anyvc Documentation

Release 0.3.5+20120606

Pida Team

June 06, 2012

CONTENTS

Contents:

CHAPTER

ONE

ABOUT

Anyvc 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.

CHAPTER

TWO

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 commited the list of added files is empty now:

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

2.2 Workdir Api

open (*path*)

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'>
```

CHAPTER

THREE

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`**(***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

CHAPTER

FOUR

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 Bazaar

The Bazaar backend is implemented in terms of bzrlib. It is to be considered as ‘passes the tests’ not as first class citizen

4.4 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.

CHAPTER

FIVE

INTERNAL DETAILS

Following is supposed to be helful 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 reposity 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)

CHAPTER

SIX

ROADMAP

6.1 wanted features

wordir 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
bzr	yes	partial	no	no
svn	yes	partial	no	no
git	messy	partial	no	no

CHAPTER

SEVEN

THE TESTING PROCESS

Anyvc an 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

pytest_funcarg__backend(*request*)
: create a cached backend instance that is used the whole session makes instanciating backend cheap

pytest_funcarg__mgr(*request*)
: create a preconfigured `tests.helplers.VcsMan` instance pass the currently tested backend and create a tmpdir for the vcs/test combination

  auto-check for the vcs features and skip if necessary

pytest_funcarg__repo(*request*)
: create a repo below mgf called 'repo'

pytest_funcarg__wd(*request*)
: create a workdir below mgr called 'wd' if the feature "wd:heavy" is not supported use repo as help

7.2.3 Utility Classes

class **`VcsMan`** (*vc*, *base*, *xspec*, *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
- **$statename** (*list of relative path*) – state name for that particular file list

Returns True if all supplied files have the asumed state

`delete_files` (**relpaths*)

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

`has_files` (**files*)

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

`put_files` (*mapping*)

the text content will be rstripped and get a newline appended

CHAPTER

EIGHT

INDICES AND TABLES

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

PYTHON MODULE INDEX