BugLink Documentation

Release 0.9

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

QuickStart

Setup

Alice is the developer of the company.

Bob cares of the support calls.

Carol watch over the quality of the product.

Dave is the manager.

Support call arrives

Bob, receive a phone call from an polite customer not happy about the greenish colour of the main button, and asks if it could be red.

Bob tries to reproduce the issue, realizes that the red button would better fit in the design of the product, and opens *issue256* in the company's ticketing system, which cares of triggering *Alice*.

Issue get worked at

Alice receive a mail notifying her about *issue256*, reads it carrefully, and begins coding.

Once done, *Alice* indicates to Mercurial that the commit just done belongs to *issue256*:

```
$ hg commit -m "Make the button red"
$ hg link tip issue256
Associating 42685d8fcc0f with issue256
$
```

Alice checks the local associations out of curiosity:

```
$ hg links
issue256
$
```

That done, *Alice* push her modification to the server:

```
$ hg push https://hg.company.com/
pushing to https://hg.company.com/
searching for changes
remote: adding changesets
remote: adding manifests
remote: adding file changes
remote: added 1 changesets with 1 changes to 1 files
Remote accepted 1 links.
Those will be removed from the local storage
$
```

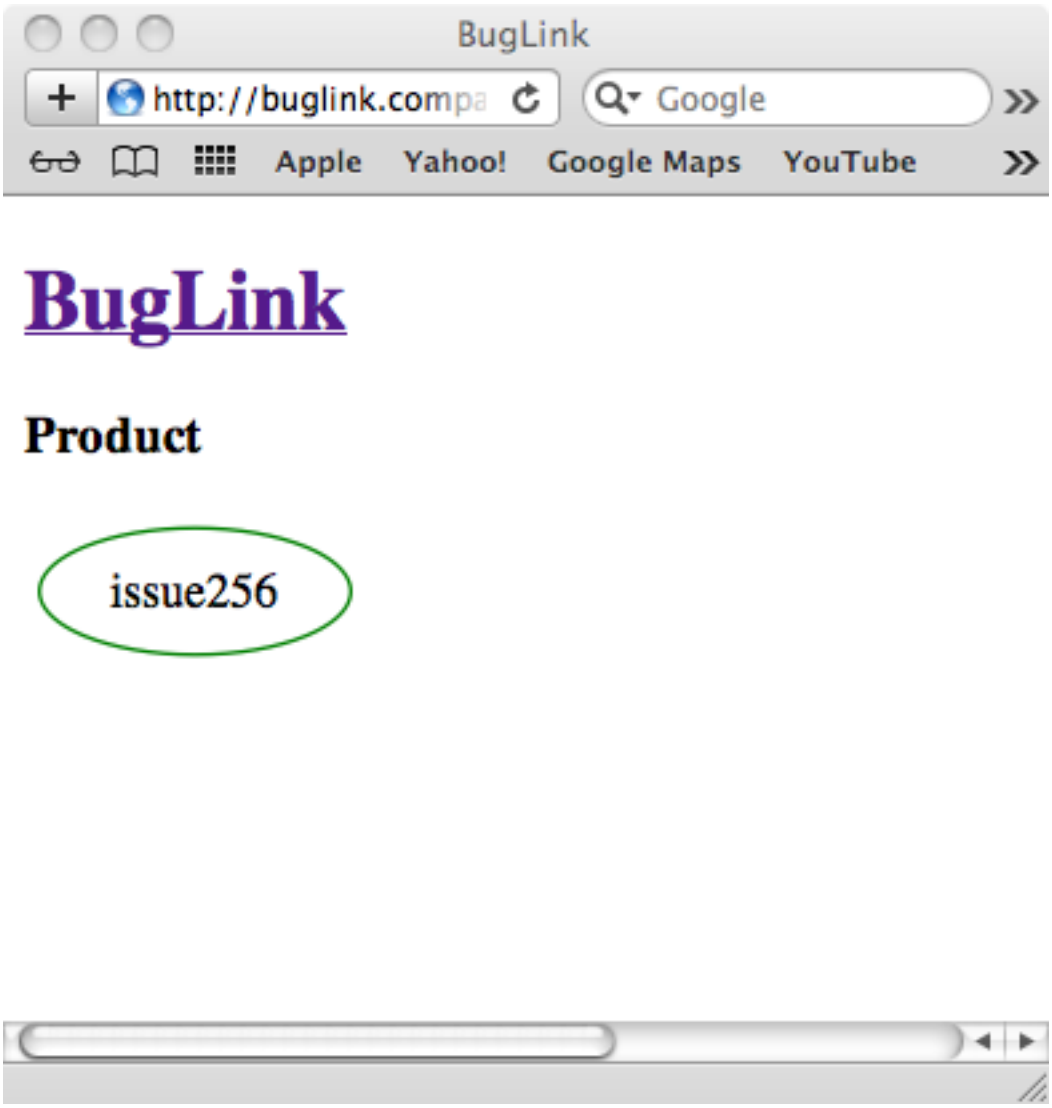
Issue get tested

Alice then release a test build for *Carol*, and set the status of *issue256* to *test*.

Carol get notified from the ticketing system that *issue256* waits for her approval and test it, satisfied from the red button, she set the issue status to *resolved*, and notifies *Dave* that a new issue has been solved, and that it might be time to cut a new release.

New release is made

Carol and *Dave* together anayse the list of solved issue since the last release by requesting it to the local web server:



In this case, *Carol* knows well this issue has been solved, but she can check it again with the ticketing system, and then *Dave* can make the decision to release a new version.

CHAPTER 2

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/
```

The location of the database has to be configured in the same configuration file. This has to be a sqlalchemy database url. For instance:

```
[buglink]
db_uri = sqlite:////tmp/dblink.db
```

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
```

Flask having its own integrated web server, it can be tested as follow (the optional parameter is the name of the configuration file):

```
python server/web.py [config.yaml]
```

A sample WSGI script is given in `server/buglink.wsgi`. This file is reproduced here:

Configuration

The web server is configured through the usage of a configuration dictionary. This one can be given in the WSGI script or through a config file in the YAML format.

the following configuration values are understood:

`DB_URL` This is the url to the database in the sqlalchemy format. For instance:

```
sqlite:////var/buglink/buglink.db
```

`SECRET_KEY` This is a random string used to encrypt the cookies used by the web application.

`links` This configuration key is used to configure cross links between buglink and other web instances (for instance the mercurial server, or the bug tracker.)

The `links` key support the following sub-keys:

`repository` This is used to link a repository name to a web page, usually, the mercurial server.

`changeset` This configuration key is used to link a changeset to a web-page usually on the mercurial server.

`issue` This is used to link an issue to a web page (the bug tracker for instance).

All the sub-keys follow the same format. They are a list of settings, each setting can have three sub-keys: `re`, `default` and `url`. At least the `url` sub-key should be filled to generate a link.

The following mechanism is used for each setting of each sub-keys in the order they are defined, later values being only used if the previous one did not matched:

1. The value (*repository path*, *changeset hash*, or *issue reference*) is matched against the content of the `re` value.
2. If the regular expression matches, we go on to the next step, else the next setting is tried.
3. A dictionary is created with each matched variable of the regular expression (in the form `(?P<id>...)`). Optional variables get their values from the `default` setting. This setting is a dictionary associating variable name to default-values.
4. The url is generated from the content of the `url` setting, by replacing each format string (in the form `%(name)s`) with its matched value during the regular expression matching.

As an example, *tortoisehg*, a project stored on Bitbucket.org should use the following setting:

```
links:
  repository:
    -
      url: https://bitbucket.org/tortoisehg/%(path)s/overview
  changeset:
    -
      url: https://bitbucket.org/tortoisehg/%(path)s/changeset/%(hash)s
  issue:
    -
      re: #(?P<id>\d+)
      url: https://bitbucket.org/tortoisehg/%(path)s/issue/%(id)s
```

CHAPTER 3

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 4

Web interface

Introduction

While the mercurial interface provide an append only interface to the database, the web one provides a read and modify interface to it.

Dependencies

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

CHAPTER 5

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, (you know, just interverting two numbers : `2658` and `2568` ...). As those errors cannot be easily corrected if stored in the commit message, we prefer to store this information at another place. 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.

Todo

Add a graph view on the command line

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 commit message. You can import them as follow (Or if you don't understand it, give it to your system administrator):

```
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 generated list via the **`hg debugimportlink`** command:

```
hg debugimportlink issuelist
```

CHAPTER 6

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 (probably `buglink`). 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

Unit-tests

Some parts of the web-server can be independently tested. In those cases, write teste into the `tests/server` directory.

To run those tests use the following command line in the `buglink` directory:

```
$ PYTHONPATH=. python tests/server/runtests.py
```

Flask-tests

Todo

Write flask tests as described under http://flask.pocoo.org/docs/testing/.

Upgrade the DB Schema

The DB is versionned, which means that each upgrade should follow a specific procedure. This one is described there.

1. Write a new script under `server/dbmigrate/versions`
2. Test the upgrade procedure with sqlalchemy-migrate
3. Upgrade the DB model version and (maybe) dscription in `server/models.py`.
4. Optionnaly integrate your change in the view.

A tentative is made to keep the schema decribed in the following figure:

Fig. 6.1: Database schema version 5

CHAPTER 7

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/latest/docs/dev.rst, line 47.)

Todo

Write flask tests as described under http://flask.pocoo.org/docs/testing/.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/buglink/checkouts/latest/docs/dev.rst, line 66.)

Todo

Add a graph view on the command line

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/buglink/checkouts/latest/docs/faq.rst, line 30.)

Todo

Add a `--graph` option

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

Index

Symbols

E

H

I

P

R