vcs is an abstraction layer over various version control systems. It is designed as feature-rich Python library with clear API Reference.

vcs uses Semantic Versioning

Features

- Common API for SCM backends
- Fetching repositories data lazily
- Simple caching mechanism so we don’t hit repo too often
- In memory commits API
- Command Line Interface

Incoming

- Full working directories support
- Extra backends: Subversion, Bazaar
Installation:

1.1 Quickstart

Say you don’t want to install vcs or just want to begin with really fast tutorial? Not a problem, just follow sections below.

1.1.1 Prepare

We will try to show you how you can use vcs directly on repository. But hey, vcs is maintained within git repository already, so why not use it? Simply run following commands in your shell

```bash
cd /tmp
git clone git://github.com/codeinn/vcs.git
cd vcs
```

Now run your python interpreter of choice:

```bash
$ python
>>> 
```

**Note:** You may of course put your clone of vcs wherever you like but running python shell inside of it would allow you to use just cloned version of vcs.

1.1.2 Take the shortcut

There is no need to import everything from vcs - in fact, all you’d need is to import get_repo, at least for now. Then, simply initialize repository object by providing it’s type and path.

```bash
>>> import vcs
>>> # create repository representation at current dir
>>> repo = vcs.get_repo(path='')
```

**Note:** In above example we didn’t specify scm. We can provide as second argument to the get_repo function, i.e. `get_repo('','hg')`.  

1.1.3 Basics

Let's ask repo about the content...

```python
>>> root = repo.get_changeset().get_node('')
>>> print root.nodes # prints nodes of the RootNode
[<DirNode ''>, <DirNode 'docs'>, <DirNode 'tests'>, # ... (chopped)
>>> # get 10th changeset
>>> chset = repo.get_changeset(10)
>>> print chset
<GitChangeset at 10:d955cd312c17>
>>> # any backend would return latest changeset if revision is not given
>>> tip = repo.get_changeset()
>>> tip == repo.get_changeset('tip') # for git/mercurial backend 'tip' is allowed
True
>>> tip == repo.get_changeset(None) # any backend allow revision to be None (default)
True
>>> tip.raw_id == repo.revisions[-1]
True
>>> # Iterate repository
>>> for cs in repo:
...    print cs
... ...
>>> <GitChangeset at 0:c1214f7e79e0>
>>> <GitChangeset at 1:38b5fe81f109>
>>> ...
```

1.1.4 Walking

Now let's ask for nodes at revision faebbb751cc36c137127c50f57bcd5f1c540013 (https://github.com/codeinn/vcs/commit/faebbb751cc36c137127c50f57bcd5f1c540013)

```python
>>> chset = repo.get_changeset('faebbb751cc36c137127c50f57bcd5f1c540013')
>>> root = chset.root
>>> print root.dirs
[<DirNode 'docs'>, <DirNode 'tests'>, <DirNode 'vcs'>]
```

Note: `vcs.nodes` are objects representing files and directories within the repository revision.

```python
>>> # Fetch vcs directory
>>> vcs = repo.get_changeset('faebbb751cc36c137127c50f57bcd5f1c540013').get_node('vcs')
>>> print vcs.dirs
[<DirNode 'vcs/backends'>,
 <DirNode 'vcs/utils'>,
 <DirNode 'vcs/web'>]
```

```python
>>> backends_node = vcs.dirs[0]
>>> print backends_node.nodes
[<FileNode 'vcs/backends/__init__.py'>,
 <FileNode 'vcs/backends/base.py'>,
 <FileNode 'vcs/backends/git.py'>,
 <FileNode 'vcs/backends/hg.py'>]
```
1.1.5 Getting meta data

Make vcs show us some meta information

Tags and branches

```python
class repo.branche
OrderedDict([('master', 'fe568b4081755c12abf6ba673ba777fc02a415f3')])
```

```python
for tag, raw_id in repo.tags.items():
    print tag.rjust(10), |', raw_id
```

<table>
<thead>
<tr>
<th>Version</th>
<th>Raw ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0.1.9</td>
<td>341d28f0e6c5d6f0b6b77871e13c2bbb6b6bce685c</td>
</tr>
<tr>
<td>v0.1.8</td>
<td>74ebce002c088b8a5ecf40073d09375515ecd68</td>
</tr>
<tr>
<td>v0.1.7</td>
<td>4d78bf73b5c22c82b6f902f13f7881b4fffa2c</td>
</tr>
<tr>
<td>v0.1.6</td>
<td>0205cb3f44223f3b3099d12a7a96c81b798772d9</td>
</tr>
<tr>
<td>v0.1.5</td>
<td>60c0ce52b229a978889e91b38777f800e85f330b</td>
</tr>
<tr>
<td>v0.1.4</td>
<td>7d75150934cd7645ac3051903add952390324a5</td>
</tr>
<tr>
<td>v0.1.3</td>
<td>5a3a8f0b00554692b16e21de62b026677d8d3e</td>
</tr>
<tr>
<td>v0.1.2</td>
<td>0b058a466034ff25c0ca25b0aaf05d2791d63</td>
</tr>
<tr>
<td>v0.1.1</td>
<td>c60f01b777c42dce653d6b1d3b04689862c261929</td>
</tr>
<tr>
<td>v0.1.10</td>
<td>10cde8f6b794696066fb346434014f0a5668102b8</td>
</tr>
<tr>
<td>v0.1.11</td>
<td>e6ea6d16e2f26250124a1f4b4fe37a912f9d86a0</td>
</tr>
</tbody>
</table>

Give me a file, finally!

```python
import vcs
repo = vcs.get_repo('')
chset = repo.get_changeset('faebbb751cc36c137127c50f57bdc5f1c540013')
root = chset.get_node('')
backends = root.get_node('vcs/backends')
backends.files
```

<table>
<thead>
<tr>
<th>File Name</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>.py</td>
<td>vcs/backends/<strong>init</strong>.py</td>
</tr>
<tr>
<td>base.py</td>
<td>vcs/backends/base.py</td>
</tr>
<tr>
<td>git.py</td>
<td>vcs/backends/git.py</td>
</tr>
<tr>
<td>hg.py</td>
<td>vcs/backends/hg.py</td>
</tr>
</tbody>
</table>

```python
f = backends.get_node('hg.py')
f.name
hg.py
>>> f.path
'vcs/backends/hg.py'
```
u‘fixed bug in get_changeset when 0 or None was passed’
>>> f.last_changeset.author
u‘marcinkuzminski <none@none>’
>>> f.mimetype
‘text/x-python’
>>> # Following would raise exception unless you have pygments installed
>>> f.lexer
<pygments.lexers.PythonLexer>
>>> f.lexer_alias
# shortcut to get first of lexers' available aliases
‘python’
>>> # wanna go back? why? oh, whatever...
>>> f.parent
<DirNode 'vcs/backends'>
...>
>>> # is it cached? Hell yeah...
>>> f is f.parent.get_node(‘hg.py’) is chset.get_node(‘vcs/backends/hg.py’)
True

How about history?

It is possible to retrieve changesets for which file node has been changed and this is pretty damn simple. Let’s say we want to see history of the file located at vcs/nodes.py.

```python
>>> f = repo.get_changeset().get_node(‘vcs/nodes.py’)
>>> for cs in f.history:
...   print cs
...
<GitChangeset at 440:40a2d5d71b75>
<GitChangeset at 438:df89326327>
<GitChangeset at 420:162a36830c23>
<GitChangeset at 345:c994f0de0b2>
<GitChangeset at 340:5d34d2c262e>
<GitChangeset at 334:4d4278a6390e>
<GitChangeset at 298:00dffe625166>
<GitChangeset at 297:47b6be9a812e>
<GitChangeset at 289:1589fe6841cd>
<GitChangeset at 285:afafede2821>
<GitChangeset at 284:639b115ed2bo>
<GitChangeset at 283:fc7562d7305>
<GitChangeset at 256:ec8cbdb5f364>
<GitChangeset at 255:0742e2e2bf3f>
<GitChangeset at 243:6894a7d8223>
<GitChangeset at 231:31b34b599fa>
<GitChangeset at 220:3d2515dd21f>
<GitChangeset at 186:f804e27aa496>
<GitChangeset at 182:7f00513785a1>
<GitChangeset at 181:6efc3e61028c>
<GitChangeset at 175:600ce52b256a>
<GitChangeset at 165:0978e0aa8a54>
<GitChangeset at 163:0164ee729def>
<GitChangeset at 140:33fa32233551>
<GitChangeset at 126:fa014c32cd62>
<GitChangeset at 111:e686b958768e>
<GitChangeset at 109:ab5721ca0a08>
<GitChangeset at 108:c877b68d18e7>
<GitChangeset at 107:4313566d2e41>
<GitChangeset at 104:6c2303a79367>
```
Note that *history* attribute is computed lazily and returned list is reversed - changesets are retrieved from most recent to oldest.

**Show me the difference!**

Here we present naive implementation of diff table for the given file node located at `vcs/nodes.py`. First we have to get the node from repository. After that we retrieve last changeset for which the file has been modified and we create a html file using `difflib`.

```python
>>> new = repo.get_changeset(repo.tags['v0.1.11'])
>>> old = repo.get_changeset(repo.tags['v0.1.10'])
>>> f_old = old.get_node('vcs/nodes.py')
>>> f_new = new.get_node('vcs/nodes.py')
>>> out = open('/tmp/out.html', 'w')
>>> from difflib import HtmlDiff
>>> hd = HtmlDiff(tabsize=4)
>>> diffs = hd.make_file(f_new.content.split('
'), f_old.content.split('
'))
>>> out.write(diffs)
>>> out.close()
```

Now open file at `/tmp/out.html` in your favorite browser.

### 1.2 Installation

`vcs` is simply, pure python package. However, it makes use of various *version control systems* and thus, would require some third part libraries and they may have some deeper dependencies.
1.2.1 Requirements

Below is a table which shows requirements for each backend.

<table>
<thead>
<tr>
<th>SCM</th>
<th>Backend</th>
<th>Alias</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mercurial</td>
<td>vcs.backend.hg</td>
<td>hg</td>
<td>• mercurial &gt;= 1.9</td>
</tr>
<tr>
<td>Git</td>
<td>vcs.backend.git</td>
<td>git</td>
<td>• git &gt;= 1.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Dulwich &gt;= 0.8</td>
</tr>
</tbody>
</table>

1.2.2 Install from Cheese Shop

Easiest way to install vcs is to run:

easy_install vcs

Or:

pip install vcs

If you prefer to install manually simply grab latest release from http://pypi.python.org/pypi/vcs, decompress archive and run:

python setup.py install

1.2.3 Development

In order to test the package you’d need all backends underlying libraries (see table above) and unittest2 as we use it to run test suites.

Here is a full list of packages needed to run test suite:

<table>
<thead>
<tr>
<th>Package</th>
<th>Homepage</th>
</tr>
</thead>
<tbody>
<tr>
<td>mock</td>
<td><a href="http://pypi.python.org/pypi/mock">http://pypi.python.org/pypi/mock</a></td>
</tr>
<tr>
<td>unittest2</td>
<td><a href="http://pypi.python.org/pypi/unittest2">http://pypi.python.org/pypi/unittest2</a></td>
</tr>
<tr>
<td>mercurial</td>
<td><a href="http://mercurial.selenic.com/">http://mercurial.selenic.com/</a></td>
</tr>
<tr>
<td>git</td>
<td><a href="http://git-scm.com">http://git-scm.com</a></td>
</tr>
<tr>
<td>dulwich</td>
<td><a href="http://pypi.python.org/pypi/dulwich">http://pypi.python.org/pypi/dulwich</a></td>
</tr>
</tbody>
</table>

1.3 Usage

1.3.1 vcsrc

During commands execution, vcs tries to build a module specified at VCSRC_PATH. This would fail silently if module does not exist - user is responsible for creating own vcsrc file.

Creating own commands

User may create his own commands and add them dynamically by pointing them at vcs.cli.registry map. Here is very simple example of how vcsrc file could look like:
from vcs import cli

class AuthorsCommand(cli.ChangesetCommand):

    def pre_process(self, repo):
        self.authors = {}

    def handle_changeset(self, changeset, **options):
        if changeset.author not in self.authors:
            self.authors[changeset.author] = 0
            self.authors[changeset.author] += 1

    def post_process(self, repo, **options):
        for author, changesets_number in self.authors.iteritems():
            message = '%s : %s' % (author, changesets_number)
            self.stdout.write(message + '

cli.registry[''authors''] = AuthorsCommand

This would create AuthorsCommand that is mapped to authors subcommand. In order to run the command user would enter repository and type into terminal:

vcs authors

As we have subclassed ChangesetCommand, we also got all the changesets specified options. User may see whole help with following command:

vcs authors -h

Note: Please refer to vcs.cli for more information about the basic commands.

1.4 How to contribute

There are a lot of ways people may contribute to vcs. First of all, if you spot a bug please file it at issue tracker. Moreover, if you feel you can fix the problem on your own and want to contribute to vcs, just fork from your preferred scm and send us your pull request.

Note: Oh, some codes may be very ugly. If you spot ugly code, file a bug/clean it/ make it more readable/send us a note.

1.4.1 Repositories

As we do various version control systems, we also try to be flexible at where code resides and therefor, could be accessed by wider audience.

- Main git repository is at https://github.com/codeinn/vcs/
- Main Mercurial repository is at https://bitbucket.org/marcinkuzminski/vcs/

We are going to create one official repository per supported backend.
1.4.2 How to write backend

Don’t see your favorite SCM at supported backends but like vcs API? Writing your own backend is in fact very simple process - all the backends should extend from base backend, however, as there are a few classes that needs to be written (repository, changeset, in-memory-changeset, workingdir) one would probably want to review existing backends’ codebase.

1.4.3 Tests

Tests are fundamental to vcs development process. In fact we try to do TDD as much as we can, however it doesn’t always fit well with open source projects development. Nevertheless, we don’t accept patches without tests. So... test, damn it! Whole heavy-lifting is done for you already, anyway (unless you don’t intend to write new backend)!

1.5 Alternatives

There are a couple of alternatives to vcs:

- anyvc actively maintained, similar to vcs (in a way it tries to abstract scms), supports more backends (svn, bzr); as far as we can tell it’s main heart of Pida; it’s main focus however is on working directories, does not support in memory commits or history traversing;
- pyvcs not actively maintained; this package focus on history and repository traversing, does not support commits at all; is much simpler from vcs so may be used if you don’t need full repos interface

Note: If you know any other similar Python library, please let us know!

1.6 API Reference

1.6.1 vcs.cli

Command line interface for VCS

This module provides foundations for creating, executing and registering terminal commands for vcs. Moreover, ExecutionManager makes it possible for user to create own code (at .vcsrc file).

vcs.cli.ExecutionManager

class vcs.cli.ExecutionManager (argv=None, stdout=None, stderr=None)  
Class for command execution management.

    execute ()
        Executes whole process of parsing and running command.

    get_argv_for_command ()
        Returns stripped arguments that would be passed into the command.

    get_command_class (cmd)
        Returns command class from the registry for a given cmd.

            Parameters  cmd – command to run (key at the registry)
**get_commands()**
Returns commands stored in the registry.

**get_vcsr()**
Returns in-memory created module pointing at user’s configuration and extra code/commands. By default tries to create module from VCSRC_PATH.

**run_command(cmd, argv)**
Runs command.

**Parameters**
- `cmd` – command to run (key at the registry)
- `argv` – arguments passed to the command

**show_help()**
Prints help text about available commands.

---

**vcs.cli.BaseCommand**

*class* vcs.cli.BaseCommand (*stdout=None, stderr=None*)
Base command class.

**execute(*args, **options)**
Executes whole process of parsing arguments, running command and trying to catch errors.

**get_option_list()**
Returns options specified at `self.option_list`.

**get_parser(prog_name, subcommand)**
Returns parser for given `prog_name` and `subcommand`.

**Parameters**
- `prog_name` – vcs main script name
- `subcommand` – command name

**get_version()**
Returns version of vcs.

**handle(*args, **options)**
This method must be implemented at subclass.

**print_help(prog_name, subcommand)**
Prints parser’s help.

**Parameters**
- `prog_name` – vcs main script name
- `subcommand` – command name

**run_from_argv(argv)**
Runs command for given arguments.

**Parameters**
- `argv` – arguments

**usage(subcommand)**
Returns how to use command text.
vcs.cli.RepositoryCommand

class vcs.cli.RepositoryCommand (stdout=None, stderr=None, repo=None)
Base repository command.

Accepts extra argument:

Parameters repo – repository instance. If not given, repository would be calculated based on current directory.

handle (*args, **options)
Runs pre_process, handle_repo and post_process methods, in that order.

handle_repo (repo, *args, **options)
Handles given repository. This method must be implemented at subclass.

post_process (repo, **options)
This method would be run at the end of handle method. Does nothing by default.

pre_process (repo, **options)
This method would be run at the beginning of handle method. Does nothing by default.

vcs.cli.ChangesetCommand

class vcs.cli.ChangesetCommand (stdout=None, stderr=None, repo=None)
Subclass of RepositoryCommand.

Extra options

• show_progress_bar: specifies if bar indicating progress of processed changesets should be shown.

Accepts extra argument:

Parameters repo – repository instance. If not given, repository would be calculated based on current directory.

get_changesets (repo, **options)
Returns generator of changesets from given repo for given options.

Parameters repo – repository instance. Same as self.repo.

Available options

• start_date: only changesets not older than this parameter would be generated
• end_date: only changesets not younger than this parameter would be generated
• start: changeset’s ID from which changesets would be generated
• end: changeset’s ID to which changesets would be generated
• branch: branch for which changesets would be generated. If all flag is specified, this option would be ignored. By default, branch would be tried to retrieved from working directory.
• all: return changesets from all branches
• reversed: by default changesets are returned in date order. If this flag is set to True, reverse order would be applied.
• limit: if specified, show no more changesets than this value. Default is None.

get_progressbar (total, **options)
Returns progress bar instance for a given total number of clicks it should do.
handle_changeset(changeset, **options)

Handles single changeset. Must be implemented at subclass.

vcs.cli.SingleChangesetCommand

class vcs.cli.SingleChangesetCommand(stdout=None, stderr=None, repo=None)

Single changeset command. Convenient if command has to operate on single changeset rather than whole
generator. For usage i.e. with command that handles node(s) from single changeset.

Extra options

• min_args: minimal number of arguments to parse. Default is 1.

Accepts extra argument:

Parameters repo – repository instance. If not given, repository would be calculated based on current
directory.

get_changeset(**options)

Returns changeset for given options.

handle_arg(changeset, arg, **options)

Handles single argument for chosen changeset. Must be implemented at subclass.

Parameters

• changeset – chosen (by --commit option) changeset
• arg – single argument from arguments list

1.6.2 vcs.conf.settings

ARCHIVE_SPECS

Dictionary with mapping of archive types to mimetypes.

Default:

```plaintext
{ 'tar': ('application/x-tar', '.tar'),
  'tbz2': ('application/x-bzip2', '.tar.bz2'),
  'tgz': ('application/x-gzip', '.tar.gz'),
  'zip': ('application/zip', '.zip'),
}
```

BACKENDS

Dictionary with mapping of scm aliases to backend repository classes.

Default:

```plaintext
{ 'hg': 'vcs.backends.hg.MercurialRepository',
  'git': 'vcs.backends.git.GitRepository',
}
```
VCSRC_PATH

Points at a path where ExecutionManager should look for module specified by user. By default it would be $HOME/.vimrc.

This value may be modified by setting system environment VCSRC_PATH (accessible at os.environ[‘VCSRC_PATH’]).

1.6.3 vcs.nodes

Node

class vcs.nodes.Node (path, kind)

Simplest class representing file or directory on repository. SCM backends should use FileNode and DirNode subclasses rather than Node directly.

Node’s path cannot start with slash as we operate on relative paths only. Moreover, every single node is identified by the path attribute, so it cannot end with slash, too. Otherwise, path could lead to mistakes.

get_parent_path ()

Returns node’s parent path or empty string if node is root.

is_dir ()

Returns True if node’s kind is NodeKind.DIR, False otherwise.

is_file ()

Returns True if node’s kind is NodeKind.FILE, False otherwise.

is_root ()

Returns True if node is a root node and False otherwise.

is_submodule ()

Returns True if node’s kind is NodeKind.SUBMODULE, False otherwise.

name

Returns name of the node so if its path then only last part is returned.

FileNode

class vcs.nodes.FileNode (path, content=None, changeset=None, mode=None)

Class representing file nodes.

Attribute path: path to the node, relative to repository’s root

Attribute content: if given arbitrary sets content of the file

Attribute changeset: if given, first time content is accessed, callback

Attribute mode: octal stat mode for a node. Default is 0100644.

Only one of content and changeset may be given. Passing both would raise NodeError exception.

Parameters

- path – relative path to the node
- content – content may be passed to constructor
- changeset – if given, will use it to lazily fetch content
- mode – octal representation of ST_MODE (i.e. 0100644)
annotate
Returns a list of three element tuples with lineno, changeset and line

test
Returns lazily content of the FileNode. If possible, would try to decode content from UTF-8.

test
Returns filenode extension

get_mimetype()
Mimetype is calculated based on the file’s content. If _mimetype attribute is available, it will be returned (backends which store mimetypes or can easily recognize them, should set this private attribute to indicate that type should NOT be calculated).

history
Returns a list of changeset for this file in which the file was changed

is_binary
Returns True if file has binary content.

is_executable()
Returns True if file has executable flag turned on.

lexen
Returns pygment’s lexer class. Would try to guess lexer taking file’s content, name and mimetype.

lexen_alias
Returns first alias of the lexer guessed for this file.

mimetype
Wrapper around full mimetype info. It returns only type of fetched mimetype without the encoding part. use get_mimetype function to fetch full set of (type, encoding)

mode
Returns lazily mode of the FileNode. If changeset is not set, would use value given at initialization or 0100644 (default).

RemovedFileNode

class vcs.nodes.RemovedFileNode (path)
Dummy FileNode class - trying to access any public attribute except path, name, kind or state (or methods/attributes checking those two) would raise RemovedFileNodeError.

Parameters path – relative path to the node

DirNode

class vcs.nodes.DirNode (path, nodes=(), changeset=None)
DirNode stores list of files and directories within this node. Nodes may be used standalone but within repository context they lazily fetch data within same repository’s changeset.

Only one of nodes and changeset may be given. Passing both would raise NodeError exception.

Parameters
• path – relative path to the node
• nodes – content may be passed to constructor
• changeset – if given, will use it to lazily fetch content
• **size** – always 0 for DirNode

**get_node** *(path)*

Returns node from within this particular DirNode, so it is now allowed to fetch, i.e. node located at ‘docs/api/index.rst’ from node ‘docs’. In order to access deeper nodes one must fetch nodes between them first - this would work:

```python
docs = root.get_node('docs')
docs.get_node('api').get_node('index.rst')
```

**Param**  
path - relative to the current node

**Note:** To access lazily (as in example above) node have to be initialized with related changeset object - without it node is out of context and may know nothing about anything else than nearest (located at same level) nodes.

---

### RootNode

**class** *vcs.nodes.RootNode**(nodes=(), changeset=None)*

DirNode being the root node of the repository.

---

### 1.6.4 vcs.backends

**Implemented Backends**

- **vcs.backends.git**
  - GitRepository
  - GitChangeset
  - GitInMemoryChangeset

- **vcs.backends.hg**
  - MercurialRepository

**class** *vcs.backends.hg.MercurialRepository**(repo_path, create=False, baseui=None, src_url=None, update_after_clone=False)*

Mercurial repository backend

 Raises RepositoryError if repository could not be find at the given repo_path.

**Parameters**

- **repo_path** – local path of the repository
- **create=False** – if set to True, would try to create repository if it does not exist rather than raising exception
- **baseui=None** – user data
- **src_url=None** – would try to clone repository from given location
• **update_after_clone=False** – sets update of working copy after making a clone

**allbranches**
List all branches, including closed branches.

**bookmarks**
Get’s bookmarks for this repository

**get_changset (revision=None)**
Returns MercurialChangeset object representing repository’s changeset at the given revision.

**get_changesets (start=None, end=None, start_date=None, end_date=None, branch_name=None, reverse=False)**
Returns iterator of MercurialChangeset objects from start to end (both are inclusive)

**Parameters**
• **start** – None, str, int or mercurial lookup format
• **end** – None, str, int or mercurial lookup format
• **start_date** –
• **end_date** –
• **branch_name** –
• **reversed** – return changesets in reversed order

**get_config_value (section, name=None, config_file=None)**
Returns configuration value for a given [section] and name.

**Parameters**
• **section** – Section we want to retrieve value from
• **name** – Name of configuration we want to retrieve
• **config_file** – A path to file which should be used to retrieve configuration from (might also be a list of file paths)

**get_diff (rev1, rev2, path='', ignore_whitespace=False, context=3)**
Returns (git like) diff, as plain text. Shows changes introduced by rev2 since rev1.

**Parameters**
• **rev1** – Entry point from which diff is shown. Can be self.EMPTY_CHANGESET - in this case, patch showing all the changes since empty state of the repository until rev2
• **rev2** – Until which revision changes should be shown.
• **ignore_whitespace** – If set to True, would not show whitespace changes. Defaults to False.
• **context** – How many lines before/after changed lines should be shown. Defaults to 3.

**get_hook_location ()**
returns absolute path to location where hooks are stored

**get_user_email (config_file=None)**
Returns user’s email from global configuration file.

**Parameters**
• **config_file** – A path to file which should be used to retrieve configuration from (might also be a list of file paths)

**get_user_name (config_file=None)**
Returns user’s name from global configuration file.
Parameters `config_file` – A path to file which should be used to retrieve configuration from
(might also be a list of file paths)

`last_change`
Returns last change made on this repository as datetime object

`pull` *(url)*
Tries to pull changes from external location.

`remove_tag` *(name, user, message=None, date=None)*
Removes tag with the given `name`.

Parameters

- `name` – name of the tag to be removed
- `user` – full username, i.e.: “Joe Doe <joe.doe@example.com>”
- `message` – message of the tag’s removal commit
- `date` – date of tag’s removal commit

Raises `TagDoesNotExistError` if tag with given name does not exists

`revisions`
Returns list of revisions’ ids, in ascending order. Being lazy attribute allows external tools to inject shas
from cache.

`tag` *(name, user, revision=None, message=None, date=None, **kwargs)*
Creates and returns a tag for the given `revision`.

Parameters

- `name` – name for new tag
- `user` – full username, i.e.: “Joe Doe <joe.doe@example.com>”
- `revision` – changeset id for which new tag would be created
- `message` – message of the tag’s commit
- `date` – date of tag’s commit

Raises `TagAlreadyExistError` if tag with same name already exists

`tags`
Get’s tags for this repository

`workdir`
Returns `Workdir` instance for this repository.

MercurialChangeset
class `MercurialChangeset` *(repository, revision)*
Bases: `BaseChangeset`

 Represents state of the repository at the single revision.

`id`
Returns shorter version of mercurial’s changeset hexes.

`raw_id`
Returns raw string identifying this changeset.

Returns raw string identifying this changeset (40-length hex)
short_id
   Returns shortened version of raw_id (first 12 characters)

revision
   Returns integer identifying this changeset.
   Returns integer representing changeset.

parents
   Returns list of parents changesets.
   Returns list of parents changesets.

added
   Returns list of added FileNode objects.
   Returns list of added FileNode objects.

changed
   Returns list of modified FileNode objects.
   Returns list of changed FileNode objects.

removed
   Returns list of removed FileNode objects.
   Returns list of removed RemovedFileNode objects.

---

Note: Remember that those RemovedFileNode instances are only dummy FileNode objects and trying to access most of its attributes or methods would raise NodeError exception.

added
   Returns list of added FileNode objects.

affected_files
   Get's a fast accessible file changes for given changeset

as_dict()
   Returns dictionary with changeset’s attributes and their values.

author

author_email
   Returns Author email address for given commit

author_name
   Returns Author name for given commit

bookmarks

branch

changed
   Returns list of modified FileNode objects.

children
   Returns list of children changesets.

committer

committer_email
   Returns Author email address for given commit
committer_name
   Returns Author name for given commit

date

diff (ignore_whitespace=True, context=3)

fill_archive (stream=None, kind=’tgz’, prefix=None, subrepos=False)
   Fills up given stream.

   Parameters
   • stream – file like object.
   • kind – one of following: zip, tgz or tbz2. Default: tgz.
   • prefix – name of root directory in archive. Default is repository name and changeset’s raw_id joined with dash (repo-tip.<KIND>.
   • subrepos – include subrepos in this archive.

   Raises
   • ImproperArchiveTypeError – If given kind is wrong.
   • VcsError – If given stream is None

get_chunked_archive (**kwargs)
   Returns iterable archive. Tiny wrapper around fill_archive method.

   Parameters chunk_size – extra parameter which controls size of returned chunks. Default:8k.

get_fileannotate (path)
   Returns a generator of four element tuples with lineno, sha, changeset lazy loader and line

get_filechangeset (path)
   Returns last commit of the file at the given path.

get_filecontent (path)
   Returns content of the file at given path.

get_fillhistory (path, limit=None)
   Returns history of file as reversed list of Changeset objects for which file at given path has been modified.

get_filemode (path)
   Returns stat mode of the file at the given path.

get_filesize (path)
   Returns size of the file at given path.

get_filenodes_generator ()
   Returns generator that yields all file nodes.

get_node (path)
   Returns Node object from the given path. If there is no node at the given path, ChangesetError would be raised.

get_nodes (path)
   Returns combined DirNode and FileNode objects list representing state of changeset at the given path. If node at the given path is not instance of DirNode, ChangesetError would be raised.

id

last
message

def next(branch=None):
    Returns list of parents changesets.

def parents:
    Returns list of parents changesets.

def prev(branch=None):
    Returns list of removed FileNode objects.

def revision:
    Returns integer identifying this changeset.

def root:
    Returns RootNode object for this changeset.

def short_id:
    Returns raw string identifying this changeset.

def size:
    Returns total number of bytes from contents of all filenodes.

def status:
    Returns modified, added, removed, deleted files for current changeset.

def tags:

    def walk(topurl=''):  # Similar to os.walk method. Instead of filesystem it walks through changeset starting at given topurl. Returns generator of tuples (topnode, dirnodes, filenodes).

class MercurialInMemoryChangeset(vcs.backends.hg.MercurialInMemoryChangeset):
    def __init__(repository):
        super().__init__(repository)

    def add(*filenodes):
        Marks given FileNode objects as to be committed.

        Raises
        • NodeAlreadyExistsError – if node with same path exists at latest changeset
        • NodeAlreadyAddedError – if node with same path is already marked as added

    def change(*filenodes):
        Marks given FileNode objects to be changed in next commit.

        Raises
        • EmptyRepositoryError – if there are no changesets yet
        • NodeAlreadyExistsError – if node with same path is already marked to be changed
        • NodeAlreadyRemovedError – if node with same path is already marked to be removed
        • NodeDoesNotExistError – if node doesn’t exist in latest changeset
        • NodeNotChangedError – if node hasn’t really be changed

    def check_integrity(parents=None):
        Checks in-memory changeset’s integrity. Also, sets parents if not already set.
Raises CommitError if any error occurs (i.e. NodeDoesNotExistError).

```python
commit(message, author, parents=None, branch=None, date=None, **kwargs)
```

Performs in-memory commit (doesn’t check workdir in any way) and returns newly created Changeset. Updates repository’s revisions.

**Parameters**

- **message** – message of the commit
- **author** – full username, i.e. “Joe Doe <joe.doe@example.com>”
- **parents** – single parent or sequence of parents from which commit would be derived
- **date** – `datetime.datetime` instance. Defaults to `datetime.datetime.now()`.
- **branch** – branch name, as string. If none given, default backend’s branch would be used.

Raises CommitError if any error occurs while committing

```python
get_ipaths()
```

Returns generator of paths from nodes marked as added, changed or removed.

```python
get_paths()
```

Returns list of paths from nodes marked as added, changed or removed.

```python
remove(*filenodes)
```

Marks given FileNode (or RemovedFileNode) objects to be removed in next commit.

**Raises**

- **NodeAlreadyRemovedError** – if node has been already marked to be removed
- **NodeAlreadyChangedError** – if node has been already marked to be changed

```python
reset()
```

Resets this instance to initial state (cleans added, changed and removed lists).

### Base Backend

class vcs.backends.base.BaseChangeset

Each backend should implement it’s changeset representation.

**Attributes**

- **repository** repository object within which changeset exists
- **id** may be raw_id or i.e. for mercurial’s tip just tip
- **raw_id** raw changeset representation (i.e. full 40 length sha for git backend)
- **short_id** shortened (if apply) version of raw_id; it would be simple shortcut for raw_id[:12] for git/mercurial backends or same as raw_id for subversion
- **revision** revision number as integer
- **files** list of FileNode (Node with NodeKind.FILE) objects
- **dirs** list of DirNode (Node with NodeKind.DIR) objects
- **nodes** combined list of Node objects
- **author** author of the changeset, as unicode
- **message** message of the changeset, as unicode
- **parents** list of parent changesets
last True if this is last changeset in repository, False otherwise; trying to access this attribute while there is no changesets would raise EmptyRepositoryError

added
Returns list of added FileNode objects.

as_dict()
Returns dictionary with changeset’s attributes and their values.

author
Returns Author for given commit

author_email
Returns Author email address for given commit

author_name
Returns Author name for given commit

changed
Returns list of modified FileNode objects.

children
Returns list of children changesets.

committer
Returns Committer for given commit

committer_email
Returns Author email address for given commit

committer_name
Returns Author name for given commit

fill_archive (stream=None, kind='tgz', prefix=None)
Fills up given stream.

Parameters
- stream – file like object.
- kind – one of following: zip, tar, tgz or tbz2. Default: tgz.
- prefix – name of root directory in archive. Default is repository name and changeset’s raw_id joined with dash.
  repo-tip.<kind>

get_chunked_archive (**kwargs)
Returns iterable archive. Tiny wrapper around fill_archive method.

Parameters
- chunk_size – extra parameter which controls size of returned chunks. Default:8k.

get_file_changeset (path)
Returns last commit of the file at the given path.

get_file_content (path)
Returns content of the file at the given path.

get_file_history (path)
Returns history of file as reversed list of Changeset objects for which file at given path has been modified.

get_file_mode (path)
Returns stat mode of the file at the given path.
get_file_size(path)
Returns size of the file at the given path.

getfilenodes_generator()
Returns generator that yields all file nodes.

get_node(path)
Returns Node object from the given path.

Raises NodeDoesNotExistError if there is no node at the given path.

get_nodes(path)
Returns combined DirNode and FileNode objects list representing state of changeset at the given path.

Raises ChangesetError if node at the given path is not instance of DirNode.

id
Returns string identifying this changeset.

next(branch=None)
Returns next changeset from current, if branch is given it will return next changeset belonging to this branch.

Parameters branch – show changesets within the given named branch

parents
Returns list of parents changesets.

prev(branch=None)
Returns previous changeset from current, if branch is given it will return previous changeset belonging to this branch.

Parameters branch – show changesets within the given named branch

raw_id
Returns raw string identifying this changeset.

removed
Returns list of removed FileNode objects.

revision
Returns integer identifying this changeset.

root
Returns RootNode object for this changeset.

short_id
Returns shortened version of raw_id attribute, as string, identifying this changeset, useful for web representation.

size
Returns total number of bytes from contents of all filenodes.

walk(topurl='')
Similar to os.walk method. Instead of filesystem it walks through changeset starting at given topurl. Returns generator of tuples (topnode, dirnodes, filenodes).

class vcs.backends.base.BaseInMemoryChangeset(repository)
Represents differences between repository's state (most recent head) and changes made in place.

Attributes

repository repository object for this in-memory-changeset
added  list of FileNode objects marked as added
changed  list of FileNode objects marked as changed
removed  list of FileNode or RemovedFileNode objects marked to be removed
parents  list of Changeset representing parents of in-memory changeset. Should always be 2-element sequence.

add(*filenodes)
Marks given FileNode objects as to be committed.

Raises
• NodeAlreadyExistsError – if node with same path exists at latest changeset
• NodeAlreadyAddedError – if node with same path is already marked as added

change(*filenodes)
Marks given FileNode objects to be changed in next commit.

Raises
• EmptyRepositoryError – if there are no changesets yet
• NodeAlreadyExistsError – if node with same path is already marked to be changed
• NodeAlreadyRemovedError – if node with same path is already marked to be removed
• NodeDoesNotExistError – if node doesn’t exist in latest changeset
• NodeNotChangedError – if node hasn’t really be changed

check_integrity(parents=None)
Checks in-memory changeset’s integrity. Also, sets parents if not already set.

Raises CommitError if any error occurs (i.e. NodeDoesNotExistError).

commit(message, author=None, parents=None, branch=None, date=None, **kwargs)
Performs in-memory commit (doesn’t check workdir in any way) and returns newly created Changeset. Updates repository’s revisions.

Note: While overriding this method each backend’s should call self.check_integrity(parents) in the first place.

Parameters
• message – message of the commit
• author – full username, i.e. “Joe Doe <joe.doe@example.com>”
• parents – single parent or sequence of parents from which commit would be derivered
• date – datetime.datetime instance. Defaults to datetime.datetime.now().
• branch – branch name, as string. If none given, default backend’s branch would be used.

Raises CommitError if any error occurs while committing

get_ipaths()
Returns generator of paths from nodes marked as added, changed or removed.

get_paths()
Returns list of paths from nodes marked as added, changed or removed.
remove (*filenodes)
Marks given FileNode (or RemovedFileNode) objects to be removed in next commit.

Raises

- NodeAlreadyRemovedError – if node has been already marked to be removed
- NodeAlreadyChangedError – if node has been already marked to be changed

reset()
Resets this instance to initial state (cleans added, changed and removed lists).

class vcs.backends.base.BaseRepository (repo_path, create=False, **kwargs)
Base Repository for final backends

Attributes

- DEFAULT_BRANCH_NAME name of default branch (i.e. “trunk” for svn, “master” for git etc.
- scm alias of scm, i.e. git or hg
- repo object from external api
- revisions list of all available revisions’ ids, in ascending order
- changesets storage dict caching returned changesets
- path absolute path to the repository
- branches branches as list of changesets
- tags tags as list of changesets

Initializes repository. Raises RepositoryError if repository could not be find at the given repo_path or directory at repo_path exists and create is set to True.

Parameters

- repo_path – local path of the repository
- create=False – if set to True, would try to create repository.
- src_url=None – if set, should be proper url from which repository would be cloned; requires create parameter to be set to True - raises RepositoryError if src_url is set and create evaluates to False

add (filenode, **kwargs)
Commit api function that will add given FileNode into this repository.

Raises

- NodeAlreadyExistsError – if there is a file with same path already in repository
- NodeAlreadyAddedError – if given node is already marked as added

commit (message, **kwargs)
Persists current changes made on this repository and returns newly created changeset.

Raises NothingChangedError if no changes has been made

get_changeset (revision=None)
Returns instance of Changeset class. If revision is None, most recent changeset is returned.

Raises EmptyRepositoryError if there are no revisions
get_changesets

Returns iterator of MercurialChangeset objects from start to end not inclusive. This should behave just like a list, i.e. end is not inclusive.

Parameters

- start – None or str
- end – None or str
- start_date –
- end_date –
- branch_name –
- reverse –

get_config_value

Returns configuration value for a given [section] and name.

Parameters

- section – Section we want to retrieve value from
- name – Name of configuration we want to retrieve
- config_file – A path to file which should be used to retrieve configuration from (might also be a list of file paths)

get_diff

Returns (git like) diff, as plain text. Shows changes introduced by rev2 since rev1.

Parameters

- rev1 – Entry point from which diff is shown. Can be self.EMPTY_CHANGESET - in this case, patch showing all the changes since empty state of the repository until rev2
- rev2 – Until which revision changes should be shown.
- ignore_whitespace – If set to True, would not show whitespace changes. Defaults to False.
- context – How many lines before/after changed lines should be shown. Defaults to 3.

get_state

Returns dictionary with added, changed and removed lists containing FileNode objects.

get_user_email

Returns user’s email from global configuration file.

Parameters config_file – A path to file which should be used to retrieve configuration from (might also be a list of file paths)

get_user_name

Returns user’s name from global configuration file.

Parameters config_file – A path to file which should be used to retrieve configuration from (might also be a list of file paths)

in_memory_changeset

Returns InMemoryChangeset object for this repository.

is_valid

Validates repository.
remove (filenode, **kwargs)
Commit api function that will remove given FileNode into this repository.

Raises
- EmptyRepositoryError – if there are no changesets yet
- NodeDoesNotExistError – if there is no file with given path

remove_tag (name, user, message=None, date=None)
Removes tag with the given name.

Parameters
- name – name of the tag to be removed
- user – full username, i.e.: "Joe Doe <joe.doe@example.com>"
- message – message of the tag’s removal commit
- date – date of tag’s removal commit

Raises TagDoesNotExistError if tag with given name does not exists

size
Returns combined size in bytes for all repository files

tag (name, user, revision=None, message=None, date=None, **opts)
Creates and returns a tag for the given revision.

Parameters
- name – name for new tag
- user – full username, i.e.: “Joe Doe <joe.doe@example.com>”
- revision – changeset id for which new tag would be created
- message – message of the tag’s commit
- date – date of tag’s commit

Raises TagAlreadyExistError if tag with same name already exists

workdir
Returns Workdir instance for this repository.

class vcs.backends.base.BaseWorkdir (repository)
Working directory representation of single repository.

Attribute repository: repository object of working directory

checkout_branch (branch=None)
Checks out branch or the backend’s default branch.

 Raises BranchDoesNotExistError if the branch does not exist.

commit (message, **kwargs)
Commits local (from working directory) changes and returns newly created Changeset. Updates repository’s revisions list.

 Raises CommitError if any error occurs while committing

get_added ()
Returns list of FileNode objects marked as new in working directory.

get_branch ()
Returns name of current branch.
get_changed()
Returns list of FileNode objects changed in working directory.

get_changeset()
Returns current changeset.

get_removed()
Returns list of RemovedFileNode objects marked as removed in working directory.

get_status()
Returns dict with added, changed, removed and untracked lists.

get_untracked()
Returns list of FileNode objects which are present within working directory however are not tracked by repository.

update(revision=None)
Fetches content of the given revision and populates it within working directory.

class vcs.backends.base.EmptyChangeset(cs='0000000000000000000000000000000000000000',
repo=None, requested_revision=None, alias=None,
revision=-1, message='', author='', date=None)
An dummy empty changeset. It's possible to pass hash when creating an EmptyChangeset

raw_id
Returns raw string identifying this changeset, useful for web representation.

1.6.5 vcs.utils

This module provides some useful tools for vcs like annotate/diff html output. It also includes some internal helpers.

Public API

• Annotate utils
• Diffs utils
• Helpers

Private API

• Lazy attributes utils

Annotate utils

vcs.utils.annotate.annotate_highlight(filenode, annotate_from_changeset_func=None, order=None, headers=None, **options)
Returns html portion containing annotated table with 3 columns: line numbers, changeset information and pygmentized line of code.

Parameters

• filenode – FileNode object
• annotate_from_changeset_func – function taking changeset and returning single annotate cell; needs break line at the end
• order – ordered sequence of ls (line numbers column), annotate (annotate column), code (code column); Default is ['ls', 'annotate', 'code']
• headers – dictionary with headers (keys are whats in order parameter)
Diffs utils

class vcs.utils.diffs.DiffProcessor (diff, differ='diff', format='udiff')

Give it a unified diff and it returns a list of the files that were mentioned in the diff together with a dict of meta information that can be used to render it in a HTML template.

Parameters

- **diff** – a text in diff format or generator
- **format** – format of diff passed, udiff or gitdiff

as_html (table_class='code-difftable', line_class='line', new_lineno_class='lineno old',
old_lineno_class='lineno new', code_class='code')

Return udiff as html table with customized css classes

copy_iterator ()

make a fresh copy of generator, we should not iterate thru an original as it’s needed for repeating operations on this instance of DiffProcessor

prepare ()

Prepare the passed udiff for HTML rendering. It’ll return a list of dicts

raw_diff ()

Returns raw string as udiff

stat ()

Returns tuple of added and removed lines for this instance

vcs.utils.diffs.get_gitdiff (filenode_old, filenode_new, ignore_whitespace=True)

Returns git style diff between given filenode_old and filenode_new.

Parameters ignore_whitespace – ignore whitespaces in diff

vcs.utils.diffs.get_udiff (filenode_old, filenode_new, show_whitespace=True)

Returns unified diff between given filenode_old and filenode_new.

Helpers

Utilities aimed to help achieve mostly basic tasks.

vcs.utils.helpers.get_dict_for_attrs (obj, attrs)

Returns dictionary for each attribute from given obj.

vcs.utils.helpers.get_highlighted_code (name, code, type='terminal')

If pygments are available on the system then returned output is colored. Otherwise unchanged content is returned.

vcs.utils.helpers.get_repo_paths (path)

Returns path’s subdirectories which seems to be a repository.

vcs.utils.helpers.get_scm (path, search_up=False, explicit_alias=None)

Returns one of alias from ALIASES (in order of precedence same as shortcuts given in ALIASES) and top working dir path for the given argument. If no scm-specific directory is found or more than one scm is found at that directory, VCSError is raised.

Parameters

- **search_up** – if set to True, this function would try to move up to parent directory every time no scm is recognized for the currently checked path. Default: False.
• **explicit_alias** – can be one of available backend aliases, when given it will return given explicit alias in repositories under more than one version control, if explicit_alias is different than found it will raise VCSError

```python
vcs.utils.helpers.get_scms_for_path(path)
```

Returns all scm's found at the given path. If no scm is recognized - empty list is returned.

**Parameters** path – path to directory which should be checked. May be callable.

**Raises** VCSError if given path is not a directory

```python
vcs.utils.helpers.get_total_seconds(timedelta)
```

Backported for Python 2.5.

See [http://docs.python.org/library/datetime.html](http://docs.python.org/library/datetime.html).

```python
vcs.utils.helpers.parse_changesets(text)
```

Returns dictionary with start, main and end ids.

**Examples:**
```python
>>> parse_changesets('aaabbb')
{'start': None, 'main': 'aaabbb', 'end': None}
```
```python
>>> parse_changesets('aaabbb..cccddd')
{'start': 'aaabbb', 'main': None, 'end': 'cccddd'}
```

```python
vcs.utils.helpers.parse_datetime(text)
```

Parses given text and returns datetime.datetime instance or raises ValueError.

**Parameters** text – string of desired date/datetime or something more verbose, like yesterday, 2 weeks 3 days, etc.

```python
vcs.utils.helpers.run_command(cmd, *args)
```

 Runs command on the system with given args.

### Lazy attributes utils

```python
vcs.utils.lazy.LazyProperty
```

Decorator for easier creation of property from potentially expensive to calculate attribute of the class.

**Usage:**
```python
class Foo(object):
    @LazyProperty
def bar(self):
        print 'Calculating self._bar'
        return 42
```

Taken from [http://blog.pythonisito.com/2008/08/lazy-descriptors.html](http://blog.pythonisito.com/2008/08/lazy-descriptors.html) and used widely.

```python
vcs.utils.lazy.ThreadLocalLazyProperty
```

Same as above but uses thread local dict for cache storage.

### 1.7 License

Copyright (C) 2010–2013 vcs Marcin Kuźmiński & Łukasz Balcerzak

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
CHAPTER TWO

OTHER TOPICS

• genindex
• search
V
vcs.backends.??
vcs.backends.base.??
vcs.backends.hg.??
vcs.cli.??
vcs.conf.settings.??
vcs.nodes.??
vcs.utils.??
vcs.utils.annotate.??
vcs.utils.diffs.??
vcs.utils.helpers.??
vcs.utils.lazy.??