
anyvc Documentation

Release 0.3.7

Pida Team

February 05, 2015

1	About	3
2	Workdir Operations	5
2.1	Workdir Api Examples	5
2.2	Workdir Api	5
3	Repository Operations	9
3.1	Repository Api	9
4	VCS Abstraction Backends	11
4.1	Mercurial	11
4.2	Git	11
4.3	Subversion	11
5	Internal Details	13
5.1	Per Backend Metadata	13
6	Roadmap	15
6.1	wanted features	15
6.2	Status	15
7	The Testing Process	17
7.1	Workdir Testcases	17
7.2	Testing Utilities	17
8	Indices and tables	19

Contents:

About

Anycv is a library to abstract common vcs operations. It was born in an effort to enhance vcs operations in [PIDA](#).

The current version is mainly tailored to working with the working directories of the different vcs's and performing operations like adding/renaming/moving files, showing differences to the current commit and creating new commits.

It's still in the early stages of development, but has already proved its practical value in the version control service of [PIDA](#).

Future versions will gradually expand the scope from just workdir to interacting with history as well as managing repositories and branches.

Due to the differences in the vcs's not all operations are available on all vcs's, the abstraction will degrade/warn/error in those cases.

Workdir Operations

The workdir handling is accessible as an api as well as a rather simple pretty much feature-free cli.

2.1 Workdir Api Examples

2.1.1 Interactive Example Session

Lets begin by setting up some essential basics:

```
>>> from py.path import local
>>> from anyvc import workdir
>>> path = local('~/.Projects/anyvc')
>>> wd = workdir.open(path)
```

Now lets add a file:

```
>>> path.join('new-file.txt').write('test')
>>> wd.add(paths=['new-file.txt'])
```

Paths can be relative to the workdir, absolute paths, or *py.path.local* instances.

Now lets take a look at the list of added files:

```
>>> [s for s in wd.status() if s.state=='added']
[<added 'new-file.txt'>]
```

Since we seem to be done lets commit:

```
>>> wd.commit(
...     message='test',
...     paths=['new-file.txt'],
... )
```

Since the change is committed the list of added files is empty now:

```
>>> [s for s in wd.status() if s.state=='added']
[]
```

2.2 Workdir Api

`open(path, dont_try=())`

Parameters **path** – a local path to the worktree preferable a *py.path.local* instance

open a scm workdir

It uses the backend metadata to find the correct backend and won't import unnecessary backends to keep the import time low

checkout (*source, target*)

create a light checkout of the given source

clone (*source, target*)

create a heavy checkout/clone of the given source

class WorkDir (*path, create=False, source=None*)

Basic Workdir API

Parameters

- **path** – base path
- **create** –

commit (*paths=(), message=None, user=None*)

Parameters

- **path** – the paths
- **message** – the commit message
- **user** – optional author name

commits the given paths/files with the given commit message and author

diff (*paths=()*)

given a list of paths it will return a diff

process_paths (*paths*)

preprocess given paths

status (*paths=(), recursive=True*)

Parameters

- **path** (*sequence of string*) – the filenames
- **recursive** (*bool*) – proceed recursive for directories

yield a list of Path instances tagged with status informations

update (*paths=(), revision=None*)

Parameters **revision** – the target revision may not actually work for vcs's with tricky workdir revision setups

updates the workdir to either the closest head or or the given revision

class WorkDirWithParser (*path, create=False, source=None*)

extension of the workdir class to support parsing needs

cache (*paths=(), recursive=False*)

return a mapping of name to cached states only necessary for messed up vcs's

cache_impl (*paths=False, recursive=False*)

creates a list of vcs specific cache items only necessary by messed up vcs's

in case of doubt - dont touch ^^

parse_cache_items (*items*)

parses vcs specific cache items to a list of (name, state) tuples

parse_status_item (*item, cache*)

parse a single status item meant to be overridden

parse_status_items (*items, cache*)

default implementation

for each *item* in *items* invoke:

```
self.parse_status_item(item, cache)
```

Note: a more complex parser might need to overwrite

status (*paths=(), recursive=True*)

yield a list of Path instances tagged with status informations

status_impl (*paths=False, recursive=False*)

yield a list of vcs specific listing items

class StatedPath (*name, state='normal', base=None*)

stores status informations about files

```
>>> StatedPath('a.txt')
<normal 'a.txt'>
>>> StatedPath('a.txt', 'changed')
<changed 'a.txt'>
```

Repository Operations

3.1 Repository Api

open (*path*, *backends=None*)

Parameters **backends** – optional list of backends to try

open a repository backend at the given path

find (*root*, *backends=None*)

Parameters **root** (*py.path.local* or *path string*) – the search root

find all repositories below *root*

class Repository (*path*, ***extra*)

represents a repository

prepare_default_structure ()

if the vcs has a common standard repo structure, set it up

pull (*source=None*, *rev=None*)

counterpart to push

push (*dest=None*, *rev=None*)

push to a location

Parameters

- **dest** – the destination
- **rev** – the maximum revision to push, may be none for latest

class Revision

id

The revision id the vcs gave this commit

Type int or string

VCS Abstraction Backends

Currently anyvc ships with support for

4.1 Mercurial

The Mercurial backend is implemented in Terms of the basic Merucrial api. It does not support extension discovery or extensions.

4.2 Git

The Git backend is split. Workdir support is implemented in terms of the git CLI because Dulwich has no complete support. Workdirs are still agnostic to the existence of the git index. Repository support is implemented in terms of Dulwich, cause its supported and the better 'api'.

4.3 Subversion

The Subversion backend is split as well. The workdir part is implemented in terms of the CLI, because the Subversion checkout api requires complicated locking patterns. The Repository support is implemented in terms of subvertpy.

Internal Details

Following is supposed to be helpful information for debugging.

5.1 Per Backend Metadata

Backend metadata is located in each backend's `__init__.py`.

currently the following variables are used:

repo class the full name of the repository class in setuptools notation

workdir class the full name of the workdir class in setuptools notation

workdir control the name of the directory that identifies a workdir

Other required (but not yet implemented) fields

repo_control lists sets of paths that will exist in a repository

repo features same in green

repo commands required executables for repo to function propper

repo modules required modules to function propper

serving_class the full name of the repository serving class in setuptools notation

workdir features stuff the repo can do like graph, merge, props

workdir commands required executables for repo to function propper

workdir modules required modules to function propper

license the license of the backend code (would help with avoiding license problems)

Roadmap

6.1 wanted features

workdir control common ops to change the state of the workingtree

workdir status get the file states of the worktree

repo access find repos, get worktrees from them

histbrowse work with the history

branchman manage branches

6.2 Status

VCS	Workdir	Repo	histbrowse	branchman
hg	yes	partial	no	no
svn	yes	partial	no	no
git	messy	partial	no	no

The Testing Process

Anyvc and its backends are developed using TDD. If you want to develop additional backends it is important to understand the details of the general test running process as well as the specific testcases.

7.1 Workdir Testcases

7.2 Testing Utilities

7.2.1 additional py.test options

--vcs {name}
limit the testing for backends to the given vcs

--local-remoting
if given also test the local remoting

--no-direct-api
Don't run the normal local testing, useful for remote-only

7.2.2 pytest funcargs

7.2.3 Utility Classes

class VcsMan (*vc, base, backend*)

utility class to manage the creation of repositories and workdirs inside of a specific path (usually the tmpdir funcarg of a test)

base

Type `py.path.local`

the base directory

vc

The name of the managed vcs

backend

Type `anyvc.backend.Backend`

The backend instance giving access to the currently tested vcs

remote

boolean telling if the remoting support is used

xspec

A `execnet.XSpec` telling remote python if remoting is used

create_wd (*workdir*, *source=None*)

Parameters

- **workdir** (*str*) – name of the target workdir
- **source** (*repo or None*) – name of a source repository

create a workdir if *source* is given, use it as base

make_repo (*name*)

Parameters **name** – name of the repository to create

create a repository using the given name

class WdWrap (*wd*)

Parameters **wd** (subclass of `anyvc.common.workdir.Workdir`) – the workdir to wrap

decorator for a vcs workdir instance adds testing utility functions and proxies the other methods/attributes to the real instance

check_states (*exact=True*, ***kw*)

Parameters

- **exact** (*bool*) – if true, ignore additional states
- **\$statername** (*list of relative path*) – state name for that particular file list

Returns True if all supplied files have the assumed state

delete_files (**relpaths*)

Parameters **relpaths** – listing of files to remove

has_files (**files*)

Parameters **files** – a listing of filenames that should exist

put_files (*mapping*)

the text content will be stripped and get a newline appended

Indices and tables

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

Symbols

-local-remoting
 py.test command line option, 17
 -no-direct-api
 py.test command line option, 17
 -vcs {name}
 py.test command line option, 17

B

backend (VcsMan attribute), 17
 base (VcsMan attribute), 17

C

cache() (WorkDirWithParser method), 6
 cache_impl() (WorkDirWithParser method), 6
 check_states() (WdWrap method), 18
 checkout() (in module anyvc.workdir), 6
 clone() (in module anyvc.workdir), 6
 commit() (WorkDir method), 6
 create_wd() (VcsMan method), 18

D

delete_files() (WdWrap method), 18
 diff() (WorkDir method), 6

F

find() (in module anyvc.repository), 9

H

has_files() (WdWrap method), 18

I

id (Revision attribute), 9

M

make_repo() (VcsMan method), 18

O

open() (in module anyvc.repository), 9

open() (in module anyvc.workdir), 5

P

parse_cache_items() (WorkDirWithParser method), 6
 parse_status_item() (WorkDirWithParser method), 7
 parse_status_items() (WorkDirWithParser method), 7
 prepare_default_structure() (Repository method), 9
 process_paths() (WorkDir method), 6
 pull() (Repository method), 9
 push() (Repository method), 9
 put_files() (WdWrap method), 18
 py.test command line option
 -local-remoting, 17
 -no-direct-api, 17
 -vcs {name}, 17

R

remote (VcsMan attribute), 17
 Repository (class in anyvc.common.repository), 9
 Revision (class in anyvc.common.repository), 9

S

StatedPath (class in anyvc.common.workdir), 7
 status() (WorkDir method), 6
 status() (WorkDirWithParser method), 7
 status_impl() (WorkDirWithParser method), 7

U

update() (WorkDir method), 6

V

vc (VcsMan attribute), 17
 VcsMan (class in tests.helpers), 17

W

WdWrap (class in tests.helpers), 18
 WorkDir (class in anyvc.common.workdir), 6
 WorkDirWithParser (class in anyvc.common.workdir), 6

X

xspec (VcsMan attribute), 18