Repoze Documentation

Release 1.0

Agendaless Consulting, Inc. and Contributors

December 12, 2014

Contents

The Repoze project is a collection of technologies which bridges the WSGI and Zope worlds. The project's goals:

- Make it possible for non-Zope Python developers to selectively use Zope features in a WSGI environment.
- Help Zope developers integrate their applications into a WSGI environment.

Contents:

CHAPTER 1

Overview of the Repoze Project

1.1 Problems Addressed

Some of the things that Zope does are useful enough to be applicable to non-Zope-based WSGI applications: transaction management, virtual hosting, form marshalling, declarative access control, etc. But currently it's not easy for Python web developers who aren't fluent in Zope to make use of these features, because they aren't always decomposed into individually reusable pieces.

At the same time, WSGI deployment of Python applications has become a defacto standard in the wider Python world. At the time of the project's inception, it was reasonably difficult to serve up Zope (particularly Zope 2) via a WSGI server.

Note: The current release line (2.13.x) of Zope2 has landed changes originally made here to support WSGI natively.

1.2 Solutions Provided

The Repoze project provides components which reimplement core Zope features as WSGI middleware (repoze.vhm, repoze.retry, repoze.tm2), and WSGI applications (repoze.zope2, repoze.grok). The Repoze project also reuses existing WSGI middleware (Paste) and servers where possible.

The bits of Repoze that reimplement core Zope features can be ignored or used as necessary in non-Zope contexts.

1.3 Software Requirements and Limitations

- The packages in the `repoze.` namespace require setuptools for installation.
- None of the `repoze.*` software has been tested under any version of Windows. It has only been tested under UNIX variants (Linux and Mac OS X at the time of this writing).

1.4 Technology Dependencies

Many Repoze components depend on Paste, as well as setuptools and the WSGI specification. Repoze reimplements and reuses some technologies originated within Zope.

1.5 Licensing

The original bits that make up Repoze are released under a BSD-style license.

Some non-original parts of Repoze are licensed under the ZPL (another BSD-style license).

1.6 Resources

- Github organization: https://github.com/repoze/
- Mailing lists: http://lists.repoze.org/
- Blog: http://blog.repoze.org/
- IRC channel: irc://irc.freenode.net/#repoze

1.7 Legacy Resources

- Subversion repository (via ViewCVS): http://www.repoze.org/viewcvs/
- Subversion repository (via http): http://svn.repoze.org/

Note: Currently-maintained packages have their repositories on Github under the repoze organiztion.

- (Old) Repoze bug tracker: http://bugs.repoze.org/.

Note: Currently-maintained packages have their issue trackers located with their repositories on Github.

- Repoze Python package repositories: http://dist.repoze.org/

Note: The packages on `dist.repoze.org` were added to deal with some no-longer-relevant issues on the PyPI index. They are no longer being updated.

1.8 Contributing

The preferred mechanism for contributons is via pull requests on the project's Github repository.

To obtain write access to any oft, you will be required to sign a contributor's agreement.

CHAPTER 2

Current Repoze Components

The Repoze project consists of of the following software components under active maintenance.

2.1 WSGI Middleware

- `repoze.who`

  Identification and authentication framework for arbitrary WSGI applications. Can be used as WSGI middleware, or as a library within application code. It is inspired by Zope 2's Pluggable Authentication Service (PAS) but it is not dependent on Zope in any way.

  Docs: http://repozewho.rtfd.org/

  Github: https://github.com/repoze/repoze.who

- `repoze.tm2`

  WSGI middleware that implements a transaction policy. It uses the transaction module to commit or abort (if there is an exception) a transaction after each WSGI request.

  Applications may register callbacks with `repoze.tm2` (e.g., to close a connection, mainly). This behavior used to be hardcoded in Zope's publisher. `repoze.tm2` is useful in any web application that requires transactions.

  Docs: http://repozetm2.rtfd.org/

  Github: https://github.com/repoze/repoze.tm2

- `repoze.retry`

  WSGI middleware that retries a WSGI request some configurable number of times if any child application raises a "retriable" exceptin (e.g., ZODB's `ConflictError`, or the equivalent PostgreSql "repeatable read" error). This behavior used to be hardcoded in Zope's publisher.

  Docs: http://repozeretry.rtfd.org/

  Github: https://github.com/repoze/repoze.retry

- `repoze.profile`

  WSGI middleware component which aggregates Python profiling data across all requests to a WSGI application. It provides an optional HTML UI for viewing profiling data.

  Github: https://github.com/repoze/repoze.profile

- `repoze.debug`

  Middleware which can help with in-production forensic debugging.

  Github: https://github.com/repoze/repoze.debug

- `repoze.zodbconn`

  Library which manages ZODB databases and WSGI middleware which makes a ZODB connection available to downstream applications.

  Github: https://github.com/repoze/repoze.zodbconn

- `repoze.vhm`

  WSGI middleware which normalizes specially-mangled URLs (or individual header values) into ordinary paths, with the additional information stored in the environment for use in generating dynamic URLs. It is an analogue / replacement for the Zope2 Virtual Host Monster.

  Github: https://github.com/repoze/repoze.vhm

- `repoze.errorlog`

  WSGI middleware filter which intercepts exceptions and writes them to a Python `logging` module channel (or the `wsgi.errors` filehandle, if no channel is configured). It also provides an optional HTML UI for browsing limited exception history.

  Github: https://github.com/repoze/repoze.errorlog

2.2 Libraries

- `repoze.evolution`

  Allows a developer to keep persistent data structures (data in a relational database, on the filesystem, in a persistent object store, etc.) in sync with changes made to software. It provides a framework for developers to create and use a package full of monotonically named "evolve" scripts which modify the data; each script brings the data up to some standard of a software version.

  Github: https://github.com/repoze/repoze.evolution

- `repoze.catalog`

  An indexing and searching system based on `zope.index`.

  Github: https://github.com/repoze/repoze.catalog

- `repoze.session`

  A sessioning system (server side state for web applications) based on ZODB.

  Github: https://github.com/repoze/repoze.session

- `repoze.formapi`

  A form library which integrates with HTML forms instead of abstracting them away. It provides a small framework to take you through the entire process of rendering a form, provide default values, validate and execute form actions.

  Github: https://github.com/repoze/repoze.formapi

- `repoze.bitblt`

  WSGI middleware component which automatically scales images according to the `width` and `height` property in an HTML img tag.

Github: https://github.com/repoze/repoze.bitblt

- `repoze.squeeze`

  This package provides a WSGI middleware component which "squeezes" HTML documents by merging browser resources (javascript and stylesheets).

  Github: https://github.com/repoze/repoze.squeeze

CHAPTER 3

Obsolete Repoze Components

These components are no longer actively maintained: in most cases, they have been obsoleted by newer software.

3.1 WSGI Applications

- `repoze.bfg`

  `repoze.bfg` is a WSGI web framework inspired by Zope, Pylons, and Django. It uses Zope libraries to do much of its work.

  Obsoleted by Pyramid (see http://trypyramid.com/).

  Website: http://bfg.repoze.org/

- `repoze.obob`

  A stripped-down object publisher that acts as a WSGI application. It is responsible for:

  o selecting the "root" object of the graph for a given request / URL;

  o traversing from that root object along the "edges" defined by the URL path elements to find the "published object";

  o invoking the published object to obtain the body;

  o mapping response headers and body, along with the result from calling the published object, into appropriate WSGI output;

  o serializing / encoding the response based on the type of the request (e.g., JSON / XML-RPC).

  `repoze.obob` is currently more of a "publisher driver" than a publisher. Most of the actual work is done by "bob" modules which obob drives (repoze.zope2 is a "bob").

  `repoze.obob` is used by `repoze.zope2`, but is otherwise not being actively developed at this point.

  Obsoleted by Pyramid (see http://trypyramid.com/).

  SVN: http://svn.repoze.org/repoze.obob/trunk/

- `repoze.zope2`

  A "bob" helper module that implements an analogue of the Zope 2 ZPublisher, with some major simplifications and cleanups. Its core mission is to allow publishing existing Zope2 applications in a WSGI environment that externalizes some of the features of "classic" Zope2 into middleware.

  `repoze.zope2` is capable of publishing all known Zope applications, including applications which rely on WebDAV and XML-RPC, as well as all known Plone applications.

Obsoleted by Zope 2.13.x release

SVN: http://svn.repoze.org/repoze.zope2/trunk/

- `repoze.plone`

  A meta-egg which depends on all Plone component eggs as well as `repoze.zope2`.

  Obsoleted by Plone4 release

  SVN: http://svn.repoze.org/repoze.plone/trunk

- `repoze.grok`

  A "bob" helper module that implements an analogue of the Zope 3 publication machinery in order to serve up Grok applications.

  Abandoned.

  SVN: http://svn.repoze.org/repoze.grok/trunk/

- `repoze.mmwsgi`

  WSGI wrapper that allows Mailman to be run simply under a WSGI server.

  Abandoned.

  SVN: http://svn.repoze.org/repoze.mmwsgi/trunk/

- `repoze.kiss`

  A "bob" module which publishes content (files, images, templates) from the filesystem, using the `repoze.zope2` helper.

  Runs the http://www.repoze.org/ website.

  SVN: http://svn.repoze.org/repoze.kiss/trunk/

3.2 WSGI Middleware

- `repoze.what`

  An authorization framework for WSGI applications, based on `repoze.who`.

  Abandoned (after moving to Github).

  Github: https://github.com/repoze/repoze.what

- `repoze.browserid`

  `repoze.browserid` is WSGI middleware loosely based on the Zope 2 concept of "browser ids", which are cookies which represent a browser, for use by sessioning libraries.

  Abandoned (after moving to Github).

  Github: https://github.com/repoze/repoze.browserid

- `repoze.tempita`

  `repoze.tempita` is WSGI middleware egress filter which conditionally causes the body returned by the application to be run through the Tempita templating engine, using replacement values defined within the `repoze.tempita` Paste middleware configuration. Abandoned.

  SVN: http://svn.repoze.org/repoze.tempita/trunk/

- `repoze.decsec`

  Declarative ACL-based security via middleware for WSGI applications.

  Not widely used.

  SVN: http://svn.repoze.org/repoze.decsec/trunk/

3.3 Libraries

- `repoze.monty`

  A library that, given a WSGI environment dictionary (and a `wsgi.input` file pointer if the request is a POST request), will return a dictionary containing "converted" form/query string elements. The form and query string elements contained in the request are converted into simple Python types when the form element names are decorated with special suffixes.

  SVN: http://svn.repoze.org/repoze.monty/trunk/

- `repoze.urispace`

  A library implementig the URISpace 1.0 spec, as proposed to the W3C by Akamai. Its aim is to provide an implementation of that language as a vehicle for asserting declarative metadata about a resource based on pattern matching against its URI.

  SVN: http://svn.repoze.org/repoze.urispace/trunk/

  Docs: http://docs.repoze.org/urispace/

3.4 Buildout-related

The following are `zc.buildout` (see http://www.buildout.org) recipes and configuration files:

- Buildouts for `repoze.bfg`: http://svn.repoze.org/buildouts/repoze.bfg/
- Buildouts for `repoze.zope2`: http://svn.repoze.org/buildouts/repoze.zope2/
- Buildouts for `repoze.plone` http://svn.repoze.org/buildouts/repoze.plone/
- `repoze.recipe.egg`

  A fork of the `zc.recipe.egg` *zc.builout'* recipe. (see http://pypi.python.org/pypi/zc.recipe.egg). It does exactly what `zc.recipe.egg` does, except it also automatically installs scripts from dependent eggs. This software is deprecated.

  SVN: http://svn.repoze.org/repoze.recipe.egg/trunk

3.5 Miscellany

These components are (mostly) unsupported "convenience" things:

- `repoze.django`

  A mechanism to run Django under a Paste server.

  SVN: http://svn.repoze.org/repoze.django/trunk/

- `repoze.trac`

  A mechanism to run Trac under a Paste server.

  SVN: http://svn.repoze.org/repoze.trac/trunk/

- `whoplugins`

  Contributed `repoze.who` plugins.

  SVN: http://svn.repoze.org/whoplugins/

3.6 Re-packaged Software

- `zopelib` (no SVN)

  `zopelib` is the entire set of Zope "software home" `Products`-namespace packages packaged as a setuptools-compatible package. The script that allows for this is checked into Repoze's CVS repository That script is meant to be dropped into a checkout of a Zope "software home" and run from there to repeatably package Zope 2 as an sdist or bdist.

- `cmflib` (no SVN)

  `cmflib` is the Zope CMF packaged as a setuptools-compatible package. It includes all the Zope `Products`-namespace packages that are present in the classic CMF distribution (Products.CMFActionIcons, Products.CMFCalendar, Products.CMFCore, Products.CMFDefault, Products.CMFTopic, Products.CMFUid, Products.DCWorkflow) save for one: it has a dependency on an independently release-managed distribution of Products.GenericSetup. It was generated by using a "setup.py", http://svn.zope.org/Sandbox/chrism/eggcmf/2.1.0/setup.py?view=markup checked into a location which depends on externals in Zope Corporation's SVN repository

- `plonelibs` (no SVN)

  `plonelibs` is a setuptools-compatible repackaging of the packages that ship in Plone 3's "lib/python" directory.

- `ploneproducts` (no SVN)

  `ploneproducts` is a setuptools-compatible repackaging of the `Products`-namespace packages that ship in Plone 3.

  It depends on separately released-managed distributions of Products.PluggableAuthService and Products.PluginRegistry.

- `PIL` (no SVN)

  `PIL` is a repackaging of the Python Imaging Library (see http://www.pythonware.com/products/pil/) as a setuptools-compatible package.

  Obsoleted by Pillow (https://pypi.python.org/pypi/Pillow).

CHAPTER 4

History of the Repoze Project

4.1 Early Developments

The Repoze project began in September, 2007, as a response to the perception that the Zope community was stuck in a "ghetto", isolated from the rest of the Python web development world. The project had two main goals:

- Help make interesting features of the Zope development ecosystem available and interesting to the wider Python web developer community.
- Make the cool new features offered by that community accessible to Zope developers.

One primary focus for this early work was making the core Zope2 application server usable inside the new de facto standards of eggs (for distributions of Python projects) and WSGI (for connecting Python web servers interoperably with Python web applications).

Part of that task involved mapping the features of Zope's "publisher" (the framework mapping URLs and request variables onto Python code) onto the WSGI world. As a result of that work, important features of the publisher machinery became usable outside of Zope as WSGI middleware:

- `repoze.tm` and `repoze.tm2` made it possible to use Zope's model of transaction-per-request cleanly onto any web application which uses the `transaction` package originally developed as part of `ZODB`.
- `repoze.retry` made it possible to map the feature of the publisher which retries requests which would otherwise fail due to conflicts in an optimistic concurrency model, e.g. as raised by ZODB or Postgres.

While disentangling these features of the Zope publisher into components which could be re-used outside of Zope, it became clear that there was a smaller, more cohesive model for doing the real business of the publisher. The initial implementation of such a generic package (see `repoze.obob` in the repository) formed the basis for `repoze.zope2`, an almost- compabible implementation of the Zope2 publisher as a well-behaved WSGI application. This package delegates a number of the hard-wired features of the Zope2 publisher (transaction and retry handling, virtual hosting, etc.) to the equivalent Repoze middleware components.

4.2 Later Developments

Since its inception, the Repoze project has overstepped the bounds of its original goals. Instead of acting strictly as a clearinghouse which makes Zope technologies available to a larger Python community, the Repoze community is now producing its own components. These components may only be inspired by Zope technology, or may have no analogue in the Zope world.

For example:

- Chris McDonough set out to re-implement the Zope publisher model as a leaner, lighter weight web framework, now called BFG.
- Tres Seaver created Compoze which provides various tools for working with Python "egg" distributions and package indexes.

And so on.

See docs.repoze.org for documentation for all the components produced so far by the Repoze project.

See svn.repoze.org for all the source code produced by the Repoze project.

CHAPTER 5

Hacking on Repoze Components

5.1 Coding Standards

As a general rule, projects in the `repoze` repositories abide by the following standards:

- PEP 8 coding style. In particular, *Python code should never exceed 80 columns*.
- Project trunks should be kept in "ready-to-release" state: all unit tests pass, changelogs are kept update, etc.
- 100% test coverage before release, as reported by the `nose` test runner with the `coverage` add-on:
- Tests should be runnable using the default `setuptools` test runner:
- Full documentation of features and APIs.
- Solid release management, including releases to PyPI corresponding to "pristine" tags, detailed change logs, etc.

While some older projects may not be completely in line with this culture, we are committed to moving them all closer with any change. As a corollary: if you are suubmitting a patch to a project in this repository, and you want expedite its acceptance, ensure that your patch maintains or improves the target project's conformance to these goals.

5.2 Layout and Conventions

Each project should consist of a single, top-level project folder in Subverion, containing three conventional folders: `trunk`, where the majority of development work occurs, `tags`, containing the "pristine" tags made when releasing the project, and `branches`, containing both "maintenance" branches where bug fixes to a released version might be made, and "development" branches, for work which would otherwise de- stabilize the trunk.

Because we are mostly working on Python code here, the trunk and folders under the `tags` or `branches` folders are normally arranged as a `distutils` project, e.g.:

```
<directory>
- setup.py
- README.txt
- CHANGES.txt
+ docs/
  - Makefile
  - conf.py
  - index.rst
  - api.rst
  + .static/
  + .build/
```

```
  + .templates/
+ package/
  - __init__.py
  + subpacakge/
    - __init__.py
    - module.py
    + tests/
      - __init__.py
      - test_module.py
    + templates/
      - template.pt
```

For an example of this layout, see http://svn.repoze.org/template/trunk/ .

5.3 Distributed Version Control Systems

Under Subversion, the version repository is kept on a central server: each developer has a working subset checked out from that server onto her own machine.

Using a distributed version control systems (DVCS), each developer clones the entire repository onto her machine. Although it may take more space or bandwidth, having this clone allows the developer a lot of flexibility and freedom: she can hack, make commits, etc., to her local clone without needing network access, or even permission to write back to the source server!

Note: Actively-supported Repoze projects are now hosted on Github under the Repoze organization

5.3.1 Using Specific VCS Tools with Repoze Projects

For the actively maintained components maintained on Github, use the "normal" git / Github workflow, i.e.:

- Fork the repository.
- Hack on a clone of your fork in a branch.
- When ready, push your branch and submit a pull request.

We advise that you *not* hack on the `master` branch of your fork, so that you can sync more easily as changes are pushed to the upstream Repoze repository.

Working with Subversion

Warning: These instructions are only valid for Repoze components which have not been migrated to Github.

How-to: Get a read-only Subversion checkout

Anonymous, read-only Subversion checkouts can be made over HTTP:

```
$ svn co http://svn.repoze.org/repoze.who/trunk who-trunk
```

You should then be able to work inside `who-trunk`, fixing a bug or adding a feature. You can use Subversion commands as normal, e.g.:

```
$ cd who-trunk/
$ svn info
Path: .
URL: http://svn.repoze.org/repoze.who/trunk
Repository Root: http://svn.repoze.org
Repository UUID: 8f1d8bf8-68d2-4fbe-a113-2afb08c80ed9
Revision: 8672
Node Kind: directory
Schedule: normal
Last Changed Author: Tres Seaver <tseaver@palladion.com>
Last Changed Rev: 8673
Last Changed Date: 2010-03-26 16:21:39 -0400 (Fri, 26 Mar 2010)
```

Let's say you wanted ot add a bit of explanation to the `README.txt` file:

```
$ vi README.txt
...
```

Subversion knows about the changes you made:

```
$ svn stat
M      README.txt
$ svn diff
Index: README.txt
===================================================================
--- README.txt       (revision 8276)
+++ README.txt       (working copy)
@@ -8,6 +8,8 @@
 for arbitrary WSGI applications.  ``repoze.who`` can be configured
 either as WSGI middleware or as an API for use by an application.

+Blah, blah
+
 ``repoze.who`` is inspired by Zope 2's Pluggable Authentication
 Service (PAS) (but ``repoze.who`` is not dependent on Zope in any
 way; it is useful for any WSGI application).  It provides no facility
```

You can keep your checkout updated with ongoing changes, too:

```
$ svn up
U    docs/api.rst
U    docs/conf.py
Updated to revision 8673.
```

and you may have to deal with changes which conflict with those you have made.

However, because you are working in an anonymous, read-only checkout, you cannot commit your changes back to the repository.

```
$ svn commit -m "R00l da world."
svn: Commit failed (details follow):
svn: Can't create directory '/home/repoze/svn/db/transactions/8675-1.txn': \
  Permission denied
```

Oops, is all your hard work in vain?

How-to: Submit a patch from your Subversion checkout

Once you have fixed the bug or added the feature in your checkout, double- check that you have touched all the bases (see *Coding Standards* and *Layout and Conventions*). All is well, the tests pass, you added documentation for your cool new feature, so it is time to submit the patch.

First, **don't** try to cut and paste the output from `svn diff` into an e-mail message or a web-browser textarea: such operations usually end up mangling the line endings or other bits of the diff, and make it impossible to apply cleanly. The maintainer who has to do reconstructive surgery on such a victim may just give up and ignore the patch.

Avoiding the cut-and-paste train wreck is straightforward: just create the patch as a file:

```
$ svn diff > /tmp/repoze.who-my_cool_feature.patch
```

And then send or upload that file as an attachment: mailers and web-browsers are nearly as good at leaving attachments alone as they are at destroying sensitive inline text!

For `repoze` projects, the default place to submit patches is to the repoze tracker. You will need to register for an account, but you should then be able to create a new issue and upload your patch file to it. Good titles, descriptions, and tags on the issue should help it get the attention of the right maintainer for the project: if you don't hear back fairly quickly, try asking on the repoze IRC channel, or follow up to the repoze-dev mailing list.

How-to: Get a writable Subversion checkout

The Repoze project grants write access to the Subversion repository to developers who are active with the project. To obtain write access to the Repoze subversion repository, you must sign a contributor's agreement. This agreement is available in two varieties:

- Form for electronic signature Instructions for signing and remitting are included in the agreement.
- Form for physical signature A physically signed agreement should be mailed to the address below or faxed to (United States) 540 479 1706

```
Agendaless Consulting, Inc
20 Pawnee Drive
Fredericksburg, VA 22401
U.S.A.
```

Once you have submitted the form, a core developer will respond in e-mail requesting your SSH public key. Once that key is uploaded, you can make a writable checkout from Subversion:

```
$ svn co svn+ssh://repoze@svn.repoze.org/svn/repoze.who/trunk who-trunk
```

and then commit your changes back directly:

```
$ svn commit -m "Add new feature."
```

Working with Git against the Subversion Repository

> **Warning:** These instructions are only valid for Repoze components which have not been migrated to Github.

How-to: Branch with Git directly from Subversion

Git ships with a `git svn` subcommand which allows a developer to interoperate with a project whose main repository is in Subversion. Using this plugin, you can clone the branch from its native HTTP URL:

```
$ cd ~/repoze
$ git svn --stdlayout clone svn+http://svn.repoze.org/repoze.who
```

You may check out any subdirectory of the Repoze SVN repository that has the typical SVN "branches", "tags" and "trunk" layout. The branches, tags, and trunk will be imported into the resulting local Git repository in a sensible manner.

Inside the checkout, you can commit as usual using `git`, but you won't be able to `git svn dcommit` the code back to Subversion unless you use an `svn+ssh` checkout URL (see *How-to: Get a writable Subversion checkout*).

How-to: Submit a patch from your Git branch

From your Git branch, you can use `git format-patch` to create a series of patch files, and then submit them via e-mail or the issue tracker, just as in *How-to: Submit a patch from your Subversion checkout*.

```
$ git format-patch origin -CM --subject="Cool feature" \
    --to=repoze-dev@lists.repoze.org --from=your.email@example.com
```

Git has another built-in plugin, `git send-email` , which you can use to automate submitting the patch files via e-mail:

```
$ git send-email origin -CM --subject="Cool feature" \
    --to=repoze-dev@lists.repoze.org --from=your.email@example.com \
    --smtp-server=locahost --smtp-server-port=25
```

Please see the git-send-email documentation for directions on how to configure your mail transport properly.

How-to: Push your Git branch to a public server

As an alternative to uploading a patch from your Git branch (or e-mailing it), you can also publish your branch to a server where it can be cloned over HTTP for others to use, as well as for review and merging by the package maintainer.

Let's ssume that you have been hacking on `repoze.who`, and want to publish your 'saml-2.0' feature branch in hopes of landing it in the next release. Let's also assume that you have an account on GitHub, and want to publish your branch there. First, create the new empty repository on Github's New Repository page. Give the repository the name `repoze.who-saml_2.0`, add a description, and hit submit.

Then, from your terminal, push your branch to the new repository:

```
$ git remote add github git@github.com:<userid>/repoze.who-saml_2.0.git
$ git push github master
```

Replace `<userid>` with your Github account ID.

Pushing to other services According to Wikipedia's Git article, a number of other code-hosting services support Git branches. You should be able to publish your branch to any of them in a similar way.

Pushing to your own server You should be able to pubish your branch on any public webserver where you have space available, using the SSH protocol. E.g., assume that you have an account on `example.com`, where the contents of your home directory's `htdocs` directory are published under your userid:

```
$ git clone --bare /path/to/repoze.who-saml_2.0 repoze.who-saml_2.0.git
$ cd repoze.who-saml_2.0.git
$ touch git-daemon-export-ok
$ git --bare update-server-info
```

```
$ mv hooks/post-update.example hooks/post-update
$ cd ..
$ rsync -avz repoze.who-saml_2.0.git \
   example.com:/home/<youraccount>/htdocs/
```

You can then use http://example.com/~youraccount/repoze.who-saml_2.0.git to make the branch available to others.

How-to: Request a Merge

After pushing your branch, you can include its URL in an e-mail you send to the maintainer, requesting a merge of your branch, or in a comment or description of an issue in the tracker.

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*