BugLink Documentation

Release 0.1

Benoit Allard

Sep 27, 2017

Contents

BugLink is a Mercurial extension that helps link changesets with Issue IDs.

The final goal is to be able to ask the list of issue being tackled between two changesets.

BugLink is composed of three components:

- client-side extension
- server-side extension
- web interface

The typical workflow is as follow:

1. A developper is working on an issue on his computer where the client-side extension is enabled, during develpement, he indicates the issue ID he is working on.
2. The developper reached a milestone and decides to push his progress to the server. During the push operation, the information about the issue(s) being worked-on is transfered to the server.
3. The server upon reception of the information updates the database with the changesets and their corresponding issue ID.
4. Management heard some progress have been made toward the next release, they point their web-browser to the local web shop, and smile as they see that three more issues have been worked at since the last time they looked at it.

Fig. 1: BugLink workflow
The developpers push to the server which updates the database, and management access the database through a web interface

CHAPTER 1

Overview

Installation

Client-side extension

This is the most common setup for this extension, being enabled on the client-side at the developer desk to allow him to indicate alongside with his changeset on which issue he is busy.

This is nothing fancier than enabling a general mercurial extension. No weird dependencies are needed, as the comunication with the database is done from the server-side.

After cloning the BugLink repository, the following lines have to be added to your user specific mercurial configuration file (`Mercurial.ini` on windows or `.hgrc`):

```
[extensions]
buglink = path/to/buglink/client/
```

Server-side extension

This one has to be enabled on each repository that may receive changesets from developpers. This extension will care about updating the database with the issue ID the developpers have been busy with. This can be a server-wide setting, a repository specific setting, or a setting reserved for the web instance of mercurial (`hgweb.conf` for instance).

This extension needs to talk to the database, and thus has dependency on SQLAlchemy.

As any other mercurial extensions, it is enabled through modification of the mercurial configuration file (`Mercurial.ini` on windows or `.hgrc`), where the following lines have to be added:

```
[extensions]
buglink_srv = path/to/buglink/server/
```

Web Interface

We are talking here about a WSGI application. there are plenty of guide on the internet on how to attach a WSGI application to your favorite web server. It looks like the last trendy one is to have a Gunicorn instance behind a nginx proxy. Apache and mod_wsgi also works fine.

The web interface has a bit more dependencies as the rest, as it needs to talk to the database, and deliver content for a web browser. The needed dependencies can be found in `srv-requirements.txt`. This file follows the pip requirements file format. So that within your virtualenv you just need to run:

```
pip -r srv-requirements.txt
```

The web app is located in the following file:

```
server/web.py
```

Usage

I will take for granted that you know how to handle a web browser, the server side-extension does not needs anything more than being enabled, we will also only discuss here the usage of the client-side extension.

This extension will record for each specified changeset an issue ID. Those are free-formed strings (without `\n`) locally stored within mercurial until they are pushed to the server and during the following pull operation the server aknowledge that they have been integrated to the database.

This information can be stored using different method:

During push

If you want to add an issue ID to all the pushed changeset together during your push operation, just use *`hg push --issueid`*, and the given issue ID will be linked to the pushed changesets and transmitted to the remote side.

During commit

You can also specify the issue that was worked on at commit-time, for this, you just have to use *`hg commit --issueid`* and the commited changeset will get a link with the specified issue ID.

At any other time

You can always use the command **`hg link`** or **`hg unlink`** to add or edit the issue IDs stored in the local cache.

CHAPTER 2

Mercurial Extension

Client side

Action upon push

All known links are also pushed to the remote repository. If the operation succeeded, the extension can assume its links made it to the database.

Added commands

Those commands acts on the local issue cache. This one will be transfered to the remote side upon push. the local cache can also be augmented from a remote side if two developpers having the client-side extension enabled push/pull to each other. In such a case, the issues referenced on one side will be duplicated on the other side. And the first one to push to a repository with the server-side extension will publish the full list to the database.

link

This command link a changeset with an issue ID.

`REVSET`
: This indicate the revisions on which the operation should be done.

`ISSUE_ID`
: This is the name of the issue to be associated.

`--remove`
: This tell Mercurial not to add a new correspondance, but to remove the one

---

Note: Only one of *`--remove`* or *`ISSUE_ID`* should be given at the same time. See **`hg unlink`** to remove a issue from the local cache.

---

unlink

This command, well, unlink a (or multiple) issue. The result will be that the unlinked issue will not be referenced any more in the local cache.

ISSUE_ID
This is the reference to the issue to be unlinked

links

This command takes two revisions as parameters (defaulting to `-1` and `tip`) and output the issues that have been worked on between them.

--revisions
This option output the revisions where the issues have been worked on together with the issue ID itself.

Todo

Add a `--graph` option

Aded options

hg commit

--issueid
To specify an issue ID for the to-be-commited changeset.

hg push

--issueid
To specify an issue ID for the to-be-pushed changesets.

Server side

Warning: Due to dependencies on (among others) sqlalchemy, this extension cannot works on Windows with a packaged version of Mercurial. Those one having only access to the pre-packaged dependencies.

The server-side extension will mostly reacts on push from a developer and update the database with the pushed information.

Action upon pushkey

Upon a push, initiated from the client-side, a set of changeset will be transmitted from the client side. A client also using the buglink extension will also try to push all its known issue IDs. This will be pushed using mercurial pushkey protocol.

The extension will be pushed a set of key from the client. Each key correspond to a changeset, and the corresponding value is a free string indicating the corresponding issue ID.

For each received key, the server will optionaly parse the free string, and update the database about the link between the changeset and the issueID.

For each pushed changeset, a new entry will be made in the database with the corresponding repository and the parent(s) revision(s).

Added commands

createdb

This will initialise an empty database to the last schema version.

updatedb

This will update an old database schema so that the updated server-side extension can continue to work on the same database. This operation is safe to run multiple time. If the database is up-to-date, this operation will simply do nothing.

It is also possible to downgrade the database to an previous schema version (For instance if stuck by a show-stopper bug in the last version). No command is provided for this operation, but the right mechanism is included as part of the sqlalchemy-migrate versionning.

Configuration

This extension can be configured through the mercurial config file.

The following configuration values have effects:

`buglink.db_url` (default to `sqlite:///dblink.db`) This is a SQLAlchemy database url to the database which should receive the issue links.

`buglink.strip` (also `notify.strip`, or `3`) This is the number of directory to be stripped to the base of the repository path for reference into the database.

CHAPTER 3

Web interface

GraphViz needs to be installed on the server and **`dot`** be in the `PATH`.

CHAPTER 4

Frequently Asked Questions

Why not using the commit message ?

Commit messages in mercurial are written once in a changeset and cannot be changed again without altering the whole history based their corresponding changeset.

Humans are not errors-prone, and it (sometimes) happen that someone forget to indicate which issue ID correspond to the changeset he just committed, or even enter a wrong ID. As those errors cannot be easily corrected, this information is not stored this way. This way, it is still possible to change the issue ID referenced by a changeset commited three years ago.

Do I need the server-side stuff ?

If you don't care about the web interface, you can only enable the extension on your computer and use the locally stored links between changeset and issue ID as a reference. the command **`hg links`** can be used for this purpose.

Why not storing the bug state ?

That's the job of the bug tracker, we're only here to make the missing link between him and changesets. Furthermore, developers (for who this tools is written) don't know if the bug has indeed been fixed until Q&A validates it.

How do I import all my issues as a batch ?

Say, like Mercurial, you used to store your issue number in your comit message. You can import them as follow:

```
for i in `hg log --template '{rev}\n'`; do ref=`hg log -r $i --template '{desc}\n' |␣
↪perl -ne 's/.*(issue\d+).*/$1/ && print'`; if ! [ -z "$ref" ]; then echo $i $ref >>␣
↪issuelist;fi; done;
```

And then import the list via the **hg debugimportlink** command:

```
hg debugimportlink issuelist
```

CHAPTER 5

For Developpers

Write tests

Client-side extension

The client-side extension uses Mercurial's own test framework to be tested. This means that mercurial has to be present as a source package for tests to be run.

The recommended setup is to clone mercurial repository next to the buglink one.

Then To run the tests, simply run:

```
$ make tests
```

within the buglink root directory. If you added some, and want to update the corresponding output, call the tests as follow:

```
$ make tests TESTFLAGS=-i
```

This way, you will be asked if the given output match your expectations, and if you want to include it in the test-case.

Server-side extension

Database migration

database migration, using sqlalchemy-migrate can be tested with their own tools, just run:

```
migrate test sqlite:////path/to/db .
```

in the `server/dbmigrate` directory. (Don't forget to run it only on test-databases).

Mercurial hook

Todo

Write tests

Web Server

Todo

Write flask tests

Extend the DB Schema

The raw program, although bringing a good stable base to manage issues might not be enough for your setup, it, for instance, does not tries to match different repositories together. You might want to tweak it to match your release procedure.

I do it by branching the repository, which means bug fixes have to be regulary merged into my private branch. Feel free to do the same.

CHAPTER 6

ToDo List

Note: This list is autogenerated from the doc itself.

Todo

Write tests

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/buglink/checkouts/stable/docs/dev.rst, line 46.)

Todo

Write flask tests

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/buglink/checkouts/stable/docs/dev.rst, line 51.)

Todo

Add a `--graph` option

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/buglink/checkouts/stable/docs/mercurial.rst, line 77.)

Index

Symbols

E

H

I

P

R