section will guide you through the steps required to get a blohg-based blog up and running.

Make sure that you read all the content available here in order to know how to use blohg properly.

1.3.1 Initializing the repository

blohg will install a script called `blohg` for you. This script is able to create a new Mercurial (or Git) repository, using the default template and/or run the development server. It will be your main tool to interact with blohg.

To create a new repository, type:

```
$ blohg initrepo --repo-path my_blohg
```

Where `my_blohg` is the directory where the new repository will be created.

Make sure that the directory doesn't exist, or is empty, before try to initialize the repository.

If you want to use Git instead of Mercurial, type:

```
$ blohg initrepo --repo-path my_blohg --git
```

When the repository is created, do the initial commit:

```
$ hg commit -Am 'initial commit'
```

Or for Git:

```
$ git add .
$ git commit -m 'initial commit'
```

Repository structure

The repository structure is pretty easy to understand:

```
my_blohg/
|-- config.yaml
|-- content
|   |-- about.rst
|   |-- attachments
|   |   `-- mercurial.png
|   `-- post
|       |-- example-post.rst
|       `-- lorem-ipsum.rst
|-- static
|   `-- screen.css
`-- templates
    |-- base.html
    |-- post_list.html
    `-- posts.html
```

Directory/File	Description
`config.yaml`	The main configuration file.
`content/`	The main content directory (for pages, posts and attachments).
`content/post/`	The posts directory. Any content stored here is handled as blog post, instead of page.
`content/attachments/`	The attachments directory. Any images or static files used in posts and pages should be here.
`static/`	The directory with static files used in the templates, like CSS files or images.
`templates/`	The directory with the Jinja2 templates.

1.3.2 Configuration options

You can heavily change the behavior of blohg by changing some configuration options.

These are the built-in configuration options for the `config.yaml` file:

Configuration option	Description	Default value
POSTS_PER_PAGE	Number of posts per page. Used by the posts pagination and the Atom feeds.	`10`
POSTS_PER_ATOM_FEED	Number of posts listed on the Atom feed	POSTS_PER_PAGE
AUTHOR	The name of the main author of the blog. Used by the Atom feeds.	`'Your Name Here'`
TAGLINE	A short tagline for the blog.	`'Your cool tagline'`
TITLE	The title of the blog, without HTML tags.	`'Your title'`
TITLE_HTML	The title of the blog, with HTML tags.	`'Your HTML title'`
CONTENT_DIR	The directory of the repository where the content is stored.	`content`
TEMPLATES_DIR	The directory of the repository where the templates are stored.	`templates`
STATIC_DIR	The directory of the repository where the static files are stored.	`static`
ATTACHMENT_DIR	The directory of the repository where the attachments are stored.	`content/attachments`
ROBOTS_TXT	Enable `robots.txt`, to prevent search engines from indexing source files, a.k.a. don't follow "View Source" hiperlinks.	`True`
SHOW_RST_SOURCE	Enable the view that shows the reStructured text source of your posts and pages.	`True`
POST_EXT	The extension of your post/page files.	`'.rst'`
OPENGRAPH	Enable the Open Graph meta tags block.	`True`
EXTENSIONS	List of enabled extensions.	`[]`
EXTENSIONS_DIR	The directory of the repository where the extensions are stored.	`ext`
RST_HEADER_LEVEL	reStructuredText header level	`3`

The default values are used if the given configuration key is ommited (or commented out) from the `config.yaml` file.

1.3.3 Customizing your templates

If you look at the `my_blohg` directory you'll see a `templates` directory. It stores some Jinja2 templates that are used by blohg.

Take a look at the Jinja2 documentation to learn how it works. The default templates provided by `blohg initrepo` are a good start point.

These are the blohg built-in variables globally available for your templates:

Variable	Description
`version`	A string with the current version.
`is_post`	A function with one argument, that returns True if the given argument is a the path of a post.
`current_path`	A string with the path of the current page/post.
`active_page`	A string with the first piece of the current path, useful to highlight the menu link for the current page.
`tags`	An iterable with all the available tags, ordered alphabetically.
`config`	A dictionary with all the configuration options.

Built-in templates

These are the built-in templates, that can be overriden from the repository:

404.html

Template for the 404 error page. You don't need to override it on your Mercurial repository if you don't want to customize something.

_posts.html

Template with some Jinja2 blocks that can be used by your custom templates. If you don't want to use the custom blocks just don't call them from the templates, and they will be ignored. You don't need to override this file in the repository.

These are the custom blocks available:

Type	Block name	Where to place
Disqus	`disqus_header`	inside the html header, in `base.html`.
Disqus	`disqus_post`	after the post contents, in `posts.html`.
Disqus	`disqus_footer`	at the end of `base.html`, before the `</body>` tag.
Pagination	`pagination`	at the end of `posts.html`, inside the main `div`. There's a CSS class, called `pagination`, to help you when changing the style.
Open Graph	`opengraph_header`	inside the html header, in `base.html`

Disqus support depends on the a `DISQUS` configuration variable, that should contain the value of the Disqus identifier of your blog. To get it, create an account at http://disqus.com/.

Open Graph support depends on a `OPENGRAPH` boolean configuration variable, that defaults to `True`.

base.html

The main template file, it's mandatory that this provided in the Mercurial repository. This template is inherited from by all others.

posts.html

Template used by the views that show partial/full content of pages and posts.

It inherites from `_posts.html` and can make use of its Jinja2 blocks.

Local variables available for this tempalte:

Variable	Description
`title`	A string with the page/post title.
`posts`	A list with all the posts (Metadata objects).
`full_content`	A boolean that enables display full content of `posts` and not just the abstracts.
`pagination`	A dictionary with 2 items (`num_pages`: number of pages, and `current` current page), used by the pagination block.
`tag`	A list of strings with tag identifiers, used by the view that list posts by tags.

post_list.html

Template for the page with the listing of blog posts, without content, just the name, the date and the link.

Local variables available for this template:

Variable	Description
`title`	A string with the page title (usually "Posts").
`posts`	A list with all the posts (Metadata objects).

1.3.4 Static files

The `static/` directory will store your static files, like CSS and images. You should avoid storing big files inside the Mercurial repository.

1.3.5 Dealing with search engines

blohg will disallow search engines from index your source files (`/source/` path), creating a `robots.txt` file in the root of your blohg instance. If you isn't running blohg from the root of your domain, you should make the requests pointing to `/robots.txt` redirect to `/path-to-your-blohg/robots.txt` in your webserver configuration.

If you don't want this `robots.txt` file, you can just add the following content to your `config.yaml` file:

```
ROBOTS_TXT: False
```

1.3.6 Hiding reStructuredText sources

blohg enables a `/source/` endpoint by default, that shows the reStructuredText source for any post/page of the blog. You can disable it by setting the `SHOW_RST_SOURCE` configuration parameter to `False`. It will raise a 404 error.

1.3.7 Using blohg as a CMS

You can use blohg to manage your "static" website, without the concept of blog posts. Actually the default setup of blohg is already pretty much like a CMS, but the initial page is a list of posts (or abstracts of posts), and you don't want it if you don't have blog posts at all.

You can use a static page as the initial page. You just need to save the text file as `content/index.rst` on your repository.

You can also use a static initial page for your blog, if you want, but you'll need to create a menu link pointing to the page with the list of posts. You can use the `views.posts` endpoint to build it:

```
<a href="{{ url_for('views.posts') }}">Posts</a>
```

1.3.8 Listing posts by tag

Each tag will have its own HTML page with all the posts:

- http://example.org/tag/foo/
- http://example.org/tag/bar/

It is also possible to combine multiple tags and get a HTML page:

- http://example.org/tag/foo/bar/

1.3.9 Atom feeds

blohg generates an Atom feed for all the posts and/or tags.

To include all the posts (actually just the `POSTS_PER_ATOM_FEED` last posts), use the following URL:

http://example.org/atom/

For each tag, use URLs of this form:

- http://example.org/atom/foo/
- http://example.org/atom/bar/

For multiple combined tags, use URLs of this form:

- http://example.org/atom/foo/bar/

1.3.10 Facebook/Google+ integration

We provide Facebook/Google+ integration using Open Graph HTML meta-tags.

There's a Jinja2 block available, that will add all the needed property tags for you. See *_posts.html*.

These are the property tags that will be created:

Property	Value
title	`TITLE` or the page/post title, if applicable.
description	`TAGLINE` or the page/post first paragraph, if applicable. Can be overrided by a `.. description:` reStructuredText comment.
image	Full URLs of all the images found in the page/post, if applicable. Each image will have its own `meta` tag.

If you don't want to use the default block, just remove the block call from your `base.html` template and write your own tags there. Use the default block, from `_posts.html`, as reference.

1.4 Writing blog pages/posts

blohg uses the standard reStructuredText syntax for pages and posts, with some additional directives.

1.4.1 Tagging posts

blohg implements the concept of tags for posts. Tags are defined as a comma-separated list inside a reStructuredText comment in the post source:

```
.. tags: my,cool,tags
```

Put this comment wherever you want inside the post.

1.4.2 Overriding the creation date

blohg retrieves the creation date of each page and post from the Mercurial repository, using the date of the first commit of the source file on it. If you want to override this date, just insire the UNIX timestamp of the desired date inside a reStructuredText comment:

```
.. date: 1304124215
```

A more readable timestamp format is also allowed (`YYYY-MM-DD HH:MM:SS`):

```
.. date: 2011-04-30 00:43:35
```

Timestamps should be in UTC.

This is useful if you want to migrate content from another blog.

1.4.3 Scheduling the post/page for a future date

If you want to have a post/page published in a future date automatically, you can add the same reStructuredText comment of the previous section, but with the timestamp of the future date. The page/post will not be listed until that date.

Warning: Make sure the date of your server is properly setup. Run NTP would be a good idea. :)

Warning: If your Mercurial repository is public (e.g. you have a hgweb instance running), people will be able to see the reStructuredText source before the publishing date.

1.4.4 Overriding the post/page author

blohg retrieves the author of each post/page from the Mercurial repository, as it does with the creation date. This data can be used in templates through the variables `post.author_name` and `post.author_email`. To override this data, add a reStructuredText comment like this:

```
.. author: John <john@example.com>
```

1.4.5 Post aliases

When migrating from another blogging system or URL structure, you can have blohg redirect your readers to the new URL's by providing your posts with URL aliases. If you need this, insert a reStructuredText comment with a comma separated list of the aliases for the post like this:

```
.. aliases: /my-old-post-location/,/another-old-location/
```

By default, blohg will issue a 302 (temporary) redirection. If you want, you can have blohg issue a 301 (permanent) redirection instead like this:

```
.. aliases: 301:/my-old-post-location/,/another-old-location/
```

The `301:` prefix is per URL and must be repeated for every URL you wish to 301 redirect.

1.4.6 Adding attachments

You may want to add some images and attach some files to your posts/pages. To atach a file, just put it in the directory `content/attachments` of your Mercurial repository and use one of the custom reStructuredText directives and roles below in your post/page.

Directive `attachment-image`

Identical to the `image` directive, but loads the image directly from your `content/attachments` directory.

Usage example:

```
.. attachment-image:: mercurial.png
```

Directive `attachment-figure`

Identical to the `figure` directive, but loads the image directly from your `content/attachments` directory.

Usage example:

```
.. attachment-figure:: mercurial.png
```

Interpreted Text Role `attachment`

Interpreted Text Role that generates a link to the attachment (`reference` node). You can add a custom label for link after '|'.

Usage example:

```
This is the attachment link: :attachment:`mercurial.png`
This is the attachment link: :attachment:`mercurial.png|link to file`
```

1.4.7 Additional reStructuredText directives/interpreted text roles

These are additional custom directives, that add some interesting functionality to the standard reStructuredText syntax.

Directive `youtube`

reStructuredText directive that creates an embed object to display a video from YouTube.

Usage example:

```
.. youtube:: erPnyi90cIc
   :align: center
   :height: 344
   :width: 425
```

Directive `vimeo`

reStructuredText directive that creates an embed object to display a video from Vimeo.

Usage example:

```
.. vimeo:: 2539741
   :align: center
   :height: 344
   :width: 425
```

Directive `code`

reStructuredText directive that creates a pre tag suitable for decoration with http://alexgorbatchev.com/SyntaxHighlighter/

Usage example:

```
.. code:: python

    print "Hello, World!"

.. raw:: html

    <script type="text/javascript" src="http://alexgorbatchev.com/pub/sh/current/scripts/shCore.js"><
    <script type="text/javascript" src="http://alexgorbatchev.com/pub/sh/current/scripts/shBrushPytho
    <link type="text/css" rel="stylesheet" href="http://alexgorbatchev.com/pub/sh/current/styles/shCo
    <script type="text/javascript">SyntaxHighlighter.defaults.toolbar=false; SyntaxHighlighter.all();
```

Directive `sourcecode`

reStructuredText directive that does syntax highlight using Pygments.

Usage example:

```
.. sourcecode:: python
   :linenos:

    print "Hello, World!"
```

The `linenos` option enables the line numbering.

To be able to use this directive you should generate a CSS file with the style definitions, using the `pygmentize` script, shipped with Pygments.

```
$ pygmentize -S friendly -f html > static/pygments.css
```

Where `friendly` will be your Pygments style of choice.

This file should be included in the main template, usually `base.html`:

```
<link type="text/css" media="screen" rel="stylesheet" href="{{
    url_for('static', filename='pygments.css') }}" />
```

This directive is based on `rst-directive.py`, created by the Pygments authors.

Directive `math`

reStructuredText directive that creates an image HTML object to display a LaTeX equation, using Google Chart API.

Usage example:

```
.. math::

    \\frac{x^2}{1+x}
```

Directive `include`

reStructuredText directive that reads a reStructuredText-formatted text file and parses it in the current document's context at the point of the directive. The directive argument is the path to the file to be included, relative to the repository root.

This directive replaces the `include` directive, provided by docutils, that can be harmful when running on shared environments.

Usage example:

```
.. include:: somefile.txt
```

More detailed documentation can be viewed in the Docutils' documentation.

This directive, unlike default implementation, will include files stored in the Mercurial repository.

The directive `include-hg` is an alias for this directive.

reStructuredText variables declared as comments in the included files are going to be ignored.

Directive `subpages`

reStructuredText directive that creates a bullet-list with the subpages of the current page, or of a given page.

Usage example:

```
.. subpages::
```

Or:

```
.. subpages:: projects
```

Supposing that you have a directory called `content/projects` and some reStructuredText files on it. Subdirectories are also allowed.

It is also possible to change the way the bullet-list is sorted, using the options `sort-by` and `sort-order`:

```
.. subpages::
   :sort-by: slug
   :sort-order: desc
```

Available options for `sort-by` are `slug` (default option), `title` and `date`, and for `sort-order` are `asc` (default option) and `desc`.

This directive will only show the files from the root of the directory. It's not recursive.

Interpreted Text Role `page`

Interpreted Text Role that generates a link to the given page. The text displayed is by default the title of the linked page. You can replace it with a custom title using this syntax: `:page:'Link title <linked-page>'`.

Usage example:

```
This is the :page:`posts/my-first-blog-post`
This is my :page:`Introduction Post <posts/my-first-blog-post>`
```

1.4.8 Previewing your post/page

After writing your post/page you will want to preview it in your browser. You should use the `blohg` script to run the development server:

```
$ blohg runserver --repo-path my_blohg
```

Supposing that your Mercurial repository is the `my_blohg` directory.

If the blohg script is running on the debug mode, which is the default, it will load all the uncommited content available on your local copy.

If you disable the debug mode (`--no-debug` option), it will only load the content that was already commited. This is the default behavior of the application when running on the production server.

For help with the script options, type:

```
$ blohg runserver -h
```

1.4.9 Commiting your post/page

After finishing your post and previewing it in your browser, commit your reStructuredText to the repo as usual.

1.5 Deploying your blog

Warning: This guide does not cover the deploy using a Git repository, but there are no big differences. Just use the equivalent `git` commands.

1.5.1 Using a WSGI app

At this point you should have a Mercurial repository with your blog ready to be deployed.

Copy it to your remote server as usual. e.g. using `ssh`:

```
$ hg clone my_blohg ssh://user@yourdomain.tld/path/to/my_blohg/
```

Supposing that your Mercurial repository is `my_blohg`.

Don't forget to add the remote path to your local `my_blohg/.hg/hgrc` [paths] section.

The blohg deployment process is similar to any other Flask-powered application. Take a look at the Flask deployment documentation:

http://flask.pocoo.org/docs/deploying/

To create your Flask `app` object, use the following code:

```
from blohg import create_app
application = create_app('/path/to/my_blohg')
```

There's a sample `blohg.wsgi` file (for Apache mod_wsgi) available here:

http://hg.rafaelmartins.eng.br/blohg/file/tip/share/blohg.wsgi

1.5.2 Using static pages

You can use the *freeze* command to generate a static version of your blog. This will create a `build` directory with the content of your blog as static pages. This way, you can put those pages (via **ftp**, **rsync**, **hg**, ...) on a static hosting provider.

-serve
: This option will serve your generated pages as a local web server. This can be used to check that all links works fine, or that all content has been generated.

-noindex
: This option will generate your post as html files rather than as directories containing a `index.html` file.

Note: This command uses Frozen-Flask as underlying generator. The configuration parameters from Frozen-Flask are also efective for this command, just put them inside blohg's *configuration file*. One worth mentionning is `FREEZER_BASE_URL`, as it indicates which base url to put in front of the external links, like is used for all the attachments.

1.6 Writing extensions

Warning: This feature is experimental and mostly not documented. Use it at your own risk. Things may change on the go. You should be ready to look at the source code if something breaks.

Warning: This is a last resource feature, that may open a security hole on shared environments, if not deployed correctly! Be careful!

Blohg extensions are usual python scripts or modules, that are imported by blohg itself, and that can change a big part of its behavior.

Extensions can be installed anywhere on your python path (safe for shared environments) or even inside the blog repository (unsafe for shared environments).

The script (or package directory) should be named following the convention:

```
blohg_$name
```

For example, the python script that implements the `foo` extension would be called `blohg_foo.py`.

Extensions can do almost anything, from adding a new view to adding a new reStructuredText directive to be used in the posts.

Flask, Werkzeug and docutils documentation should be quite useful here, please read them carefully.

1.6.1 Simple extension example

Save the code bellow as `blohg_hello.py`, somewhere in your python path:

```
# -*- coding: utf-8 -*-

from blohg.ext import BlohgBlueprint, BlohgExtension

ext = BlohgExtension(__name__)
hello = BlohgBlueprint('hello', __name__, url_prefix='/hello')

@hello.route('/')
def home():
    return 'Hello world!'


@ext.setup_extension
def setup_extension(app):
    app.register_blueprint(hello)
```

Add the following content to your `config.yaml` file:

```
EXTENSIONS:
    - hello
```

And run your development server:

```
$ blohg runserver
```

If everything is ok, you should see a "Hello World", if you point your browser at http://127.0.0.1:5000/hello/.

1.6.2 Running extensions from repository

If you want to run an extension from the repository, you should create an `ext` (configurable using the `EXTENSIONS_DIR` variable in your configuration file) directory at the root of the repository and place the extensions there. The extension setup is the same as explained above.

Extensions shipped inside the repository are called "embedded extensions", and you need to enable them explicitly in your WSGI script, to avoid security issues. Replace the `create_app` call with something like this:

```
application = create_app('/path/to/your/repository', embedded_extensions=True)
```

1.7 Upgrading blohg

1.7.1 From <=0.5.1

blohg 0.6 introduces support to Flask 0.7, that comes with some backwards incompatibilities.

You'll need to run the flask-07-upgrade.py script inside your blog repository to fix your templates, as described in the Flask documentation:

http://flask.pocoo.org/docs/upgrading/#version-0-7

1.7.2 From <=0.9.2

blohg 0.10 introduces Facebook/Google+ integration using the Open Graph protocol.

See *_posts.html*, or just add the following content to your `base.html` template, inside of the `<head>` and `</head>` tags:

```
<!-- begin opengraph header -->
{% block opengraph_header %}{% endblock %}
<!-- end opengraph header -->
```

blohg 0.10 uses `jinja2.Markup` to return HTML content from the models, deprecating the usage of the `safe` filter. You may want to fix your templates:

```
--- a/templates/posts.html
+++ b/templates/posts.html
@@ -21,9 +21,9 @@

 <!-- begin html parsed by docutils -->
 {% if full_content -%}
-    {{ post.full_html|safe }}
+    {{ post.full_html }}
 {% else -%}
-    {{ post.abstract_html|safe }}
+    {{ post.abstract_html }}
 {%- endif %}
 <!-- end html parsed by docutils -->
```

The directive `.. include::` was patched and will just have access to files from the Mercurial repository for now. This change improves the security, avoiding the access of files from the host filesystem, and makes it possible to include files inside the repository. Please remove any calls of this directive that were using files outside the repository.

CHAPTER 2

Additional Notes

2.1 License

blohg is released under the GPL-2 license. The full content of the license is available on the source tarball, or online: http://hg.rafaelmartins.eng.br/blohg/file/tip/LICENSE

2.1.1 Authors

blohg is written and maintained by Rafael G. Martins and various contributors:

Development Lead

- Rafael G. Martins <rafael@rafaelmartins.eng.br>

Patches and Suggestions

- Anton Novosyolov
- Benoit Allard
- Bruno Yporti
- Christian Joergensen
- Ry4an Brase