vcs-repo-mgr

Release 4.2

Apr 26, 2018

Contents

Welcome to the documentation of *vcs-repo-mgr* version 4.2! The following sections are available:

CHAPTER 1

User documentation

The readme is the best place to start reading, it's targeted at all users and documents the command line interface:

1.1 vcs-repo-mgr: Version control repository manager

coverage 93% The Python package *vcs-repo-mgr* provides a command line program and Python API to perform common operations (in the context of packaging/deployment) on version control repositories. It's currently tested on Python 2.6, 2.7, 3.4, 3.5 and 3.6 on Linux and Mac OS X. Bazaar, Mercurial and Git repositories are supported.

1.1.1 Installation

The *vcs-repo-mgr* package is available on PyPI which means installation should be as simple as:

```
$ pip install vcs-repo-mgr
```

There's actually a multitude of ways to install Python packages (e.g. the per user site-packages directory, virtual environments or just installing system wide) and I have no intention of getting into that discussion here, so if this intimidates you then read up on your options before returning to these instructions ;-).

You will also need Bazaar, Mercurial and/or Git installed (depending on the type of repositories you want to work with). Here's how you install them on Debian and Ubuntu based systems:

```
$ sudo apt-get install bzr mercurial git-core
```

1.1.2 Usage

There are two ways to use the *vcs-repo-mgr* package: As the command line program `vcs-tool` and as a Python API. For details about the Python API please refer to the API documentation available on Read the Docs. The command line interface is described below.

Usage: *vcs-tool [OPTIONS] [ARGS]*

Command line program to perform common operations (in the context of packaging/deployment) on version control repositories. Supports Bazaar, Mercurial and Git repositories.

Supported options:

Option	Description
`-r`, `--repository=REPOSITORY`	Select a repository to operate on by providing the name of a repository defined in one of the configuration files ~/.vcs-repo-mgr.ini and /etc/vcs-repo-mgr.ini. Alternatively the location of a remote repository can be given. The location should be prefixed by the type of the repository (with a "+" in between) unless the location ends in ".git" in which case the prefix is optional.
`--rev`, `--revision=REVISION`	Select a revision to operate on. Accepts any string that's supported by the VCS system that manages the repository, which means you can provide branch names, tag names, exact revision ids, etc. This option is used in combination with the `--find-revision-number`, `--find-revision-id` and `--export` options. If this option is not provided a default revision is selected: "last:1" for Bazaar repositories, "master" for git repositories and "default" (not "tip"!) for Mercurial repositories.
`--release=RELEASE_ID`	Select a release to operate on. This option works in the same way as the `--revision` option. Please refer to the vcs-repo-mgr documentation for details on "releases". Although release identifiers are based on branch or tag names they may not correspond literally, this is why the release identifier you specify here is translated to a global revision id before being passed to the VCS system.
`-n`, `--find-revision-number`	Print the local revision number (an integer) of the revision given with the `--revision` option. Revision numbers are useful as a build number or when a simple, incrementing version number is required. Revision numbers should not be used to unambiguously refer to a revision (use revision ids for that instead). This option is used in combination with the `--repository` and `--revision` options.
`-i`, `--find-revision-id`	Print the global revision id (a string) of the revision given with the `--revision` option. Global revision ids are useful to unambiguously refer to a revision. This option is used in combination with the `--repository` and `--revision` options.
`--list-releases`	Print the identifiers of the releases in the repository given with the `--repository` option. The release identifiers are printed on standard output (one per line), ordered using natural order comparison.
`--select-release=RELEASE_ID`	Print the identifier of the newest release that is not newer than `RELEASE_ID` in the repository given with the `--repository` option. The release identifier is printed on standard output.
`-s`, `--sum-revisions`	Print the summed revision numbers of multiple repository/revision pairs. The repository/revision pairs are taken from the positional arguments to vcs-repo-mgr. This is useful when you're building a package based on revisions from multiple VCS repositories. By taking changes in all repositories into account when generating version numbers you can make sure that your version number is bumped with every single change.
`--vcs-control-field`	Print a line containing a Debian control file field and value. The field name will be one of "Vcs-Bzr", "Vcs-Hg" or "Vcs-Git". The value will be the repository's remote location and the selected revision (separated by a "#" character).
`-u`, `--update`	Create/update the local clone of a remote repository by pulling the latest changes from the remote repository. This option is used in combination with the `--repository` option.
`-m`, `--merge-up`	Merge a change into one or more release branches and the default branch. By default merging starts from the current branch. You can explicitly select the branch where merging should start using the `--rev`, `--revision` and `--release` options. You can also start by merging a feature branch into the selected release branch before merging the change up through later release branches and the default branch. To do so you pass the name of the feature branch as a positional argument. If the feature branch is located in a different repository you can prefix the location of the repository to the name of the feature branch with a "#" token in

The primary way to use the `vcs-tool` command requires you to create a configuration file:

```
$ cat > ~/.vcs-repo-mgr.ini << EOF
[coloredlogs]
type = git
local = /tmp/coloredlogs
remote = git@github.com:xolox/python-coloredlogs.git
EOF
```

Because the `-r`, `--repository` option accepts remote repository locations in addition to names it's not actually required to create a configuration file. Of course this depends on your use case(s).

Below are some examples of the command line interface. If you're interested in using the Python API please refer to the online documentation.

Updating repositories

If the configuration file defines a local *and* remote repository and the local repository doesn't exist yet it will be created the first time you update it:

```
$ vcs-tool --repository coloredlogs --update
2014-05-04 18:55:54 INFO Creating Git clone of git@github.com:xolox/python-
↪coloredlogs.git at /tmp/coloredlogs ..
Cloning into bare repository '/tmp/coloredlogs'...
remote: Reusing existing pack: 96, done.
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 101 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (101/101), 28.11 KiB, done.
Resolving deltas: 100% (44/44), done.
```

Later runs will pull the latest changes instead of performing a full clone:

```
$ vcs-tool --repository coloredlogs --update
2014-05-04 18:55:56 INFO Updating Git clone of git@github.com:xolox/python-
↪coloredlogs.git at /tmp/coloredlogs ..
From github.com:xolox/python-coloredlogs
 * branch HEAD -> FETCH_HEAD
```

Finding revision numbers/ids

Revision numbers are integer numbers that increment with every added revision. They're very useful during packaging/deployment:

```
$ vcs-tool --repository coloredlogs --revision master --find-revision-number
24
```

Revision ids (hashes) are hexadecimal strings that uniquely identify revisions. They are useful to unambiguously refer to a revision and its history (e.g while building a package you can embed the revision id as a hint about the origins of the package):

```
$ vcs-tool --repository coloredlogs --revision master --find-revision-id
bce75c1eea88ebd40135cd45de716fe9591e348c
```

Exporting revisions

By default the repositories created by *vcs-repo-mgr* do not contain a working tree, just the version control files (in Git terminology this is called a "bare repository"). This has two reasons:

1. Bare repositories help conserve disk space. This is insignificant for small repositories, but on large repositories it can make a noticeable difference. Especially if you're using a lot of them :-)
2. Bare repositories enforce the principle that the working tree shouldn't be used during packaging (instead you should export the tree at a specific revision to a temporary directory and use that). This insistence on not using the working tree during packaging has two reasons:
   (a) The working tree can contain files which are not under version control. Such files should certainly *not* be included in a package unintentionally.
   (b) If the working tree of a repository is used, this makes it impossible to safely perform parallel builds from the same repository (the builds can corrupt each other's working tree).

This means that if you want to do something with the files in the repository you have to export a revision to a (temporary) directory:

```
$ vcs-tool --repository coloredlogs --export /tmp/coloredlogs-snapshot
2014-05-04 19:17:24 INFO Exporting revision master of /tmp/coloredlogs to /tmp/
↪coloredlogs-snapshot ..

$ ls -l /tmp/coloredlogs-snapshot
total 28K
drwxrwxr-x 2 peter peter 4.0K May  3 14:31 coloredlogs
drwxrwxr-x 3 peter peter 4.0K May  3 14:31 vim
-rw-rw-r-- 1 peter peter 1.1K May  3 14:31 LICENSE.txt
-rw-rw-r-- 1 peter peter   56 May  3 14:31 MANIFEST.in
-rw-rw-r-- 1 peter peter 5.4K May  3 14:31 README.rst
-rwxrwxr-x 1 peter peter 1.1K May  3 14:31 setup.py
```

1.1.3 Future improvements

This section is currently a "braindump" which means I haven't committed to any of these improvements, I'm just thinking out loud ;-).

Improve interactive repository selection Two improvements for interactive usage of the `vcs-tool` program:

- Automatically load a repository's configuration when a pathname is given that matches an entry in a configuration file (right now you need to give the repository's name in order to load its configuration).
- Do the obvious thing when no repository is specified on the command line but the working directory matches a configured repository.

Wildcard matching in configuration files It might be interesting to support shell wildcard matching against local directory names to apply a default configuration to a group of repositories?

Enable more extensive customization Right now the version control commands are hard coded and not easy to customize for those cases where the existing API gets you 90% of where you want to be but makes that last 10% impossible. Technically this is already possible through subclassing, but a more lightweight solution would certainly be nice to have :-).

1.1.4 Known issues

This section documents known issues that users may run into.

Problematic dependencies

Bazaar and Mercurial are both written in Python and available on PyPI and as such I included them in the installation requirements of *vcs-repo-mgr*, because I couldn't think of a good reason not to.

Adding support for Python 3 to *vcs-repo-mgr* made things more complicated because Bazaar and Mercurial didn't support Python 3, leading to installation errors. To cope with this problem the Bazaar and Mercurial requirements were made conditional on the Python version:

- On Python 2 the Bazaar and Mercurial packages would be installed together with *vcs-repo-mgr*.
- On Python 3 the user was instead responsible for making sure that Bazaar and Mercurial were installed (for example using system packages).

This works fine because *vcs-repo-mgr* only invokes Bazaar and Mercurial using the command line interfaces so it doesn't matter whether the version control system is using the same version of Python as *vcs-repo-mgr*.

Since then the installation of the Bazaar package has started failing on PyPy, unfortunately this time there is no reliable and backwards compatible way to make the Bazaar dependency optional in wheel distributions due to bugs in setuptools.

When I investigated support for environment markers that match Python implementations (refer to the link above) I decided that instead of writing a setup script full of nasty and fragile hacks I'd rather just drop official (tested) support for PyPy, as silly as the reason for it may be.

1.1.5 Contact

The latest version of *vcs-repo-mgr* is available on PyPI and GitHub. The documentation is hosted on Read the Docs and includes a changelog. For bug reports please create an issue on GitHub. If you have questions, suggestions, etc. feel free to send me an e-mail at peter@peterodding.com.

1.1.6 License

CHAPTER 2

API documentation

The following API documentation is automatically generated from the source code:

2.1 API documentation

The following API documentation was automatically generated from the source code of *vcs-repo-mgr* 4.2:

2.1.1 `vcs_repo_mgr`

Python API for the *vcs-repo-mgr* package.

Note: This module handles subprocess management using `executor`. This means that the `ExternalCommandFailed` exception can be raised at (more or less) any point.

Getting started

When using *vcs-repo-mgr* as a Python API the following top level entities should help you get started:

- The `Repository` class implements most of the functionality exposed by the *vcs-repo-mgr* project. In practice you'll use one of the subclasses which implement support for a specific VCS system (`BzrRepo`, `GitRepo` and `HgRepo`).
  - `Repository` objects construct `Revision` and `Release` objects so you'll most likely be using these.
- The `find_configured_repository()` function constructs instances of `Repository` subclasses based on configuration files. This is useful when you find yourself frequently instantiating the same `Repository` instances and you'd rather refer to a repository name in your code than repeating the complete local and remote locations everywhere in your code (this kind of duplication is bad after all :-).
- You can choose to directly instantiate `BzrRepo`, `GitRepo` and/or `HgRepo` instances or you can use one of the helper functions that instantiate repository objects for you (`coerce_repository()` and `repository_factory()`).

Common operations

The operations supported by Bazaar, Git and Mercurial have confusingly similar names *except when they don't* (don't even get me started about subtly different semantics ;-) and so one challenge while developing *vcs-repo-mgr* has been to come up with good names that adequately capture the semantics of operations (just for the record: I'm not claiming that I always succeed on the first try :-).

In case you find yourself as confused as I have found myself at times, the following table lists common repository operations supported by *vcs-repo-mgr* and their equivalent Bazaar, Git and Mercurial commands:

Python API (*vcs-repo-mgr*)	Bazaar	Git	Mercurial
`Repository.create()`	bzr init/branch	git init/clone	hg init/clone
`Repository.pull()`	bzr pull	git fetch/pull	hg pull
`Repository.push()`	bzr push	git push	hg push
`Repository.checkout()`	(not implemented)	git checkout	hg update
`Repository.commit()`	(not implemented)	git commit	hg commit
`Repository.create_branch()`	(not implemented)	git checkout -b	hg branch
`Repository.merge()`	(not implemented)	git merge –no-commit	hg merge

Note: As you can see from the table above I'm slowly but surely forgetting about keeping Bazaar support up to par, if only because I don't like the "lowest common denominator" approach where useful Git and Mercurial features aren't exposed because there's no clear alternative for Bazaar. Also I work a lot less with Bazaar which means I'm lacking knowledge; keeping Bazaar support up to par at all times drags down my progress significantly.

In contrast while there are of course a lot of small details that differ between Git and Mercurial, I'm still convinced that it's useful to hide these differences, because overall the two systems are so similar that it seems worth it to me (so far :-).

`vcs_repo_mgr.`**`USER_CONFIG_FILE = '~/.vcs-repo-mgr.ini'`**
The location of the user-specific configuration file (a string, parsed using `parse_path()`).

`vcs_repo_mgr.`**`SYSTEM_CONFIG_FILE = '/etc/vcs-repo-mgr.ini'`**
The pathname of the system wide configuration file (a string).

`vcs_repo_mgr.`**`UPDATE_VARIABLE = 'VCS_REPO_MGR_UPDATE_LIMIT'`**
The name of the environment variable that's used to rate limit repository updates (a string).

`vcs_repo_mgr.KNOWN_RELEASE_SCHEMES = ('branches', 'tags')`
The names of valid release schemes (a tuple of strings).

`vcs_repo_mgr.BUNDLED_BACKENDS = ('bzr', 'git', 'hg')`
The names of the version control modules provided by *vcs-repo-mgr* (a tuple of strings).

`vcs_repo_mgr.REPOSITORY_TYPES = set([])`
Available `Repository` subclasses (a `set` of `type` objects).

`vcs_repo_mgr.HEX_PATTERN = <_sre.SRE_Pattern object>`
Compiled regular expression pattern to match hexadecimal strings.

`vcs_repo_mgr.coerce_author(value)`
Coerce strings to `Author` objects.

Parameters **value** – A string or `Author` object.

Returns An `Author` object.

Raises `ValueError` when *value* isn't a string or `Author` object.

`vcs_repo_mgr.coerce_feature_branch(value)`
Convert a string to a `FeatureBranchSpec` object.

Parameters **value** – A string or `FeatureBranchSpec` object.

Returns A `FeatureBranchSpec` object.

`vcs_repo_mgr.coerce_repository(value, context=None)`
Convert a string (taken to be a repository name or location) to a `Repository` object.

Parameters

- **value** – The name or location of a repository (a string) or a `Repository` object.
- **context** – An execution context created by `executor.contexts` (defaults to `executor.contexts.LocalContext`).

Returns A `Repository` object.

Raises `ValueError` when the given value is not a string or a `Repository` object or if the value is a string but doesn't match the name of any configured repository and also can't be parsed as the location of a repository.

The `coerce_repository()` function creates `Repository` objects:

1. If the value is already a `Repository` object it is returned to the caller untouched.
2. If the value is accepted by `find_configured_repository()` then the resulting `Repository` object is returned.
3. If the value is a string that starts with a known VCS type prefix (e.g. `hg+https://bitbucket.org/ianb/virtualenv`) the prefix is removed from the string and a `Repository` object is returned:
   - If the resulting string points to an existing local directory it will be used to set `local`.
   - Otherwise the resulting string is used to set `remote`.
4. If the value is a string pointing to an existing local directory, the VCS type is inferred from the directory's contents and a `Repository` object is returned whose `local` property is set to the local directory.
5. If the value is a string that ends with `.git` (a common idiom for git repositories) a `Repository` object is returned:
   - If the value points to an existing local directory it will be used to set `local`.
   - Otherwise the value is used to set `remote`.

`vcs_repo_mgr.find_cache_directory(remote)`

Find the directory where temporary local checkouts are to be stored.

Returns The absolute pathname of a directory (a string).

`vcs_repo_mgr.find_configured_repository(name)`

Find a version control repository defined by the user in a configuration file.

Parameters **name** – The name of the repository (a string).

Returns A *Repository* object.

Raises *NoSuchRepositoryError* when the given repository name doesn't match any of the configured repositories.

Raises *AmbiguousRepositoryNameError* when the given repository name is ambiguous (i.e. it matches multiple repository names).

Raises *UnknownRepositoryTypeError* when a repository definition with an unknown type is encountered.

The following configuration files are supported:

1. `/etc/vcs-repo-mgr.ini`
2. `~/.vcs-repo-mgr.ini`

Repositories defined in the second file override repositories defined in the first. Here is an example of a repository definition:

```
[vcs-repo-mgr]
type = git
local = ~/projects/vcs-repo-mgr
remote = git@github.com:xolox/python-vcs-repo-mgr.git
bare = true
release-scheme = tags
release-filter = .*
```

Three VCS types are currently supported: `hg` (`mercurial` is also accepted), `git` and `bzr` (`bazaar` is also accepted).

`vcs_repo_mgr.load_backends()`

Load the backend modules bundled with *vcs-repo-mgr*.

Returns The value of *REPOSITORY_TYPES*.

When *REPOSITORY_TYPES* is empty this function will import each of the backend modules listed in *BUNDLED_BACKENDS* before it accesses *REPOSITORY_TYPES*, to make sure that all of the *Repository* subclasses bundled with *vcs-repo-mgr* are registered.

`vcs_repo_mgr.normalize_name(name)`

Normalize a repository name.

Parameters **name** – The name of a repository (a string).

Returns The normalized repository name (a string).

This makes sure that minor variations in character case and/or punctuation don't disrupt the name matching in *find_configured_repository()*.

`vcs_repo_mgr.repository_factory(vcs_type, **kw)`

Instantiate a *Repository* object based on the given type and arguments.

Parameters

- **vcs_type** – One of the strings 'bazaar', 'bzr', 'git', 'hg' or 'mercurial' or a subclass of *Repository*.
- **kw** – The keyword arguments to *Repository.__init__()*.

Returns A *Repository* object.

Raises *UnknownRepositoryTypeError* when the given type is unknown.

vcs_repo_mgr.**sum_revision_numbers**(*arguments*)

Sum revision numbers of multiple repository/revision pairs.

Parameters **arguments** – A list of strings with repository names and revision strings.

Returns A single integer containing the summed revision numbers.

This is useful when you're building a package based on revisions from multiple VCS repositories. By taking changes in all repositories into account when generating version numbers you can make sure that your version number is bumped with every single change.

class vcs_repo_mgr.**limit_vcs_updates**

Avoid duplicate repository updates.

This context manager uses an environment variable to ensure that each configured repository isn't updated more than once by the current process and/or subprocesses.

__enter__()

Set *UPDATE_VARIABLE* to the current time when entering the context.

__exit__(*exc_type=None*, *exc_value=None*, *traceback=None*)

Restore the previous value of *UPDATE_VARIABLE* when leaving the context.

class vcs_repo_mgr.**Author**(***kw*)

An author for commits in version control repositories.

combined

The name and e-mail address of the author combined into one string (a string).

email

The e-mail address of the author (a string).

Note: The *email* property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *email* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

name

The name of the author (a string).

Note: The *name* property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *name* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

class vcs_repo_mgr.**FeatureBranchSpec**(***kw*)

Simple and human friendly feature branch specifications.

expression

The feature branch specification provided by the user (a string).

The value of this property is parsed as follows:

- If *expression* contains two nonempty substrings separated by the character # it is split into two parts where the first part is used to set *location* and the second part is used to set *revision*.
- Otherwise *expression* is interpreted as a revision without a location (in this case *location* will be None).

Some examples to make things more concrete:

```
>>> from vcs_repo_mgr import FeatureBranchSpec
>>> FeatureBranchSpec(expression='https://github.com/xolox/python-vcs-repo-
↪mgr.git#remote-feature-branch')
FeatureBranchSpec(expression='https://github.com/xolox/python-vcs-repo-mgr.git
↪#remote-feature-branch',
                  location='https://github.com/xolox/python-vcs-repo-mgr.git',
                  revision='remote-feature-branch')
>>> FeatureBranchSpec(expression='local-feature-branch')
FeatureBranchSpec(expression='local-feature-branch',
                  location=None,
                  revision='local-feature-branch')
```

Note: The *expression* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *expression* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

location

The location of the repository that contains *revision* (a string or None).

Note: The *location* property is a mutable_property. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use del or delattr().

revision

The name of the feature branch (a string).

Note: The *revision* property is a mutable_property. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use del or delattr().

class vcs_repo_mgr.**RepositoryMeta**(*name*, *bases*, *dict*)

Metaclass for automatic registration of *Repository* subclasses.

__init__(*name*, *bases*, *dict*)

Register a *Repository* subclass as soon as it is defined.

class vcs_repo_mgr.**Repository**(**args*, ***kw*)

Abstract base class for managing version control repositories.

In general you should not use the *Repository* class directly, instead you should use the relevant subclass (*BzrRepo*, *GitRepo* or *HgRepo*).

ALIASES = []

A list of strings with names for the repository type.

The *repository_factory()* function searches the *ALIASES* of all known subclasses of *Repository* in order to map repository specifications like `hg+https://bitbucket.org/ianb/virtualenv` to the correct *Repository* subclass.

repr_properties = ['local', 'remote']
The properties included in the output of `repr()`.

classmethod contains_repository(*context*, *directory*)
Check whether the given directory contains a local repository.

Parameters **directory** – The pathname of a directory (a string).

Returns `True` if the directory contains a local repository, `False` otherwise.

By default *contains_repository()* just checks whether the directory reported by *get_vcs_directory()* exists, but *Repository* subclasses can override this class method to improve detection accuracy.

static get_vcs_directory(*context*, *directory*)
Get the pathname of the directory containing the version control metadata files.

Parameters

- **context** – An execution context created by `executor.contexts`.
- **directory** – The pathname of a directory (a string).

Returns The pathname of the directory containing the version control metadata files (a string). In most cases this will be a subdirectory of the given directory, but it may also be the directory itself.

This static method needs to be implemented by subclasses:

- If *directory* doesn't exist this should not raise exceptions.
- If *directory* does exist its contents may influence the result of *get_vcs_directory()* in order to cope with version control backends whose directory layout changes depending on whether they are *bare* (I'm looking at you git).

author
The author for new commits (an *Author* object or `None`).

When you set this property the new value is coerced using *coerce_author()* (that is to say, strings are automatically converted to an *Author* object).

The default value of this property is computed by *find_author()* (a method that needs to be implemented subclasses).

Note: The *author* property is a `custom_property`. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

---

bare
Whether the local repository should have a working tree or not (a boolean or `None`).

This property specifies whether the local repository should have a working tree or not:

- `True` means the local repository doesn't need and shouldn't have a working tree (in older versions of *vcs-repo-mgr* this was the default and only choice).
- `False` means the local repository does need a working tree (for example because you want to create new commits).

The value of *bare* defaults to auto-detection using *is_bare* for repositories that already exist locally, if only to preserve compatibility with versions of *vcs-repo-mgr* that didn't have working tree support.

For repositories that don't exist locally yet, *bare* defaults to `True` so that *create()* defaults to creating repositories without a working tree.

If *bare* is explicitly set and the local clone already exists it will be checked by *__init__()* to make sure that the values of *bare* and *is_bare* match. If they don't an exception will be raised.

Note: The *bare* property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

branches

A dictionary that maps branch names to *Revision* objects.

Here's an example based on a mirror of the git project's repository:

```
>>> from pprint import pprint
>>> from vcs_repo_mgr.backends.git import GitRepo
>>> repository = GitRepo(remote='https://github.com/git/git.git')
>>> pprint(repository.branches)
{'maint':  Revision(repository=GitRepo(...), branch='maint',  revision_id=
↪'16018ae'),
 'master': Revision(repository=GitRepo(...), branch='master', revision_id=
↪'8440f74'),
 'next':   Revision(repository=GitRepo(...), branch='next',   revision_id=
↪'38e7071'),
 'pu':     Revision(repository=GitRepo(...), branch='pu',     revision_id=
↪'d61c1fa'),
 'todo':   Revision(repository=GitRepo(...), branch='todo',   revision_id=
↪'dea8a2d')}
```

compiled_filter

The result of `re.compile()` on *release_filter*.

If *release_filter* isn't a string then it is assumed to be a compiled regular expression object and returned directly.

Note: The *compiled_filter* property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

context

An execution context created by `executor.contexts`.

Note: The *context* property is a `custom_property`. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

control_field

The name of the Debian control file field for the version control system (a string).

Note: The *control_field* property is a `required_property`. You are required to provide a value

for this property by calling the constructor of the class that defines the property with a keyword argument named *control_field* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

current_branch

The name of the branch that's currently checked out in the working tree (a string or `None`).

This property needs to be implemented by subclasses. It should not raise an exception when the current branch can't be determined.

default_pull_remote

The default remote for pulls (a `Remote` object or `None`).

default_push_remote

The default remote for pushes (a `Remote` object or `None`).

default_revision

The default revision of this version control system (a string).

This property needs to be implemented by subclasses.

Note: The `default_revision` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *default_revision* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

exists

`True` if the local repository exists, `False` otherwise.

friendly_name

A user friendly name for the version control system (a string).

Note: The `friendly_name` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *friendly_name* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

is_bare

`True` if the repository has no working tree, `False` if it does.

This property needs to be implemented by subclasses.

is_clean

`True` if the working tree is clean, `False` otherwise.

This property needs to be implemented by subclasses.

known_remotes

Remote repositories connected to the local repository (a list of `Remote` objects).

This property needs to be implemented by subclasses.

last_updated

The date and time when *vcs-repo-mgr* last checked for updates (an integer).

Used internally by `pull()` when used in combination with `limit_vcs_updates`. The value is a UNIX time stamp (0 for remote repositories that don't have a local clone yet).

last_updated_file
The pathname of the file used to mark the last successful update (a string).

local
The pathname of the local repository (a string).

Note: The *local* property is a `custom_property`. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

---

merge_conflicts
The filenames of any files with merge conflicts (a list of strings).

This property needs to be implemented by subclasses.

ordered_branches
The values in *branches* ordered by branch name (a list of *Revision* objects).

The list is ordered by performing a natural order sort of branch names in ascending order (i.e. the first value is the "oldest" branch and the last value is the "newest" branch).

ordered_releases
The values in *releases* ordered by release identifier (a list of *Release* objects).

The list is ordered by performing a natural order sort of release identifiers in ascending order (i.e. the first value is the "oldest" release and the last value is the "newest" release).

ordered_tags
The values in *tags* ordered by tag name (a list of *Revision* objects).

The list is ordered by performing a natural order sort of tag names in ascending order (i.e. the first value is the "oldest" tag and the last value is the "newest" tag).

release_branches
A dictionary that maps branch names to *Release* objects.

release_filter
The repository's release filter (a string or regular expression, defaults to `.*`).

The value of *release_filter* should be a string containing a regular expression or the result of `re.compile()`. The regular expression is used by *Repository.releases* to match tags or branches that signify "releases". If the regular expression contains a single capture group, the identifier of a *Release* object is set to the substring captured by the capture group (instead of the complete tag or branch name). This defaults to the regular expression `.*` which matches any branch or tag name.

Note: The *release_filter* property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

---

release_scheme
The repository's release scheme (a string, defaults to 'tags').

The value of *release_scheme* determines whether *Repository.releases* is based on *Repository.tags* or *Repository.branches*. It should match one of the values in *KNOWN_RELEASE_SCHEMES*. If an invalid value is set `ValueError` will be raised.

Note: The *release_scheme* property is a `mutable_property`. You can change the value of this

property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

releases

A dictionary that maps release identifiers to *`Release`* objects.

Here's an example based on a mirror of the git project's repository which shows the last ten releases based on tags, where each release identifier captures a tag without its 'v' prefix:

```
>>> from pprint import pprint
>>> from vcs_repo_mgr.backends.git import GitRepo
>>> repository = GitRepo(remote='https://github.com/git/git.git',
...                      release_scheme='tags',
...                      release_filter=r'^v(\d+(?:\.\d+)*)$')
>>> pprint(repository.ordered_releases[-10:])
[Release(revision=Revision(..., tag='v2.2.2', ...), identifier='2.2.2'),
 Release(revision=Revision(..., tag='v2.3.0', ...), identifier='2.3.0'),
 Release(revision=Revision(..., tag='v2.3.1', ...), identifier='2.3.1'),
 Release(revision=Revision(..., tag='v2.3.2', ...), identifier='2.3.2'),
 Release(revision=Revision(..., tag='v2.3.3', ...), identifier='2.3.3'),
 Release(revision=Revision(..., tag='v2.3.4', ...), identifier='2.3.4'),
 Release(revision=Revision(..., tag='v2.3.5', ...), identifier='2.3.5'),
 Release(revision=Revision(..., tag='v2.3.6', ...), identifier='2.3.6'),
 Release(revision=Revision(..., tag='v2.3.7', ...), identifier='2.3.7'),
 Release(revision=Revision(..., tag='v2.4.0', ...), identifier='2.4.0')]
```

remote

The location of the remote repository (a string or `None`).

Note: The *`remote`* property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

supports_working_tree

`True` if the repository supports a working tree, `False` otherwise.

This property needs to be implemented by subclasses.

tags

A dictionary that maps tag names to *`Revision`* objects.

Here's an example based on a mirror of the git project's repository:

```
>>> from pprint import pprint
>>> from vcs_repo_mgr.backends.git import GitRepo
>>> repository = GitRepo(remote='https://github.com/git/git.git')
>>> pprint(repository.tags)
{'v0.99': Revision(repository=GitRepo(...),
                   tag='v0.99',
                   revision_id='d6602ec5194c87b0fc87103ca4d67251c76f233a'),
 'v0.99.1': Revision(repository=GitRepo(...),
                     tag='v0.99.1',
                     revision_id='f25a265a342aed6041ab0cc484224d9ca54b6f41'),
 'v0.99.2': Revision(repository=GitRepo(...),
                     tag='v0.99.2',
                     revision_id='c5db5456ae3b0873fc659c19fafdde22313cc441'),
 ..., # dozens of tags omitted to keep this example short
```

```
'v2.3.6': Revision(repository=GitRepo(...),
                   tag='v2.3.6',
                   revision_id='8e7304597727126cdc52771a9091d7075a70cc31'),
'v2.3.7': Revision(repository=GitRepo(...),
                   tag='v2.3.7',
                   revision_id='b17db4d9c966de30f5445632411c932150e2ad2f'),
'v2.4.0': Revision(repository=GitRepo(...),
                   tag='v2.4.0',
                   revision_id='67308bd628c6235dbc1bad60c9ad1f2d27d576cc')}
```

vcs_directory
The pathname of the directory containing the version control metadata files (a string).

__init__(**args*, ***kw*)
Initialize a `Repository` object.

Refer to the initializer of the superclass (`PropertyManager`) for details about argument handling.

During initialization `ValueError` can be raised for any of the following reasons:

- Neither `local` nor `remote` is specified.
- The local repository doesn't exist and `remote` isn't specified.
- The local repository already exists but the values of `bare` and `is_bare` don't match.
- The `release_scheme` is invalid.
- The `release_filter` regular expression contains more than one capture group (if you need additional groups but without the capturing aspect use a non-capturing group).

add_files(**filenames*, ***kw*)
Include added and/or removed files in the working tree in the next commit.

Parameters

- **filenames** – The filenames of the files to include in the next commit (zero or more strings). If no arguments are given all untracked files are added.
- **kw** – Keyword arguments are ignored (instead of raising `TypeError`) to enable backwards compatibility with older versions of *vcs-repo-mgr* where the keyword argument *all* was used.

checkout(*revision=None*, *clean=False*)
Update the working tree of the local repository to the specified revision.

Parameters

- **revision** – The revision to check out (a string, defaults to `default_revision`).
- **clean** – `True` to discard changes in the working tree, `False` otherwise.

commit(*message*, *author=None*)
Commit changes to tracked files in the working tree.

Parameters

- **message** – The commit message (a string).
- **author** – Override `author` (refer to `coerce_author()` for details on argument handling).

create()
Create the local repository (if it doesn't already exist).

Returns `True` if the local repository was just created, `False` if it already existed.

What `create()` does depends on the situation:

- When `exists` is `True` nothing is done.
- When the `local` repository doesn't exist but a `remote` repository location is given, a clone of the remote repository is created.
- When the `local` repository doesn't exist and no `remote` repository has been specified then a new local repository will be created.

When `create()` is responsible for creating the `local` repository it will make sure the `bare` option is respected.

create_branch(*branch_name*)

Create a new branch based on the working tree's revision.

Parameters **branch_name** – The name of the branch to create (a string).

This method automatically checks out the new branch, but note that the new branch may not actually exist until a commit has been made on the branch.

create_release_branch(*branch_name*)

Create a new release branch.

Parameters **branch_name** – The name of the release branch to create (a string).

Raises The following exceptions can be raised:

- `TypeError` when `release_scheme` isn't set to 'branches'.
- `ValueError` when the branch name doesn't match the configured `release_filter` or no parent release branches are available.

This method automatically checks out the new release branch, but note that the new branch may not actually exist until a commit has been made on the branch.

create_tag(*tag_name*)

Create a new tag based on the working tree's revision.

Parameters **tag_name** – The name of the tag to create (a string).

delete_branch(*branch_name*, *message=None*, *author=None*)

Delete or close a branch in the local repository.

Parameters

- **branch_name** – The name of the branch to delete or close (a string).
- **message** – The message to use when closing the branch requires a commit (a string or `None`, defaults to the string "Closing branch NAME").
- **author** – Override `author` (refer to `coerce_author()` for details on argument handling).

ensure_clean()

Make sure the working tree is clean (contains no changes to tracked files).

Raises `WorkingTreeNotCleanError` when the working tree contains changes to tracked files.

ensure_exists()

Make sure the local repository exists.

Raises `ValueError` when the local repository doesn't exist yet.

ensure_hexadecimal_string(*value*, *command=None*)
Make sure the given value is a hexadecimal string.

Parameters

- **value** – The value to check (a string).
- **command** – The command that produced the value (a string or `None`).

Returns The validated hexadecimal string.

Raises `ValueError` when *value* is not a hexadecimal string.

ensure_release_scheme(*expected_scheme*)
Make sure the release scheme is correctly configured.

Parameters **expected_scheme** – The expected release scheme (a string).

Raises `TypeError` when *`release_scheme`* doesn't match the expected release scheme.

ensure_working_tree()
Make sure the local repository has working tree support.

Raises *`MissingWorkingTreeError`* when the local repository doesn't support a working tree.

export(*directory*, *revision=None*)
Export the complete tree from the local version control repository.

Parameters

- **directory** – The directory where the tree should be exported (a string).
- **revision** – The revision to export (a string or `None`, defaults to *`default_revision`*).

find_author()
Get the author information from the version control system.

Returns An *`Author`* object or `None`.

This method needs to be implemented by subclasses. It is expected to get the author information from the version control system (if available).

find_branches()
Find information about the branches in the repository.

Returns A generator of *`Revision`* objects.

This method needs to be implemented by subclasses.

find_tags()
Find information about the tags in the repository.

Returns A generator of *`Revision`* objects.

This method needs to be implemented by subclasses.

find_remote(*default=False*, *name=None*, *role=None*)
Find a remote repository connected to the local repository.

Parameters

- **default** – `True` to only look for default remotes, `False` otherwise.
- **name** – The name of the remote to look for (a string or `None`).
- **role** – A role that the remote should have (a string or `None`).

Returns A `Remote` object or `None`.

find_revision_id(*revision=None*)

Find the global revision id of the given revision.

Parameters **revision** – A reference to a revision, most likely the name of a branch (a string, defaults to `default_revision`).

Returns The global revision id (a hexadecimal string).

This method needs to be implemented by subclasses.

find_revision_number(*revision=None*)

Find the local revision number of the given revision.

Parameters **revision** – A reference to a revision, most likely the name of a branch (a string, defaults to `default_revision`).

Returns The local revision number (an integer).

This method needs to be implemented by subclasses:

- With each commit that is added to the repository, the local revision number needs to increase.
- Whether revision numbers start counting from zero or one is left to the version control system. To make things more concrete: While Bazaar and git count from one, Mercurial counts from zero.

generate_control_field(*revision=None*)

Generate a Debian control file field referring for this repository and revision.

Parameters **revision** – A reference to a revision, most likely the name of a branch (a string, defaults to `default_revision`).

Returns A tuple with two strings: The name of the field and the value.

This generates a *Vcs-Bzr* field for Bazaar repositories, a *Vcs-Git* field for Git repositories and a *Vcs-Hg* field for Mercurial repositories. Here's an example based on the public git repository of the *vcs-repo-mgr* project:

```
>>> from vcs_repo_mgr import coerce_repository
>>> repository = coerce_repository('https://github.com/xolox/python-vcs-repo-
↪mgr.git')
>>> repository.generate_control_field()
('Vcs-Git', 'https://github.com/xolox/python-vcs-repo-mgr.git
↪#b617731b6c0ca746665f597d2f24b8814b137ebc')
```

get_add_files_command(**filenames*)

Get the command to include added and/or removed files in the working tree in the next commit.

Parameters **filenames** – The filenames of the files to include in the next commit (zero or more strings). If no arguments are given all untracked files are added.

Returns A list of strings.

This method needs to be implemented by subclasses.

get_checkout_command(*revision, clean=False*)

Get the command to update the working tree of the local repository.

Parameters

- **revision** – The revision to check out (a string, defaults to `default_revision`).
- **clean** – `True` to discard changes in the working tree, `False` otherwise.

This method needs to be implemented by subclasses.

get_commit_command(*message*, *author=None*)
Get the command to commit changes to tracked files in the working tree.

Parameters

- **message** – The commit message (a string).
- **author** – An `Author` object or `None`.

Returns A list of strings.

This method needs to be implemented by subclasses.

get_create_command()
Get the command to create the local repository.

Returns A list of strings.

This method needs to be implemented by subclasses:

- When `remote` is set the command is expected to create a local repository based on the remote repository.
- When `remote` isn't set the command is expected to create an empty local repository.
- In either case `bare` should be respected.

get_create_branch_command(*branch_name*)
Get the command to create a new branch based on the working tree's revision.

Parameters **branch_name** – The name of the branch to create (a string).

Returns A list of strings.

This method needs to be implemented by subclasses.

get_create_tag_command(*tag_name*)
Get the command to create a new tag based on the working tree's revision.

Parameters **tag_name** – The name of the tag to create (a string).

Returns A list of strings.

get_delete_branch_command(*branch_name*, *message=None*, *author=None*)
Get the command to delete or close a branch in the local repository.

Parameters

- **branch_name** – The name of the branch to create (a string).
- **message** – The message to use when closing the branch requires a commit (a string, defaults to the string "Closing branch NAME").
- **author** – Override `author` (refer to `coerce_author()` for details on argument handling).

Returns A list of strings.

This method needs to be implemented by subclasses.

get_export_command(*directory*, *revision*)
Get the command to export the complete tree from the local repository.

Parameters

- **directory** – The directory where the tree should be exported (a string).
- **revision** – The revision to export (a string, defaults to `default_revision`).

This method needs to be implemented by subclasses.

get_merge_command(*revision*)

Get the command to merge a revision into the current branch (without committing the result).

Parameters **revision** – The revision to merge in (a string, defaults to `default_revision`).

This method needs to be implemented by subclasses.

get_pull_command(*remote=None*, *revision=None*)

Get the command to pull changes from a remote repository into the local repository.

Parameters

- **remote** – The location of a remote repository (a string or `None`).
- **revision** – A specific revision to pull (a string or `None`).

Returns A list of strings.

This method needs to be implemented by subclasses.

get_push_command(*remote=None*, *revision=None*)

Get the command to push changes from the local repository to a remote repository.

Parameters

- **remote** – The location of a remote repository (a string or `None`).
- **revision** – A specific revision to push (a string or `None`).

Returns A list of strings.

This method needs to be implemented by subclasses.

interactive_merge_conflict_handler(*exception*)

Give the operator a chance to interactively resolve merge conflicts.

Parameters **exception** – An `ExternalCommandFailed` object.

Returns `True` if the operator has interactively resolved any merge conflicts (and as such the merge error doesn't need to be propagated), `False` otherwise.

This method checks whether `sys.stdin` is connected to a terminal to decide whether interaction with an operator is possible. If it is then an interactive terminal prompt is used to ask the operator to resolve the merge conflict(s). If the operator confirms the prompt, the merge error is swallowed instead of propagated. When `sys.stdin` is not connected to a terminal or the operator denies the prompt the merge error is propagated.

is_feature_branch(*branch_name*)

Try to determine whether a branch name refers to a feature branch.

Parameters **branch_name** – The name of a branch (a string).

Returns `True` if the branch name appears to refer to a feature branch, `False` otherwise.

This method is used by `merge_up()` to determine whether the feature branch that was merged should be deleted or closed.

If the branch name matches `default_revision` or one of the branch names of the `releases` then it is not considered a feature branch, which means it won't be closed.

mark_updated()

Mark a successful update so that `last_updated` can report it.

merge(*revision=None*)

Merge a revision into the current branch (without committing the result).

Parameters **revision** – The revision to merge in (a string or `None`, defaults to `default_revision`).

Raises The following exceptions can be raised:

- `MergeConflictError` if the merge command reports an error and merge conflicts are detected that can't be (or haven't been) resolved interactively.
- `ExternalCommandFailed` if the merge command reports an error but no merge conflicts are detected.

Refer to the documentation of `merge_conflict_handler` if you want to customize the handling of merge conflicts.

merge_conflict_handler

The merge conflict handler (a callable, defaults to `interactive_merge_conflict_handler()`).

Note: The `merge_conflict_handler` property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

---

merge_up(*target_branch=None*, *feature_branch=None*, *delete=True*, *create=True*)

Merge a change into one or more release branches and the default branch.

Parameters

- **target_branch** – The name of the release branch where merging of the feature branch starts (a string or `None`, defaults to `current_branch`).
- **feature_branch** – The feature branch to merge in (any value accepted by `coerce_feature_branch()`).
- **delete** – `True` (the default) to delete or close the feature branch after it is merged, `False` otherwise.
- **create** – `True` to automatically create the target branch when it doesn't exist yet, `False` otherwise.

Returns If *feature_branch* is given the global revision id of the feature branch is returned, otherwise the global revision id of the target branch (before any merges performed by `merge_up()`) is returned. If the target branch is created by `merge_up()` and *feature_branch* isn't given then `None` is returned.

Raises The following exceptions can be raised:

- `TypeError` when *target_branch* and `current_branch` are both `None`.
- `ValueError` when the given target branch doesn't exist (based on `branches`) and *create* is `False`.
- `ExternalCommandFailed` if a command fails.

pull(*remote=None*, *revision=None*)

Pull changes from a remote repository into the local repository.

Parameters

- **remote** – The location of a remote repository (a string or `None`).
- **revision** – A specific revision to pull (a string or `None`).

If used in combination with `limit_vcs_updates` this won't perform redundant updates.

push(*remote=None*, *revision=None*)
Push changes from the local repository to a remote repository.

Parameters

- **remote** – The location of a remote repository (a string or `None`).
- **revision** – A specific revision to push (a string or `None`).

> **Warning:** Depending on the version control backend the push command may fail when there are no changes to push. No attempt has been made to make this behavior consistent between implementations (although the thought has crossed my mind and I'll likely revisit this in the future).

release_to_branch(*release_id*)
Shortcut to translate a release identifier to a branch name.

Parameters **release_id** – A `Release.identifier` value (a string).

Returns A branch name (a string).

Raises `TypeError` when `release_scheme` isn't 'branches'.

release_to_tag(*release_id*)
Shortcut to translate a release identifier to a tag name.

Parameters **release_id** – A `Release.identifier` value (a string).

Returns A tag name (a string).

Raises `TypeError` when `release_scheme` isn't 'tags'.

select_release(*highest_allowed_release*)
Select the newest release that is not newer than the given release.

Parameters **highest_allowed_release** – The identifier of the release that sets the upper bound for the selection (a string).

Returns The identifier of the selected release (a string).

Raises `NoMatchingReleasesError` when no matching releases are found.

update(*remote=None*)
Alias for `pull()` to enable backwards compatibility.

update_context()
Try to ensure that external commands are executed in the local repository.

What `update_context()` does depends on whether the directory given by `local` exists:

- If `local` exists then the working directory of `context` will be set to `local`. This is to ensure that version control commands are run inside of the intended version control repository.
- If `local` doesn't exist then the working directory of `context` is cleared. This avoids external commands from failing due to an invalid (non existing) working directory.

class vcs_repo_mgr.**Release**(***kw*)
Release objects are revisions that specify a software "release".

Most version control repositories are used to store software projects and most software projects have the concept of "releases": *Specific versions of a software project that have been given a human and machine readable version number (in one form or another).* `Release` objects exist to capture this concept in a form that is

concrete enough to be generally useful while being abstract enough to be used in various ways (because every software project has its own scheme for releases).

By default the `Release` objects created by `Repository.releases` are based on `Repository.tags`, but using `Repository.release_scheme` you can specify that releases should be based on `Repository.branches` instead. Additionally you can use `Repository.release_filter` to specify a regular expression that will be used to distinguish valid releases from other tags/branches.

revision

The revision that the release relates to (a `Revision` object).

> **Note:** The `revision` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *revision* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

identifier

The name of the tag or branch (a string).

If a `Repository.release_filter` containing a single capture group is used this identifier is set to the captured substring instead of the complete tag or branch name.

> **Note:** The `identifier` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *identifier* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

class vcs_repo_mgr.**Remote**(**kw)

A remote repository connected to a local repository.

default

`True` if this is a default remote repository, `False` otherwise.

> **Note:** The `default` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *default* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

location

The location of the remote repository (a string).

> **Note:** The `location` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *location* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

name

The name of the remote repository (a string or `None`).

> **Note:** The `name` property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or

delattr().

repository

The local repository (a *Repository* object).

Note: The *repository* property is a custom_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *repository* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

roles

The roles of the remote repository (a list of of strings).

Currently the roles 'pull' and 'push' are supported.

Note: The *roles* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *roles* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

class vcs_repo_mgr.**Revision**(**kw)

Revision objects represent a specific revision in a *Repository*.

branch

The name of the branch in which the revision exists (a string or None).

When this property is not available its value will be None.

Note: The *branch* property is a mutable_property. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use del or delattr().

repository

The local repository that contains the revision (a *Repository* object).

Note: The *repository* property is a custom_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *repository* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

revision_id

The global revision id of the revision (a string containing a hexadecimal hash).

Global revision ids are comparable between local and remote repositories, which makes them useful to unambiguously refer to a revision and its history.

This property is always available.

Note: The *revision_id* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument

named *revision_id* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

revision_number

The local revision number of the revision (an integer or `None`).

Local revision numbers are integers that increment with each commit. This makes them useful as a build number or when a simple, incrementing version number is required. They should not be used to unambiguously refer to a revision (use `revision_id` for that instead).

When this property is not available its value will be `None`.

Note: The `revision_number` property is a `custom_property`. You can change the value of this property using normal attribute assignment syntax. This property's value is computed once (the first time it is accessed) and the result is cached. To clear the cached value you can use `del` or `delattr()`.

tag

The name of the tag associated to the revision (a string or `None`).

When this property is not available its value will be `None`.

Note: The `tag` property is a `mutable_property`. You can change the value of this property using normal attribute assignment syntax. To reset it to its default (computed) value you can use `del` or `delattr()`.

2.1.2 vcs_repo_mgr.backends

Namespace for the version control backends supported by *vcs-repo-mgr*.

The following backend modules are available:

- `vcs_repo_mgr.backends.bzr`
- `vcs_repo_mgr.backends.git`
- `vcs_repo_mgr.backends.hg`

2.1.3 vcs_repo_mgr.backends.bzr

Support for Bazaar version control repositories.

class vcs_repo_mgr.backends.bzr.**BzrRepo**(**args*, ***kw*)

Manage Bazaar version control repositories.

classmethod contains_repository(*context*, *directory*)

Check whether the given directory contains a local repository.

static get_vcs_directory(*context*, *directory*)

Get the pathname of the directory containing the version control metadata files.

control_field

The name of the Debian control file field for Bazaar repositories (the string 'Vcs-Bzr').

Note: The *control_field* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *control_field* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

default_revision
The default revision for Bazaar repositories (the string 'last:1').

Note: The *default_revision* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *default_revision* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

friendly_name
A user friendly name for the version control system (the string 'Bazaar').

is_bare
True if the repository has no working tree, False if it does.

The value of this property is computed by checking whether the .bzr/checkout directory exists (it doesn't exist in Bazaar repositories created using bzr branch --no-tree ...).

is_clean
True if the working tree is clean, False otherwise.

known_remotes
The names of the configured remote repositories (a list of *Remote* objects).

supports_working_tree
The opposite of bare (a boolean).

find_author()
Get the author information from the version control system.

find_branches()
Find information about the branches in the repository.

Bazaar repository support doesn't support branches so this method logs a warning message and returns an empty list. Consider using tags instead.

find_revision_id(*revision=None*)
Find the global revision id of the given revision.

find_revision_number(*revision=None*)
Find the local revision number of the given revision.

Note: Bazaar has the concept of dotted revision numbers:

> For revisions which have been merged into a branch, a dotted notation is used (e.g., 3112.1.5). Dotted revision numbers have three numbers. The first number indicates what mainline revision change is derived from. The second number is the branch counter. There can be many branches derived from the same revision, so they all get a unique number. The third number is the number of revisions since the branch started. For example, 3112.1.5 is the first branch from revision 3112, the fifth revision on that branch.

(From http://doc.bazaar.canonical.com/bzr.2.6/en/user-guide/zen.html#understanding-revision-numbers)

However we really just want to give a bare integer to our callers. It doesn't have to be globally accurate, but it should increase as new commits are made. Below is the equivalent of the git implementation for Bazaar.

---

find_tags()
Find information about the tags in the repository.

Note: The `bzr tags` command reports tags pointing to non-existing revisions as `?` but doesn't provide revision ids. We can get the revision ids using the `bzr tags --show-ids` command but this command doesn't mark tags pointing to non-existing revisions. We combine the output of both because we want all the information.

---

get_add_files_command(**filenames*)
Get the command to include added and/or removed files in the working tree in the next commit.

get_commit_command(*message*, *author=None*)
Get the command to commit changes to tracked files in the working tree.

get_create_command()
Get the command to create the local repository.

get_create_tag_command(*tag_name*)
Get the command to create a new tag based on the working tree's revision.

get_export_command(*directory*, *revision*)
Get the command to export the complete tree from the local repository.

get_pull_command(*remote=None*, *revision=None*)
Get the command to pull changes from a remote repository into the local repository.

get_push_command(*remote=None*, *revision=None*)
Get the command to push changes from the local repository to a remote repository.

update_context()
Make sure Bazaar respects the configured author.

This method first calls `Repository.update_context()` and then it sets the `$BZR_EMAIL` environment variable based on the value of `author` (but only if `author` was set by the caller).

This is a workaround for a weird behavior of Bazaar that I've observed when running under Python 2.6: The `bzr commit --author` command line option is documented but it doesn't prevent Bazaar from nevertheless reporting the following error:

```
bzr: ERROR: Unable to determine your name.
Please, set your name with the 'whoami' command.
E.g. bzr whoami "Your Name <name@example.com>"
```

2.1.4 vcs_repo_mgr.backends.git

Support for git version control repositories.

class vcs_repo_mgr.backends.git.**GitRepo**(**args*, ***kw*)
Manage git version control repositories.

classmethod contains_repository(*context*, *directory*)
Check whether the given directory contains a local repository.

static get_vcs_directory(*context*, *directory*)
Get the pathname of the directory containing the version control metadata files.

control_field
The name of the Debian control file field for git repositories (the string 'Vcs-Git').

Note: The `control_field` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *control_field* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

---

current_branch
The name of the branch that's currently checked out in the working tree (a string or `None`).

default_revision
The default revision for git repositories (the string 'master').

Note: The `default_revision` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *default_revision* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

---

friendly_name
A user friendly name for the version control system (the string 'git').

Note: The `friendly_name` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *friendly_name* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

---

is_bare
`True` if the repository has no working tree, `False` if it does.

The value of this property is computed by running the `git config --get core.bare` command.

is_clean
`True` if the working tree (and index) is clean, `False` otherwise.

The implementation of `GitRepo.is_clean` checks whether `git diff` reports any differences. This command has several variants:

1. `git diff` shows the difference between the index and working tree.
2. `git diff --cached` shows the difference between the last commit and index.
3. `git diff HEAD` shows the difference between the last commit and working tree.

The implementation of `GitRepo.is_clean` uses the third command (`git diff HEAD`) in an attempt to hide the existence of git's index from callers that are trying to write code that works with Git and Mercurial using the same Python API.

known_remotes
The names of the configured remote repositories (a list of `Remote` objects).

merge_conflicts
The filenames of any files with merge conflicts (a list of strings).

supports_working_tree
The opposite of `bare` (a boolean).

expand_branch_name(*name*)
Expand branch names to their unambiguous form.

Parameters **name** – The name of a local or remote branch (a string).

Returns The unambiguous form of the branch name (a string).

This internal method is used by methods like `find_revision_id()` and `find_revision_number()` to detect and expand remote branch names into their unambiguous form which is accepted by commands like `git rev-parse` and `git rev-list --count`.

find_author()
Get the author information from the version control system.

find_branches()
Find information about the branches in the repository.

find_branches_raw()
Find information about the branches in the repository.

find_revision_id(*revision=None*)
Find the global revision id of the given revision.

find_revision_number(*revision=None*)
Find the local revision number of the given revision.

find_tags()
Find information about the tags in the repository.

get_add_files_command(**filenames*)
Get the command to include added and/or removed files in the working tree in the next commit.

get_checkout_command(*revision*, *clean=False*)
Get the command to update the working tree of the local repository.

get_commit_command(*message*, *author=None*)
Get the command to commit changes to tracked files in the working tree.

get_create_branch_command(*branch_name*)
Get the command to create a new branch based on the working tree's revision.

get_create_tag_command(*tag_name*)
Get the command to create a new tag based on the working tree's revision.

get_create_command()
Get the command to create the local repository.

get_delete_branch_command(*branch_name*, *message=None*, *author=None*)
Get the command to delete or close a branch in the local repository.

get_export_command(*directory*, *revision*)
Get the command to export the complete tree from the local repository.

get_merge_command(*revision*)
Get the command to merge a revision into the current branch (without committing the result).

get_pull_command(*remote=None*, *revision=None*)
Get the command to pull changes from a remote repository into the local repository.

When you pull a specific branch using git, the default behavior is to pull the change sets from the remote branch into the local repository and merge them into the *currently checked out* branch.

What Mercurial does is to pull the change sets from the remote branch into the local repository and create a local branch whose contents mirror those of the remote branch. Merging is left to the operator.

In my opinion the default behavior of Mercurial is more sane and predictable than the default behavior of git and so `GitRepo` tries to emulate the default behavior of Mercurial.

When a specific revision is pulled, the revision is assumed to be a branch name and git is instructed to pull the change sets from the remote branch into a local branch with the same name.

> **Warning:** The logic described above will undoubtedly break when *revision* is given but is not a branch name. I'd fix this if I knew how to, but I don't...

get_push_command(*remote=None*, *revision=None*)
Get the command to push changes from the local repository to a remote repository.

2.1.5 vcs_repo_mgr.backends.hg

Support for Mercurial version control repositories.

class vcs_repo_mgr.backends.hg.**HgRepo**(**args*, ***kw*)
Manage Mercurial version control repositories.

static get_vcs_directory(*context*, *directory*)
Get the pathname of the directory containing the version control metadata files.

control_field
The name of the Debian control file field for Mercurial repositories (the string 'Vcs-Hg').

> **Note:** The `control_field` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *control_field* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

current_branch
The name of the branch that's currently checked out in the working tree (a string or `None`).

default_revision
The default revision for Mercurial repositories (the string 'default').

> **Note:** The `default_revision` property is a `required_property`. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *default_revision* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

friendly_name
A user friendly name for the version control system (the string 'Mercurial').

Note: The *friendly_name* property is a required_property. You are required to provide a value for this property by calling the constructor of the class that defines the property with a keyword argument named *friendly_name* (unless a custom constructor is defined, in this case please refer to the documentation of that constructor). You can change the value of this property using normal attribute assignment syntax.

is_bare
True if the repository has no working tree, False if it does.

The value of this property is computed by running the hg id command to check whether the special global revision id 000000000000 is reported.

is_clean
True if the working tree is clean, False otherwise.

known_remotes
The names of the configured remote repositories (a list of *Remote* objects).

merge_conflicts
The filenames of any files with merge conflicts (a list of strings).

supports_working_tree
Always True for Mercurial repositories.

find_author()
Get the author information from the version control system.

find_branches()
Find the branches in the Mercurial repository.

Returns A generator of *Revision* objects.

Note: Closed branches are not included.

find_revision_id(*revision=None*)
Find the global revision id of the given revision.

find_revision_number(*revision=None*)
Find the local revision number of the given revision.

find_tags()
Find information about the tags in the repository.

get_add_files_command(**filenames*)
Get the command to include added and/or removed files in the working tree in the next commit.

get_checkout_command(*revision*, *clean=False*)
Get the command to update the working tree of the local repository.

get_commit_command(*message*, *author=None*)
Get the command to commit changes to tracked files in the working tree.

This method uses the hg remove --after to match the semantics of git commit --all (which is _not_ the same as hg commit --addremove) however hg remove --after is _very_ verbose (it comments on every existing file in the repository) and it ignores the --quiet option. This explains why I've decided to silence the standard error stream (though I feel I may regret this later).

get_create_branch_command(*branch_name*)
Get the command to create a new branch based on the working tree's revision.

get_create_tag_command(*tag_name*)
Get the command to create a new tag based on the working tree's revision.

get_create_command()
Get the command to create the local repository.

get_delete_branch_command(*branch_name*, *message*, *author*)
Get the command to delete or close a branch in the local repository.

get_export_command(*directory*, *revision*)
Get the command to export the complete tree from the local repository.

get_merge_command(*revision*)
Get the command to merge a revision into the current branch (without committing the result).

get_pull_command(*remote=None*, *revision=None*)
Get the command to pull changes from a remote repository into the local repository.

get_push_command(*remote=None*, *revision=None*)
Get the command to push changes from the local repository to a remote repository.

2.1.6 vcs_repo_mgr.cli

Usage: *vcs-tool [OPTIONS] [ARGS]*

Command line program to perform common operations (in the context of packaging/deployment) on version control repositories. Supports Bazaar, Mercurial and Git repositories.

Supported options:

Option	Description
`-r`, `--repository=REPOSITORY`	Select a repository to operate on by providing the name of a repository defined in one of the configuration files ~/.vcs-repo-mgr.ini and /etc/vcs-repo-mgr.ini. Alternatively the location of a remote repository can be given. The location should be prefixed by the type of the repository (with a "+" in between) unless the location ends in ".git" in which case the prefix is optional.
`--rev`, `--revision=REVISION`	Select a revision to operate on. Accepts any string that's supported by the VCS system that manages the repository, which means you can provide branch names, tag names, exact revision ids, etc. This option is used in combination with the `--find-revision-number`, `--find-revision-id` and `--export` options. If this option is not provided a default revision is selected: "last:1" for Bazaar repositories, "master" for git repositories and "default" (not "tip"!) for Mercurial repositories.
`--release=RELEASE_ID`	Select a release to operate on. This option works in the same way as the `--revision` option. Please refer to the vcs-repo-mgr documentation for details on "releases". Although release identifiers are based on branch or tag names they may not correspond literally, this is why the release identifier you specify here is translated to a global revision id before being passed to the VCS system.
`-n`, `--find-revision-number`	Print the local revision number (an integer) of the revision given with the `--revision` option. Revision numbers are useful as a build number or when a simple, incrementing version number is required. Revision numbers should not be used to unambiguously refer to a revision (use revision ids for that instead). This option is used in combination with the `--repository` and `--revision` options.
`-i`, `--find-revision-id`	Print the global revision id (a string) of the revision given with the `--revision` option. Global revision ids are useful to unambiguously refer to a revision. This option is used in combination with the `--repository` and `--revision` options.
`--list-releases`	Print the identifiers of the releases in the repository given with the `--repository` option. The release identifiers are printed on standard output (one per line), ordered using natural order comparison.
`--select-release=RELEASE_ID`	Print the identifier of the newest release that is not newer than `RELEASE_ID` in the repository given with the `--repository` option. The release identifier is printed on standard output.
`-s`, `--sum-revisions`	Print the summed revision numbers of multiple repository/revision pairs. The repository/revision pairs are taken from the positional arguments to vcs-repo-mgr. This is useful when you're building a package based on revisions from multiple VCS repositories. By taking changes in all repositories into account when generating version numbers you can make sure that your version number is bumped with every single change.
`--vcs-control-field`	Print a line containing a Debian control file field and value. The field name will be one of "Vcs-Bzr", "Vcs-Hg" or "Vcs-Git". The value will be the repository's remote location and the selected revision (separated by a "#" character).
`-u`, `--update`	Create/update the local clone of a remote repository by pulling the latest changes from the remote repository. This option is used in combination with the `--repository` option.
`-m`, `--merge-up`	Merge a change into one or more release branches and the default branch. By default merging starts from the current branch. You can explicitly select the branch where merging should start using the `--rev`, `--revision` and `--release` options. You can also start by merging a feature branch into the selected release branch before merging the change up through later release branches and the default branch. To do so you pass the name of the feature branch as a positional argument. If the feature branch is located in a different repository you can prefix the location of the repository to the name of the feature branch with a "#" token in

`vcs_repo_mgr.cli.main()`
The command line interface of the `vcs-tool` program.

`vcs_repo_mgr.cli.print_directory(repository)`
Report the local directory of a repository to standard output.

`vcs_repo_mgr.cli.print_revision_number(repository, revision)`
Report the revision number of the given revision to standard output.

`vcs_repo_mgr.cli.print_revision_id(repository, revision)`
Report the revision id of the given revision to standard output.

`vcs_repo_mgr.cli.print_selected_release(repository, release_id)`
Report the identifier of the given release to standard output.

`vcs_repo_mgr.cli.print_releases(repository)`
Report the identifiers of all known releases of the given repository to standard output.

`vcs_repo_mgr.cli.print_summed_revisions(arguments)`
Report the summed revision numbers for the given arguments to standard output.

`vcs_repo_mgr.cli.print_vcs_control_field(repository, revision)`
Report the VCS control field for the given repository and revision to standard output.

2.1.7 vcs_repo_mgr.exceptions

Custom exception types raised by the *vcs-repo-mgr* package.

When *vcs-repo-mgr* encounters known errors it will raise an exception. Most of these exceptions have special types that capture the type of error so that the Python `except` statement can be used to handle different types of errors in different ways.

exception `vcs_repo_mgr.exceptions.VcsRepoMgrError`
Base class for exceptions directly raised by *vcs_repo_mgr*.

exception `vcs_repo_mgr.exceptions.AmbiguousRepositoryNameError`
Exception raised when an ambiguous repository name is encountered.

Raised by *find_configured_repository()* when the given repository name is ambiguous (i.e. it matches multiple repository names).

exception `vcs_repo_mgr.exceptions.NoMatchingReleasesError`
Exception raised when no matching releases are found.

Raised by *select_release()* when no matching releases are found in the repository.

exception `vcs_repo_mgr.exceptions.NoSuchRepositoryError`
Exception raised when a repository by the given name doesn't exist.

Raised by *find_configured_repository()* when the given repository name doesn't match any of the configured repositories.

exception `vcs_repo_mgr.exceptions.UnknownRepositoryTypeError`
Exception raised when a repository has an unknown type configured.

Raised by *find_configured_repository()* when it encounters a repository definition with an unknown type.

exception `vcs_repo_mgr.exceptions.WorkingTreeNotCleanError`
Exception raised when a working tree contains changes to tracked files.

Raised by *ensure_clean()* when it encounters a repository whose local working tree contains changes to tracked files.

exception `vcs_repo_mgr.exceptions.`**`MergeConflictError`**

Exception raised when a merge results in merge conflicts.

Raised by *`merge()`* when it performs a merge that results in merge conflicts.

exception `vcs_repo_mgr.exceptions.`**`MissingWorkingTreeError`**

Exception raised when working tree support is required but missing.

Raised by *`ensure_working_tree()`* when it finds that the local repository doesn't support a working tree.

CHAPTER 3

Change log

The change log lists notable changes to the project:

3.1 Changelog

The purpose of this document is to list all of the notable changes to this project. The format was inspired by Keep a Changelog. This project adheres to semantic versioning.

3.1.1 Release 4.2 (2018-04-26)

- Added this changelog.
- Added `license` key to setup script.

3.1.2 Release 4.1.3 (2018-03-28)

Bug fix: Restore support for exporting to directories with relative pathnames.

3.1.3 Release 4.1.2 (2018-03-28)

Bug fix: Make sure `update_context()` is called before `is_bare()` is invoked.

3.1.4 Release 4.1.1 (2018-03-08)

Bug fix: Resolve issue #5 by expanding remote git branch names to be unambiguous.

3.1.5 Release 4.1 (2018-03-08)

- Bug fix: Resolve issue #4 by implementing a new approach to "git branch name discovery" (that works equally well for local branches as it does for remote branches) by switching from `git branch --list --verbose` to `git for-each-ref`.
- Document MacOS compatibility, run MacOS tests on Travis CI. While I never specifically intended for vcs-repo-mgr to be used on Apple systems, a colleague of mine has been trying to do exactly this and has run into a number of issues that are probably caused by platform incompatibilities in vcs-repo-mgr and/or its dependencies.

3.1.6 Release 4.0 (2018-03-05)

- Backwards incompatible: Force internal merge tool for Mercurial. After isolating the test suite from `$HOME` my `~/.hgrc` was ignored and the following setting disappeared:

  ```
  [ui]
  merge = internal:merge
  ```

  Then I ran the *vcs-repo-mgr* test suite and `meld` popped up. Not what I was expecting from an automated test suite! Although it seems unlikely to me that someone would depend on the old behavior the introduction of `hg --config ui.merge=internal:merge` is backwards incompatible and version numbers are cheap, so I'm bumping the major version number :-).

  I do think the new behavior is a better default for the Mercurial backend given the focus of *vcs-repo-mgr* on automation, if only to make this backend match the behavior of the other backends.

- Isolate the test suite from `$HOME`. I recently added the following to my `~/.gitconfig`:

  ```
  [commit]
  gpgsign = true
  ```

  Then I ran the *vcs-repo-mgr* test suite and I was not amused :-P. This should fix the underlying more generic issue.

3.1.7 Release 3.0 (2018-03-05)

- Backwards incompatible: Raise an exception when a working tree is required but missing. This change is technically backwards incompatible in more than one way:

  1. Requiring subclasses to implement the `supports_working_tree` property breaks external subclasses (outside of my control).
  2. The new exception previously wasn't there and would never be raised, but then all of the affected operations (requiring a working tree) would likely end in an external command failure.

  For what it's worth: I don't expect these changes to bite any real life use cases.

- Merged pull request #3 to improve MacOS compatibility (by replacing `mkdir --parents` with `mkdir -p`).
- Starting from this release the files needed to generate documentation are included in source distributions.

- Moved the `coerce_pattern()` function to the humanfriendly package (because I wanted to be able to use it in qpass as well).

3.1.8 Release 2.0.1 (2017-08-02)

Bug fix: Ignore untracked files in `HgRepo.commit()`.

3.1.9 Release 2.0 (2017-07-14)

Various changes to `merge_up()`:

- Automatically create release branches.
- Skip merging up when target branch is default branch.
- Bug fix: Don't delete or close non-feature-branches.

3.1.10 Release 1.0 (2017-07-03)

Major rewrite: Named remotes, selective pushing, init support, etc.

Brain dump of changes:

- What triggered me to start on a major rewrite was that I'd gotten fed up with the current (horrible) test suite because it depends on the cloning of remote public repositories which makes it slow and fragile. To underline why it is fragile:

  I only know of one place to find public Bazaar repositories which is Launchpad.net, however cloning a Bazaar repository from Launchpad fails more often than it works. Recently the 'more often' turned into always and the test coverage of the Bazaar support in *vcs-repo-mgr* basically disappeared, without any action from me :-(.

  To improve the test suite I wanted to add support for `bzr init`, `git init` and `hg init`. However that would have made the code even uglier than it already was and so the rewrite was triggered :-).

  Support for `init` has been added, by the way :-P. I've also added support for creating tags, otherwise I wouldn't have been able to test the support for tags :-).

  After the rewrite I also rewrote the test suite, it's a completely different beast now. Stil slow, but much more robust and with quicker feedback about individual tests.
- The `push()` and `pull()` methods can work with specific revisions and `merge_up()` has been changed to pull a specific revision (the feature branch that it merges in).
- The API between the `Repository` superclass and the subclasses that implement support for a specific version control system has been changed completely, in various backwards incompatible ways.
- The new API enables introspection of 'remotes' (the repositories from which you clone/pull and the repositories that you push to) which enables `pull()` to know whether a 'default remote' is configured for any given repository.
- The new API has a class to represent commit authors, enabling less ad-hoc communication between the superclass, its subclasses and callers.
- In the process of rewriting everything I've switched to using execution contexts created by `executor.contexts`, this enables me to configure the working directory in two places instead of having to repeat the same thing in a hundred different places. This change also gives callers much more control over how external commands are executed (so much control that you can in theory run the commands on a remote system over SSH without having a version control system installed on your local system :-P).

- Support for specific version control systems has been extracted from the main `vcs_repo_mgr` module into separate modules under the `vcs_repo_mgr.backends` namespace. The modules in the backends namespace are loaded on demand.

3.1.11 Release 0.34 (2017-04-29)

- Improved the command line interface.
- Added Python 3.6 to tested Python versions.
- Refactored makefile (and Travis CI and Tox configs).

3.1.12 Release 0.33.1 (2016-11-30)

Update `stdeb.cfg` from `setup.py` (to avoid duplicate dependencies).

3.1.13 Release 0.33 (2016-10-26)

- Support for pushing between repositories.
- Started publishing wheel distributions.
- Improved documentation on raised exceptions.
- Improved logging in `Repository.update()`.
- Droped support for PyPy (refer to readme changes for details).

3.1.14 Release 0.32.1 (2016-08-04)

- Refactor setup script to fix issue #2 (`UnicodeDecodeError` in `setup.py` on Python 3).
- Run test suite on Travis CI under PyPy as well (works for me with tox :-)

3.1.15 Release 0.32 (2016-04-20)

Enable feature branch customization for `merge_up()`.

3.1.16 Release 0.31 (2016-04-20)

Support for interactive merge conflict resolution.

3.1.17 Release 0.30 (2016-03-18)

Added a command line interface for the `merge_up()` functionality.

3.1.18 Release 0.29 (2016-03-18)

Make it possible to merge changes up through release branches.

3.1.19 Release 0.28 (2016-03-18)

Make it possible to add new files to repositories.

3.1.20 Release 0.27.2 (2016-03-18)

Bug fix for previous commit (concerning the `hg remove --after` return code).

3.1.21 Release 0.27.1 (2016-03-18)

Run `hg remove --after` before `hg commit`.

3.1.22 Release 0.27 (2016-03-16)

Expose the name of the currently checked out branch.

3.1.23 Release 0.26.1 (2016-03-16)

Bug fix for `hg` command invocations, refer to the following Travis CI build failure for details: https://travis-ci.org/xolox/python-vcs-repo-mgr/jobs/116499864.

3.1.24 Release 0.26 (2016-03-16)

Make it possible to delete merged branches.

3.1.25 Release 0.25 (2016-03-16)

- Automatic `Repository` subclass registration using metaclasses.
- Move aliases from `repository_factory()` to `Repository` subclasses.
- Transform the `vcs_directory` and `exists` properties into static methods.
- Make `repository_factory()` a bit more flexible.
- Make `coerce_repository()` infer VCS types from local directories

3.1.26 Release 0.24.1 (2016-03-16)

Bug fix for unattended `git merge` support.

3.1.27 Release 0.24 (2016-03-16)

Make it possible to merge between branches.

3.1.28 Release 0.23.1 (2016-03-16)

Switch from `git diff` to `git diff HEAD` (see the inline documentation for more details).

3.1.29 Release 0.23 (2016-03-16)

Make it possible to create new branches.

3.1.30 Release 0.22.3 (2016-03-16)

- Start using the `@lazy_property` decorator.
- Bug fix for git commit message author handling.
- Stop Travis CI from launching builds for tags.

3.1.31 Release 0.22.2 (2016-03-16)

A bug fix for the test suite.

3.1.32 Release 0.22.1 (2016-03-16)

Improve handling of commit authors.

The general idea behind the new implementation is to reconcile two opposing forces:

- Don't rely on configuration files (I'm building a Python API after all).
- Respect the values in configuration files when available (because of the Do What I Mean aspect).

3.1.33 Release 0.22 (2016-03-16)

- Make it possible to commit changes.
- Add Python 3.5 to supported versions.

3.1.34 Release 0.21 (2016-03-16)

Make it possible to override the remote for `create()` and `update()` calls.

3.1.35 Release 0.20.1 (2016-03-16)

Fixed a Python 3 incompatibility in the test suite.

3.1.36 Release 0.20 (2016-03-16)

Enable updating of the working tree to different revisions.

3.1.37 Release 0.19 (2016-03-16)

- Start switching to property-manager.
- Force Read the Docs to install with `pip` instead of `python setup.py install`.

3.1.38 Release 0.18.2 (2016-03-15)

Enable `bare=None` in `find_configured_repository()`.

3.1.39 Release 0.18.1 (2016-03-15)

- Make preference for (non-)bare repositories more flexible.
- Give readme & documentation some much needed love.

3.1.40 Release 0.18 (2016-03-15)

Make it possible to check whether a working tree is clean.

3.1.41 Release 0.17 (2016-03-15)

Enable clones with working trees (non-bare clones).

3.1.42 Release 0.16 (2016-03-15)

- Make it possible to check for bare checkouts
- Document existing CONSTANTS, make `known_release_schemes` a documented constant as well.
- Implement and enforce PEP-8 and PEP-257 compliance.

3.1.43 Release 0.15.1 (2015-08-19)

Bug fix: Make sure `git fetch` *always* updates local branches.

To be honest I'm not sure why I never ran into this before, I've been using this functionality for months and updates always came in just fine based on the same version of git. Nevertheless the new `git fetch` command is the proper, documented way to do what I want `git` to do so here we go :-).

Detailed explanation: http://stackoverflow.com/a/10697486

3.1.44 Release 0.15 (2015-06-25)

- Expand `~/` and `$HOME` in configuration file (issue #1).
- Improve documentation of `find_configured_repository()`.
- Improve documentation on how `limit_vcs_updates` works.

3.1.45 Release 0.14 (2015-05-08)

- Move exceptions to separate module.
- Various documentation improvements.

3.1.46 Release 0.13 (2015-05-08)

Shortcuts to translate release identifiers to branches/tags (also got test coverage back up to +/- 97%).

3.1.47 Release 0.12 (2015-03-16)

Expose release specific functionality in CLI (listing & selection).

3.1.48 Release 0.11 (2015-03-16)

- Expose release selection in CLI (similar to revision selection).
- Fix RST format typo in `find_configured_repository()` docstring.

3.1.49 Release 0.10 (2015-02-19)

- Don't construct duplicate `Repository` objects (when possible to avoid).
- Improve argument validation in `Repository` initializer.
- Move autovivification of local clones to `Repository` initializer.
- `make install` should install 'dynamic dependencies' as well.

3.1.50 Release 0.9 (2015-02-19)

Changed release querying API, added "release selection" API.

3.1.51 Release 0.8 (2015-02-19)

Experimental support for "releases" (can be based on tags or branches).

3.1.52 Release 0.7 (2014-11-02)

Auto vivification of VCS repositories.

3.1.53 Release 0.6.4 (2014-09-14)

Support for generating Debian control file `Vcs-*` fields.

3.1.54 Release 0.6.3 (2014-09-14)

Another bug fix for Python 3.x compatibility in test suite.

3.1.55 Release 0.6.2 (2014-09-14)

Bug fix to make test suite compatible with Python 3.x. See https://travis-ci.org/xolox/python-vcs-repo-mgr/jobs/35273703.

3.1.56 Release 0.6.1 (2014-09-14)

Support for summing revision numbers from multiple repositories.

3.1.57 Release 0.6 (2014-09-14)

Support for Bazaar repositories.

3.1.58 Release 0.5 (2014-09-14)

Support for tags (also rewrote the test suite and increased test coverage).

3.1.59 Release 0.4 (2014-06-25)

Rename `limit_repo_updates` to `limit_vcs_updates` (backwards incompatible).

3.1.60 Release 0.3.2 (2014-06-22)

Try to unbreak Python 3.x tests on Travis CI.

3.1.61 Release 0.3.1 (2014-06-22)

Bug fix for 'rate limiting' of repository updates.

3.1.62 Release 0.3 (2014-06-19)

Support 'rate limiting' of repository updates.

3.1.63 Release 0.2.4 (2014-05-31)

- Change Mercurial from Debian dependency to Python dependency.
- Improve test coverage by testing command line interface.

3.1.64 Release 0.2.3 (2014-05-11)

- Automatically create local repositories on the first run.
- Bump humanfriendly requirement due to Python 3 compatibility.

3.1.65 Release 0.2.2 (2014-05-11)

Removed dead code and increased test coverage.

3.1.66 Release 0.2.1 (2014-05-10)

- Bug fix for `Revision.revision_number`.
- Improved test coverage, started using Coveralls.io.

3.1.67 Release 0.2 (2014-05-10)

- Document supported Python versions (2.6, 2.7 & 3.4).
- Switch git clone in tests to use HTTPS instead of SSH
- Start using Travis CI.

3.1.68 Release 0.1.5 (2014-05-05)

Bug fix: Include `stdeb.cfg` in source distributions (via `MANIFEST.in`).

3.1.69 Release 0.1.4 (2014-05-05)

- Document the dependency on `git` and `hg` executables.
- Document dependencies on Debian system packages in `stdeb.cfg`.

3.1.70 Release 0.1.3 (2014-05-04)

Add the usage message of the `vcs-tool` program to the documentation.

3.1.71 Release 0.1.2 (2014-05-04)

Added support for `vcs-tool --find-directory` option.

3.1.72 Release 0.1.1 (2014-05-04)

Bug fix: Added missing `humanfriendly` dependency.

3.1.73 Release 0.1 (2014-05-04)

The initial commit with support for cloning repositories, pulling updates, exporting revisions, querying revision ids and numbers for Git and Mercurial repositories.

Python Module Index

V

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W