python-anyvcs Documentation

Release 1.4.0

Scott Duckworth

Sep 27, 2017

Contents

1 Getting Started **3**

2 Contents **5**
2.1 The primary API 5
2.2 Git-specific functionality 10
2.3 Mercurial-specific functionality 10
2.4 Subversion-specific functionality 11

3 Indices and tables **13**

Python Module Index **15**

python-anyvcs is an abstraction layer for the homogenous, local handling of:

- Bare and non-bare Git repositories
- Mercurial repositories
- Subversion repositories (what `svnadmin create` makes)

CHAPTER 1

Getting Started

Here's a simple example for an existing repository:

```
>>> import anyvcs
>>> repo = anyvcs.open('/path/to/repo')
```

`repo` is an instance of *`anyvcs.common.VCSRepo`* with a variety of operations available for it.

CHAPTER 2

Contents

The primary API

Opening and Creating

In addition to instantiating subclasses of `anyvcs.common.VCSRepo` directly, you can also these utility functions which will infer the type based on the given parameters.

`anyvcs.open`(*path*, *vcs=None*)

Open an existing repository

Parameters

- **path** (*str*) – The path of the repository
- **vcs** – If specified, assume the given repository type to avoid auto-detection. Either `git`, `hg`, or `svn`.

Raises **UnknownVCSType** – if the repository type couldn't be inferred

If `vcs` is not specified, it is inferred via `probe()`.

`anyvcs.probe`(*path*)

Probe a repository for its type.

Parameters **path** (*str*) – The path of the repository

Raises **UnknownVCSType** – if the repository type couldn't be inferred

Returns **str** either `git`, `hg`, or `svn`

This function employs some heuristics to guess the type of the repository.

`anyvcs.clone`(*srcpath*, *destpath*, *vcs=None*)

Clone an existing repository.

Parameters

- **srcpath** (*str*) – Path to an existing repository

- **destpath** (*str*) – Desired path of new repository
- **vcs** (*str*) – Either `git`, `hg`, or `svn`

Returns VCSRepo The newly cloned repository

If `vcs` is not given, then the repository type is discovered from `srcpath` via *probe()*.

`anyvcs.`**`create`**(*path*, *vcs*)

Create a new repository

Parameters

- **path** (*str*) – The path where to create the repository.
- **vcs** (*str*) – Either `git`, `hg`, or `svn`

VCSRepo

class `anyvcs.common.`**`VCSRepo`**(*path*, *encoding='utf-8'*)

The most base type

`ancestor`(*rev1*, *rev2*)

Find most recent common ancestor of two revisions

Parameters

- **rev1** – First revision.
- **rev2** – Second revision.

Returns The common ancestor revision between the two.

`blame`(*rev*, *path*)

Blame (a.k.a. annotate, praise) a file

Parameters

- **rev** – The revision to blame.
- **path** (*str*) – The path to blame.

Returns list of annotated lines of the given path

Return type list of *BlameInfo* objects

Raises

- **PathDoesNotExist** – if the path does not exist.
- **BadFileType** – if the path is not a file.

`branches`()

Get list of branches

Returns The branches in the repository

Return type list of str

`canonical_rev`(*rev*)

Get the canonical revision identifier

Parameters **rev** – The revision to canonicalize.

Returns The canonicalized revision

The canonical revision is the revision which is natively supported by the underlying VCS type. In some cases, anyvcs may annotate a revision identifier to also encode branch information which is not safe to use directly with the VCS itself (e.g. as created by `compose_rev()`). This method is a means of converting back to canonical form.

cat (*rev*, *path*)
Get file contents

Parameters

- **rev** – The revision to use.
- **path** (*str*) – The path to the file. Must be a file.

Returns The contents of the file.

Return type str or bytes

Raises

- **PathDoesNotExist** – If the path does not exist.
- **BadFileType** – If the path is not a file.

changed (*rev*)
Files that changed from the rev's parent(s)

Parameters **rev** (list of `FileChangeInfo`.) – The revision to get the files that changed.

compose_rev (*branch*, *rev*)
Compose a revision identifier which encodes branch and revision.

Parameters

- **branch** (*str*) – A branch name
- **rev** – A revision (can be canonical or as constructed by `compose_rev()` or `tip()`)

The revision identifier encodes branch and revision information according to the particular VCS type. This is a means to unify the various branching models under a common interface.

diff (*rev_a*, *rev_b*, *path=None*)
Diff of two revisions

Parameters

- **rev_a** – The start revision.
- **rev_b** – The end revision.
- **path** (*None or str*) – If not None, return diff for only that file.

Returns str The diff.

The returned string contains the unified diff from rev_a to rev_b with a prefix of one (suitable for input to patch -p1).

empty ()
Test if the repository contains any commits

Returns bool True if the repository contains no commits.

Commits that exist by default (e.g. a zero commit) are not counted.

heads ()
Get list of heads

Returns The heads in the repository

Return type list of str

log (*revrange=None*, *limit=None*, *firstparent=False*, *merges=None*, *path=None*, *follow=False*)
Get commit logs

Parameters

- **revrange** – Either a single revision or a range of revisions as a 2-element list or tuple.
- **limit** (*int*) – Limit the number of log entries.
- **firstparent** (*bool*) – Only follow the first parent of merges.
- **merges** (*bool*) – True means only merges, False means no merges, None means both merges and non-merges.
- **path** (*str*) – Only match commits containing changes on this path.
- **follow** (*bool*) – Follow file history across renames.

Returns log information

Return type *CommitLogEntry* or list of *CommitLogEntry*

If revrange is None, return a list of all log entries in reverse chronological order.

If revrange is a single revision, return a single log entry.

If revrange is a 2 element list [A,B] or tuple (A,B), return a list of log entries starting at B and following that branch back to A or one of its ancestors (not inclusive. If A is None, follow branch B back to the beginning of history. If B is None, list all descendants in reverse chronological order.

ls (*rev*, *path*, *recursive=False*, *recursive_dirs=False*, *directory=False*, *report=()*)
List directory or file

Parameters

- **rev** – The revision to use.
- **path** – The path to list. May start with a '/' or not. Directories may end with a '/' or not.
- **recursive** – Recursively list files in subdirectories.
- **recursive_dirs** – Used when recursive=True, also list directories.
- **directory** – If path is a directory, list path itself instead of its contents.
- **report** – A list or tuple of extra attributes to return that may require extra processing. Recognized values are 'size', 'target', 'executable', and 'commit'.

Returns a list of dictionaries with the following keys:

type The type of the file: 'f' for file, 'd' for directory, 'l' for symlink.

name The name of the file. Not present if directory=True.

size The size of the file. Only present for files when 'size' is in report.

target The target of the symlink. Only present for symlinks when 'target' is in report.

executable True if the file is executable, False otherwise. Only present for files when 'executable' is in report.

Raises PathDoesNotExist if the path does not exist.

pdiff (*rev*)
Diff from the rev's parent(s)

Parameters **rev** – The rev to compute the diff from its parent.

Returns str The diff.

The returned string is a unified diff that the rev introduces with a prefix of one (suitable for input to patch -p1).

private_path

Get the path to a directory which can be used to store arbitrary data

This directory should not conflict with any of the repository internals. The directory should be created if it does not already exist.

readlink(*rev*, *path*)

Get symbolic link target

Parameters

- **rev** – The revision to use.
- **path** (*str*) – The path to the file. Must be a symbolic link.

Returns str The target of the symbolic link.

Raises

- **PathDoesNotExist** – if the path does not exist.
- **BadFileType** – if the path is not a symbolic link.

tags()

Get list of tags

Returns The tags in the repository

Return type list of str

tip(*head*)

Find the tip of a named head

Parameters **head** (*str*) – name of head to look up

Returns revision identifier of head

The returned identifier should be a valid input for `VCSRepo.ls()`. and respect the branch name in the returned identifier if applicable.

BlameInfo

class anyvcs.common.**BlameInfo**(*rev*, *author*, *date*, *line*)

Represents an annotated line in a file for a blame view.

Variables

- **rev** – Revision at which the line was last changed
- **author** (*str*) – Author of the change
- **date** (*datetime*) – Timestamp of the change
- **line** (*str*) – Line data from the file.

CommitLogEntry

class `anyvcs.common.`**`CommitLogEntry`**(*rev*, *parents*, *date*, *author*, *message*)

Represents a single entry in the commit log

Variables

- **rev** – Revision name
- **parents** – Parents of the revision
- **date** (*datetime*) – Timestamp of the revision
- **author** (*str*) – Author of the revision
- **message** (*str*) – Message from committer

subject

First line of the commit message.

FileChangeInfo

class `anyvcs.common.`**`FileChangeInfo`**(*path*, *status*, *copy=None*)

Represents a change to a single path.

Variables

- **path** (*str*) – The path that was changed.
- **status** (*str*) – VCS-specific code for the change type.
- **copy** – The source path copied from, if any.

Git-specific functionality

GitRepo

class `anyvcs.git.`**`GitRepo`**(*path*, *encoding='utf-8'*)

A git repository

Valid revisions are anything that git considers as a revision.

Mercurial-specific functionality

HgRepo

class `anyvcs.hg.`**`HgRepo`**(*path*, *encoding='utf-8'*)

A Mercurial repository

Valid revisions are anything that Mercurial considers as a revision.

Subversion-specific functionality

SvnRepo

class `anyvcs.svn.SvnRepo`(*path*)

A Subversion repository

Unless otherwise specified, valid revisions are:

- an integer (ex: 194)
- an integer as a string (ex: "194")
- a branch or tag name (ex: "HEAD", "trunk", "branches/branch1")
- a branch or tag name at a specific revision (ex: "trunk:194")

Revisions have the following meanings:

- HEAD always maps to the root of the repository (/)
- **Anything else (ex: "trunk", "branches/branch1") maps to the corresponding** path in the repository
- The youngest revision is assumed unless a revision is specified

For example, the following code will list the contents of the directory branches/branch1/src from revision 194:

```
>>> repo = SvnRepo(path)
>>> repo.ls('branches/branch1:194', 'src')
```

Branches and tags are detected in branches() and tags() by looking at the paths specified in repo.branch_glob and repo.tag_glob. The default values for these variables will detect the following repository layout:

- /trunk - the main development branch
- /branches/* - branches
- /tags/* - tags

If a repository does not fit this layout, everything other than branch and tag detection will work as expected.

dump(*stream*, *progress=None*, *lower=None*, *upper=None*, *incremental=False*, *deltas=False*)

Dump the repository to a dumpfile stream.

Parameters

- **stream** – A file stream to which the dumpfile is written
- **progress** – A file stream to which progress is written
- **lower** – Must be a numeric version number
- **upper** – Must be a numeric version number

See `svnadmin help dump` for details on the other arguments.

load(*stream*, *progress=None*, *ignore_uuid=False*, *force_uuid=False*, *use_pre_commit_hook=False*, *use_post_commit_hook=False*, *parent_dir=None*)

Load a dumpfile stream into the repository.

Parameters

- **stream** – A file stream from which the dumpfile is read
- **progress** – A file stream to which progress is written

See `svnadmin help load` for details on the other arguments.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

anyvcs, 5
anyvcs.common, 6
anyvcs.git, 10
anyvcs.hg, 10
anyvcs.svn, 11

Index

A

ancestor() (anyvcs.common.VCSRepo method), 6
anyvcs (module), 5
anyvcs.common (module), 6
anyvcs.git (module), 10
anyvcs.hg (module), 10
anyvcs.svn (module), 11

B

blame() (anyvcs.common.VCSRepo method), 6
BlameInfo (class in anyvcs.common), 9
branches() (anyvcs.common.VCSRepo method), 6

C

canonical_rev() (anyvcs.common.VCSRepo method), 6
cat() (anyvcs.common.VCSRepo method), 7
changed() (anyvcs.common.VCSRepo method), 7
clone() (in module anyvcs), 5
CommitLogEntry (class in anyvcs.common), 10
compose_rev() (anyvcs.common.VCSRepo method), 7
create() (in module anyvcs), 6

D

diff() (anyvcs.common.VCSRepo method), 7
dump() (anyvcs.svn.SvnRepo method), 11

E

empty() (anyvcs.common.VCSRepo method), 7

F

FileChangeInfo (class in anyvcs.common), 10

G

GitRepo (class in anyvcs.git), 10

H

heads() (anyvcs.common.VCSRepo method), 7
HgRepo (class in anyvcs.hg), 10

L

load() (anyvcs.svn.SvnRepo method), 11
log() (anyvcs.common.VCSRepo method), 8
ls() (anyvcs.common.VCSRepo method), 8

O

open() (in module anyvcs), 5

P

pdiff() (anyvcs.common.VCSRepo method), 8
private_path (anyvcs.common.VCSRepo attribute), 9
probe() (in module anyvcs), 5

R

readlink() (anyvcs.common.VCSRepo method), 9

S

subject (anyvcs.common.CommitLogEntry attribute), 10
SvnRepo (class in anyvcs.svn), 11

T

tags() (anyvcs.common.VCSRepo method), 9
tip() (anyvcs.common.VCSRepo method), 9

V

VCSRepo (class in anyvcs.common), 6