Polyvers Documentation

Release 0.1.1a1+3.g10c2764

JRC

21/11/2018 16:16:53

Table of Contents

1 Usage: *polyvers* cmdline-tool **3**
- 1.1 Tutorial . . . 3
- 1.2 Features . . . 7
- 1.3 Recipes . . . 11

2 Usage: *polyversion* library **13**
- 2.1 Quickstart . . . 13

3 Changes **17**
- 3.1 TODOs . . . 17
- 3.2 2018-08-07: polyversion-v0.2.2a1, polyvers-v0.1.1a1 . . . 18
- 3.3 2018-07-08: polyversion-v0.2.2a0 . . . 18
- 3.4 2018-07-06: polyvers-v0.1.1a0 . . . 18
- 3.5 2018-07-05: polyversion-v0.2.1a0 . . . 18
- 3.6 2018-07-04: polyversion-v0.2.0a2 . . . 18
- 3.7 2018-06-29: polyversion-v0.1.1a3 . . . 19
- 3.8 2018-06-27: polyversion-v0.1.1a0 . . . 19
- 3.9 2018-06-06: polyvers-v0.1.0a1, polyversion-v0.1.0a7 . . . 19
- 3.10 2018-06-05: polyvers-v0.1.0a0, polyversion-v0.1.0a6: co2mpas-ready . . . 19
- 3.11 2018-06-05: polyversion-v0.1.0a5 . . . 20
- 3.12 2018-06-05: polyversion-v0.1.0a4 . . . 20
- 3.13 2018-06-03: polyversion-v0.1.0a3: *setuptools* . . . 20
- 3.14 2018-05-24: 0.0.2a10: polyvers . . . 22
- 3.15 2018-05-23: 0.0.2a9.post0: polyvers . . . 23
- 3.16 2018-05-19: 0.0.2a8: polyvers . . . 23
- 3.17 2018-05-18: 0.0.2a6: `bump!` . . . 23
- 3.18 2018-05-18: 0.0.2a5 . . . 23
- 3.19 2018-05-17: 0.0.2a1: *monorepo*! . . . 24
- 3.20 2018-01-29: 0.0.1a0: *mono-project* . . . 24

4 Contributing **25**
- 4.1 Installation . . . 25
- 4.2 Tests & integrations . . . 25
- 4.3 Similar Tools . . . 26
- 4.4 Credits . . . 27

5 Reference **29**

5.1 Polyversion (library) . . . 29
5.2 Polyvers (command-tool) . . . 33
5.3 Cmdlets . . . 40
5.4 Utilities . . . 50

6 Indices and tables **55**

Python Module Index **57**

version 0.1.1a1+3.g10c2764

updated 21/11/2018 16:16:47

Documentation https://polyvers.readthedocs.io

repository https://github.com/JRCSTU/polyvers

pypi-repo https://pypi.org/project/polyvers/, https://pypi.org/project/polyversion/

keywords version-management, configuration-management, versioning, git, monorepos, tool, library

license EUPL 1.2

A Python 3.6+ command-line tool to manage PEP-440 version-ids of dependent sub-projects hosted in a *Git monorepo*s, independently.

The key features are:

- *setuptools integration*,
- x2 *repo scheme*s (*monorepo*, *mono-project*),
- configurable *version scheme*,
- *leaf release scheme*,
- intuitive *version-bump algebra* (TODO),
- configurable *engravings*.

The *leaf version scheme* feature departs from the logic of *Similar Tools*. Specifically, when bumping the version of sub-project(s), this tool adds **+2 tags and +1 commits**:

- one *Version tag* in-trunk like `foo-proj-v0.1.0`,
- and another *Release tag* on a new *out-of-trunk commit* (leaf) like `foo-proj-r0.1.0` (the new version-ids are *engrave*d only in this release-commit):

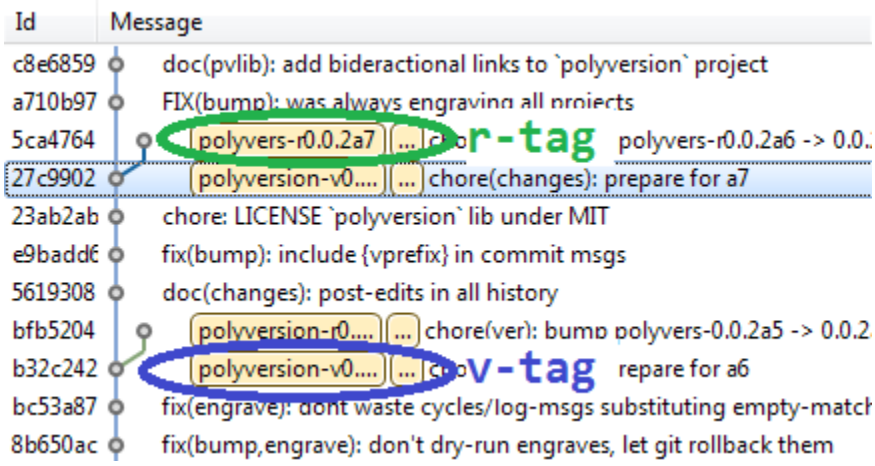

Fig. 1: Leaf-commits & version/release-tags for the two sub-project's, as shown in this repo's git history.

Note: The reason for this feature is to allow exchange code across branches (for the different sub-projects) without *engravings* getting in your way as merge-conflicts.

Additional capabilities and utilities:

- It is still possible to use plain **version tags (vtags)** like `v0.1.0`, assuming you have a single project (called hereinafter a *mono-project*)
- A separate Python 2.7+ **polyversion** project, which contains API to extract sub-project's version from past tags (provided as a separate subproject so client programs do not get `polyvers` commands transitive dependencies). The library functions as a *setuptools plugin*.

CHAPTER 1

Usage: *polyvers* cmdline-tool

1.1 Tutorial

1.1.1 Install the tool

And you get the `polyvers` command:

```
$ pip install polyvers
...
$ polyvers --version
0.0.0
$ polyvers --help
...

$ polyvers status
polyvers: Neither `setup.py` nor `.polyvers(.json|.py|.salt)` config-files found!
```

Note: Actually two projects are installed:

- **polyvers** cmd-line tool, for developing python *monorepo*s,
- **polyversion**: the base python library used by projects developed with *polyvers* tool, so that their sources can discover their subproject-version on runtime from Git.

1.1.2 Prepare project

Assuming our *monorepo* project `/monorepo.git/` contains two sub-projects, then you need enter the following configurations into your build files:

```
/monorepo.git/
    +--setup.py                # see below for contents
    +--mainprog/__init__.py
    |                          from polyversion import polyversion, polytime
    |                          __version__ = polyversion()
    |                          __updated__ = polytime()
    |                          ...
    |
    +--core-lib/
        +--setup.py:           # like above
        +--core/__init__.py    # like above
        +--...
```

Tip: You may see different sample approaches for your setup-files by looking into both *polyvers* & *polyversion* subprojects of this repo (because they eat their own dog food).

The *polyversion* library function as a *setuptools* "plugin", and adds a new `setup()` keyword `polyversion = (bool | dict)` (see `polyversion.init_plugin_kw()` for its content), which you can use it like this:

```
from setuptools import setup

setup(
    project='myname',
    version=''              # omit (or None) to abort if cannot auto-version
    polyversion={           # dict or bool
        'version_scheme: 'mono-project',
        ...  # See `polyversion.init_plugin_kw()` for more keys.
    },
    setup_requires=[..., 'polyversion'],
    ...
)
```

Hint: The `setup_requires=['polyvers']` keyword (only available with *setuptools*, and not *distutils*), enables the new `polyversion=` setup-keyword.

Alternatively, a subproject may use **PEP 0518** to pre-install *polyversion* library *before* pip-installing or launching `setup.py` script. To do that, add the `pyproject.toml` file below next to your *setup* script:

```
[build-system]
requires = ["setuptools", "wheel", "polyversion"]
```

and then you can simply import `polyversion` from your `setup.py`:

```
from setuptools import setup
from polyversion import polyversion

setup(
    project='myname',
    version=polyversion(mono_project=True)  # version implied empty string.
```

Attention: To properly install a **PEP 0518** project you need `pip-v10+` version.

1.1.3 Initialize *polyvers*

...we let the tool auto-discover the mapping of *project folders* ⇒ *project-names* and create a traitlets configuration YAML-file named as `/monorepo.git/.polyvers.py`:

```
$ cd monorepo.git

$ polyvers init --monorepo
Created new config-file '.polyvers.yaml'.

$ cat .polyvers.yaml
...
PolyversCmd:
  projects:
  - pname: mainprog     # name extracted from `setup.py`.
    basepath: .         # path discovered by the location of `setup.py`
  - pname: core
    basepath: core-lib
...

$ git add .polyvers.yaml
$ git commit -m 'add polyvers config-gile'
```

And now we can use the `polyvers` command to inspect the versions of all sub-projects:

```
$ polyvers status
- mainprog
- core
```

Indeed there are no tags in in git-history for the tool to derive and display project-versions, so only project-names are shown. With `--all` option more gets displayed:

```
$ polyvers status -a
- pname: mainprog
  basepath: .
  gitver:
  history: []
- pname: core
  basepath: core-lib
  gitver:
  history: []
```

..where `gitver` would be the result of `git-describe`.

1.1.4 Bump versions

We can now use tool to set the same version to all sub-projects:

```
$ polyvers bump 0.0.0 -f noengraves   # all projects implied, if no project-name given
00:52:06       |WARNI|polyvers.bumpcmd.BumpCmd|Ignored 1 errors while checking if at␣
↪least one version-engraving happened:
  ignored (--force=noengraves): CmdException: No version-engravings happened, bump␣
↪aborted.
00:52:07       |NOTIC|polyvers.bumpcmd.BumpCmd|Bumped projects: mainprog-0.0.0 --> 0.
↪0.0, core-0.0.0 --> 0.0.0
```

The `--force=noengraves` disables a safety check that requires at least one file modification for *engrave*ing the current version in the leaf "Release" commit (see next step).

```
 $ polyvers status
 - mainprog-v0.0.0
 - core-v0.0.0

 $ git lg    # Ok, augmented `lg` output a bit here...HEAD --> UPPER branch.
 COMMITS BRANCH TAGS                 REMARKS
 ======= ====== ==================== ======================================
      O  latest mainprog-r0.0.0      - x2 tags on "Release" leaf-commit
     /          core-r0.0.0           outside-of-trunk (not in HEAD).
    O    MASTER mainprog-v0.0.0      - x2 tags on "Version" commit
    |           core-v0.0.0           for bumping both projects to v0.0.0
    O                                - Previous commit, before version bump.

.. Hint::
   Note the difference between ``ABC-v0.0.0`` vs ``ABC-r0.0.0`` tags.

In the source code, it's only the "release" commit that has :term:`engrave`\d*␣
↪version-ids:

.. code-block:: console

 $ cat mainprog/mainprog/__init__.py    # Untouched!
 import polyvers

 __title__     = "mainprog"
 __version__ = polyvers.version('mainprog')
 ...

 $ git checkout  latest
 $ cat mainprog/mainprog/__init__.py
 import polyvers

 __title__     = "mainprog"
 __version__ = '0.0.0'
 ...

 $ git checkout  -  # to return to master.
```

1.1.5 Engrave version in the sources

Usually programs report their version somehow when run, e.g. with `cmd --version`. With *polyvers* we can derive the latest from the tags created in the previous step, using a code like this, usually in the file `/mainprog/mainprog/__init__.py`::

```
import polyvers

__title__ = "mainprog"
__version__ = polyvers.version('mainprog')
...
```

...and respectively `/core-lib/core/__init__.py`::

```
__version__ = polyvers.version('core')
```

1.1.6 Bump sub-projects selectively

Now let's add another dummy commit and then bump ONLY ONE sub-project:

```
$ git commit  --allow-empty  -m "some head work"
$ polyvers bump ^1 mainprog
00:53:07       |NOTIC|polyvers.bumpcmd.BumpCmd|Bumped projects: mainprog-0.0.0 --> 0.
↪0.1

$ git lg
COMMITS BRANCH TAGS                  REMARKS
======= ====== ==================== ========================================
     O  latest mainprog-r0.0.1.dev0 - The latest "Release" leaf-commit.
    /                                 branch `latest` was reset non-ff.
   O    MASTER mainprog-v0.0.1.dev0 - The latest "Version" commit.
   O                                - some head work
   | O         mainprog-r0.0.0      - Now it's obvious why "Release" commits
   |/          core-r0.0.0            are called "leafs".
   O           mainprog-v0.0.0
   |           core-v0.0.0
   O

$ git checkout latest
$ cat mainprog/mainprog/__init__.py
import polyvers

__title__      = "mainprog"
__version__ = '0.0.1.dev0'
...

$ cat core/core/__init__.py
import polyvers

__title__ = "core"
__version__ = '0.0.0+mainprog.0.0.1.dev0'
...
$ git checkout -
```

Notice how the the "local" part of PEP-440 (statring with `+...`) is used by the engraved version of the **un-bumped** `core` project to signify the correlated version of the **bumped** `mainprog`. This trick is not necessary for tags because they apply repo-wide, to all sub-projects.

1.2 Features

PEP 440 version ids While most versioning tools use Semantic versioning, python's `distutils` native library supports the quasi-superset, but more versatile, PEP-440 version ids, like that:

- Pre-releases: when working on new features:

```
X.YbN               # Beta release
X.YrcN  or  X.YcN   # Release Candidate
X.Y                 # Final release
```

- Post-release:

```
X.YaN.postM          # Post-release of an alpha release
X.YrcN.postM         # Post-release of a release candidate
```

- Dev-release:

```
X.YaN.devM           # Developmental release of an alpha release
X.Y.postN.devM       # Developmental release of a post-release
```

version-bump algebra When bumping, the increment over the base-version can be specified with a "relative version", which is a combination of **PEP 0440** segments and one of these modifiers: +^~= See *polyvers.vermath* for more.

repo scheme

monorepo

mono-project whether a git repo hosts a single or multiple subprojects

Rational:

When your single project succeeds, problems like these are known only too well:

Changes in **web-server** part depend on **core** features that cannot go public because the "official" **wire-protocol** is freezed.

While downstream projects using **core** as a library complain about its bloated transitive dependencies (asking why *flask* library is needed??).

So the time to "split the project" has come. But from *Lerna*:

«Splitting up large codebases into separate independently versioned packages is extremely useful for code sharing. However, making changes across many repositories is messy and difficult to track, and testing across repositories gets complicated really fast.»

So a *monorepo*[1][2] is the solution. But as Yarn put it:

«OTOH, splitting projects into their own folders is sometimes not enough. Testing, managing dependencies, and publishing multiple packages quickly gets complicated and many such projects adopt tools such as . . . »

Polyvers is such a tool.

version scheme the pattern for version-tags. 2x2 *versioning schemes* are pre-configured, for *mono-project* and *monorepo* repositories, respectively:

- *v1.2.3* (and *r1.2.3* applied on *leaf commit*s)
- *project-v1.2.3* (and *project-r1.2.3* for *leaf commit*s)

release scheme

out-of-trunk commit

leaf commit

release tag

r-tag

version tag

[1] <https://medium.com/@maoberlehner/monorepos-in-the-wild-33c6eb246cb9
[2] http://www.drmaciver.com/2016/10/why-you-should-use-a-single-repository-for-all-your-companys-projects/

v-tag Even in single-project repos, sharing code across branches may cause merge-conflicts due to the version-ids *engrave*d" in the sources. In *monorepo*s, the versions proliferate, and so does the conflicts.

Contrary to *Similar Tools*, static version-ids are engraved only in out-of-trunk (leaf) commits, and only when the sub-projects are released. In-trunk code is never touched, and version-ids are reported, on runtime, based on Git tags (like `git-describe`), so they are always up-to-date.

engrave

engravings the search-n-replace in files, to substitute the new version. Default grep-like substitutions are included, which can be re-configured in the `.polyvers.yaml` config file.

setuptools

setuptools plugin

setuptools integration The *polyversion* library function as a *setuptools* "plugin", and adds a new `setup()` keyword `polyversion = (bool | dict)` (see `polyversion.init_plugin_kw()` for its content).

bdist-check When the `setuptools:setup()` keyword `polyversion_check_bdist_enabled = True` the *setuptools plugin* aborts any *bdist...* commands if they are not run from *engrave*d sources, (ie from an *r-tag*).

To enable this check without editing the sources, add the following into your `$CWD/setup.cfg` file:

```
[global]
polyversion_check_bdist_enabled = true
...
```

Marking dependent versions across sub-projects [TODO] When bumping the version of a sub-project the "local" part of PEP-440 on all other the *dependent* sub-projects in the monorepo signify their relationship at the time of the bump.

Lock release trains as "developmental" [TODO] Specific branches can be selected always to be published into *PyPi* only as PEP-440's "Developmental" releases, meanining that users need `pip install --pre` to install from such release-trains. This is a safeguard to avoid accidentally landing half-baked code to users.

default version env-var From which env-var to read a project's *version* if git cmd fail. It does not override any value given as `default_version` keyword for *`polyversion.polyversion()`*. Also *`polyversion.polytime()`* assumes keyword `no_raise=True` if such env-var is found. [Default varname: `<pname>_VERSION`]

Other Features

- Highly configurable using traitlets, with sensible defaults; it should be possible to start using the tool without any config file (see *init* cmd), or by adding one of the flags `--monorepo`/`--mono-project` in all commands, in the face of conflicting tags.
- Always accurate version reported on runtime when run from git repos (never again wonder with which version your experimental-data were produced).

1.2.1 Known Limitations, Drawbacks & Workarounds

- PEP440 *Epoch* handling is not yet working.
- Version-bump's grammar is not yet as described in "GRAMMAR" section of command's doc:

```
$ polyvers config desc --class BumpCmd
BumpCmd(_SubCmd)
----------------
```

(continues on next page)

(continued from previous page)

```
Increase or set the version of project(s) to the (relative/absolute) version.
SYNTAX:
    polyvers config desc [OPTIONS] <version> [<project>]...
- If no project(s) specified, increase the versions on all projects.
- Denied if version for some projects is backward-in-time (or has jumped parts?);
  use --force if you might.
VERSION: - A version specifier, either ABSOLUTE, or RELATIVE to the current
version og each project:
  - *ABSOLUTE* PEP-440 version samples:
    - Pre-releases: when working on new features:
        X.YbN               # Beta release
        X.YrcN  or  X.YcN   # Release Candidate
        X.Y                 # Final release
...
```

- (not related to this tool) In `setup.py` script, the kw-argument `package_dir={'': <sub-dir>}` arg is needed for *py_modules* to work when packaging sub-projects (also useful with `find_packages()`, check this project's sources). But `<sub-dir>` must be relative to launch cwd, or else, `pip install -e <subdir>` and/or `python setup.py develop` break.
- (not related to this tool) When building projects with `python setup.py bdist_XXX`, you have to clean up your build directory (e.g. `python setup.py clean --all`) or else, the distribution package will contain the sources from all previous subprojects built. That applies also when rebuilding a project between versions.
- Installing directly from git-repos needs an engraved branch (e.g. `latest`):

  ```
  pip install git+https://github.com/JRCSTU/polyvers@latest
  ```

  If you still want to install non-engraved branhces (e.g. `master`), set the *default version env-var*; for example, since *polyvers* subproject has not customized the name of its env-var, you may install the very latest like this:

  ```
  polyvers_VERSION=1.2.3 pip install git+https://github.com/JRCSTU/polyvers
  ```

  Attention: The version given in the env-var is irrelevant. The installed version will still derive from git tags, and the local-part from the actual git-commit.

- (not related to this tool) If you don't place a `setup.py` file at the root of your git-repo (using `package_dir` argument to `setup()` function or in `find_packages()`, according to setuptools-docs), then in order to `pip install git+https://...` directly from remote URLs you have to use this official trick. For example, to install *polyversion* subproject:

  ```
  pip install "git+https://github.com/JRCSTU/polyvers@latest#egg=polyversion&
  ↪subdirectory=pvlib"
  ```

  Notice that the quotes are needed to escape the `&` char from bash. Respectively, use this to install from the very latest:

  ```
  polyversion_VERSION=1.2.3 pip install git+https://github.com/JRCSTU/polyvers
  ↪#egg=polyversion&subdirectory=pvlib"
  ```

- Set branch `latest` as default in GitHub to show *engrave*d sub-project version-ids.
- See *TODOs*.

1.3 Recipes

1.3.1 Automatically sign tags:

Add this to your `~/.polyvers.yaml`:

```
BumpCmd:
  sign_tags: true
  sign_user: <username-or-keyid>
```

CHAPTER 2

Usage: *polyversion* library

version 0.1.1a1+3.g10c2764

updated 21/11/2018 16:16:50

Documentation http://polyvers.readthedocs.io/en/latest/usage-pvlib.html

repository https://github.com/JRCSTU/polyvers

pypi-repo https://pypi.org/project/polyversion/

license MIT License

The python 2.7+ library needed by (sub-)projects managed by polyvers cmd to derive their version-ids on runtime from Git.

Specifically, the configuration file `.polyvers.yaml` is NOT read - you have to repeat any non-default configurations as function/method keywords when calling this API.

Here only a very rudimentary documentation is provided - consult *polyvers* documents provided in the link above.

Note: Only this library is (permissive) MIT-licensed, so it can be freely vendorized by any program - the respective *polyvers* command-line tool is "copylefted" under EUPLv1.2.

2.1 Quickstart

There are 4 ways to use this library:

- As a *setuptools plugin*;
- through its Python-API (to dynamically version your project);
- through its barebone cmdline tool: `polyversion` (installation required);

- through the standalone executable wheel: `bin/pvlib.run` (no installation, but sources required; behaves identically to `polyversion` command).

2.1.1 *setuptools* usage

The *polyversion* library function as a *setuptools* "plugin", and adds two new `setup()` keywords for deriving subproject versions from PKG-INFO or git tags (see `polyversion.init_plugin_kw()`):

1. **keyword: `polyversion --> (bool | dict)`** When a dict, its keys roughly mimic those in *`polyversion()`*, and can be used like this:

```
from setuptools import setup

setup(
    project='myname',
    version=''              # omit (or None) to abort if cannot auto-version
    polyversion={           # dict or bool
        'mono_project': True, # false by default
        ...  # See `polyversion.init_plugin_kw()` for more keys.
    },
    setup_requires=[..., 'polyversion'],
    ...
)
```

2. **keyword: `polyversion_check_bdist_enabled --> bool`** When it is true, the *bdist-check* is enabled, and any *bdist_** setup-commands (e.g. `bdist_wheel`) will abort if not run from *engrave*d sources (ie from an *release tag*).

   To enable this check without editing the sources, add the following into your `$CWD/setup.cfg` file:

```
[global]
polyversion_check_bdist_enabled = true
...
```

2.1.2 API usage

An API sample of using also *`polytime()`* from within your `myproject.git/myproject/__init__.py` file:

```
from polyversion import polyversion, polytime  # no hack, dependency already installed

__version__ = polyversion()  # project assumed equal to this module-name: 'myproject'
__updated__ = polytime()
...
```

Tip: Depending on your repo's *versioning scheme* (eg you have a *mono-project* repo, with version-tags simply like `vX.Y.Z`), you must add in both invocations of *`polyversion.polyversion()`* above the kw-arg `mono_project=True`.

2.1.3 Console usage

The typical command-line usage of this library (assuming you don't want to install the full blown *polyvers* command tool) is given below:

```
user@host:~/ $ polyversion --help
Describe the version of a *polyvers* projects from git tags.

USAGE:
    polyversion [PROJ-1] ...
    polyversion [-v | -V ]      # print my version information

user@host:~/ $ polyversion polyversion    # fails, not in a git repo
b'fatal: not a git repository (or any of the parent directories): .git\n'
  cmd: ['git', 'describe', '--match=cf-v*']
Traceback (most recent call last):
  File "/pyenv/site-packages/pvlib/polyversion/__main__.py", line 18, in main
    polyversion.run(*sys.argv[1:])
  File "/pyenv/site-packages/pvlib/polyversion/__init__.py", line 340, in run
    res = polyversion(args[0], repo_path=os.curdir)
  File "/pyenv/site-packages/pvlib/polyversion/__init__.py", line 262, in polyversion
    pvtag = _my_run(cmd, cwd=repo_path)
  File "/pyenv/site-packages/pvlib/polyversion/__init__.py", line 106, in _my_run
    raise sbp.CalledProcessError(proc.returncode, cmd)
subprocess.CalledProcessError: Command '['git', 'describe', '--match=cf-v*']'␣
↪returned non-zero exit status 128.

user@host:~/ $ cd polyvers.git
user@host:~/polyvers.git (dev) $ polyversion polyvers polyversion
polyvers: 0.0.2a10
polyversion: 0.0.2a9
```

2.1.4 Standalone wheel

Various ways to use the standalone wheel from *bash* (these will still work without having installed anything):

```
user@host:~/polyvers.git (master) $
user@host:~/polyvers.git (master) $ ./bin/pvlib.run polyversion
polyversion: 0.0.2a9
user@host:~/polyvers.git (master) $ python ./bin/pvlib.run --help
...
user@host:~/polyvers.git (master) $ python ./bin/pvlib.run -m polyversion  -v
version: 0.0.2a9
user@host:~/polyvers.git (master) $ PYTHONPATH=./bin/pvlib.run  python -m polyversion␣
↪ -V
version: 0.0.2a9
updated: Thu, 24 May 2018 02:47:37 +0300
```

For the rest, consult the *polyvers* project: https://polyvers.readthedocs.io

CHAPTER 3

Changes

. contents:: Releases

local

3.1 TODOs

- Parse `git describe` like setuptools_scm plugin does.
- Drop *pvcmd/pvtags.py*, and replace it with *polyversion*?
- Engravings: Not easy to extend!
  - Configurable hooks - refactor *engravings* as one of them. to run, for example, housekeeping commands on all subprojects like `pip install -e <project>` and immediately start working in "develop mode".

    This would allow housekeeping commands and *validate tests* before/after every bump:

    ```
    ## Pre-release hook
    #
    pytest tests


    ## Post-release hook
    #
    rm -r dist/* build/*;
    python setup.py sdist bdist_wheel
    twine upload dist/*whl -s
    ```

  - Add top-level engrave glob-excludes.
  - Use astor grafter.
- Refactor *version-bump algebra* to support a single modifier per segment (see `multivermath` branch).

- Lock release-trains as "alpha/beta".., specific branches can be selected Based on *version-bump algebra*), this will force users to always use `pip install --pre` to fetch such release-trains. This is a safeguard to avoid accidentally landing half-baked code to users.
- Retrofit *polyversion* library as a plugin of *polyvers* command.
- Function as plugin for other 3rd-party projects, bake a cookiecutter
- Check what happens with *no-commit*, e.g. to temporarily package a wheel.

3.2 2018-08-07: polyversion-v0.2.2a1, polyvers-v0.1.1a1

Maintenance release.

- enh: *polyversion* now logs which was cmd not found on Windows (usually it is executing `git`).
- chore: Merge all *pyupd* requests for dependencies.

3.3 2018-07-08: polyversion-v0.2.2a0

- FIX: *git < 2.15.0* was buggy with multiple match-patterns in command:

  ```
  git describe --match=...  --match=...
  ```

  (see https://github.com/git/git/blob/master/Documentation/RelNotes/2.15.0.txt)

  The `polyvers` cmd still affected by this.

3.4 2018-07-06: polyvers-v0.1.1a0

FIX: engravings were applied unsorted (based on their start-point) and offsets were miss-located when switching graft in a same file.

3.5 2018-07-05: polyversion-v0.2.1a0

- Feature: The `POLYVERSION_LOG_LEVEL` control *polyversion* verbosity. Run `polyversion -h` fo help.
- Change: minor reordering when searching version from package-metadata.
- fix: add standalone `bin/pvlib.run` from last release.
- fix: *`polyversion()`*/*`polytime()`* are guessing `basepath` keyword from the path of caller's top-package (not just from caller's fpath).

3.6 2018-07-04: polyversion-v0.2.0a2

- Version *v0.2.0a0* not in pypi, consumed for standalone `bin/pvlib.run`.
- Version *v0.2.0a1* were not finding sbling-dir versions if `pip install git+...`, and had not replaced all skip-bdist flags.

3.6.1 Features

- Teach non-engraved projects how to retrieve polyversion when pip-installed:
  - The functions `polyversion()` & `polytime()` now attempt to fetch version from package/site-package infos.
  - And the function doing this `polyversion.pkg_metadata_version()` retrofitted to:
    * search for *<pname-<version>.egg-info/PKG-INFO* in *baspath* sibling folder (before searching PKG-INFO, METADATA in *basepath*),
    * so now *basepath* always needed in `polyversion()/polytime()` functions to locate sibling dir.

3.6.2 Breaking Changes

- Rename *setuptools* flag from `skip_polyversion_check --> polyversion_check_bdist_enabled` to flip its default logic (not checking by default), since non-engraved wheels install just fine now.
- Rename the keyword of `polyversion()`/`polytime()` functions from `repo_path --> basepath` to denote its importance for retrieving the version of installed projects from sibling dirs inside `PYTHONPATH/site-packages/`.

3.7 2018-06-29: polyversion-v0.1.1a3

(change actually done in *v0.1.1a1*, just a fixes & doc-msg in *a2*)

- FIX: teach *setuptools plugin* about *default version env-var*. Now can `pip install git+https://some.git.repo/but-from/non-engraved-branch`.

3.8 2018-06-27: polyversion-v0.1.1a0

- FEAT: Introduce configurable *default version env-var* to fall-back to `<pname>_VERSION` if it exists, in case of errors (e.g. no git). The presence of such a variable also sets `polytime(no_raise=True)`, which now also support the `pname` and `default_version_env_var` kwds.

3.9 2018-06-06: polyvers-v0.1.0a1, polyversion-v0.1.0a7

Mostly docs, combined release.

- FEAT: reinstated *engravings* on `_version.py` (see previous release for rational).

3.10 2018-06-05: polyvers-v0.1.0a0, polyversion-v0.1.0a6: co2mpas-ready

- FEAT: reinstated *engravings* on `setup.py` (dropped only for a while in *2018-06-03: polyversion-v0.1.0a3: setuptools*), since, assuming clients have adopted the new *setuptools plugin* keyword, it is the *default_version* that will be engraved, which is fine.
- fix: report any version matched both from *v-tag*s and *r-tag*'s.

- fix: `bump` command does not engrave *egg*-related files.
- `polyversion` command got a bit more civilized (with logging to explain problems with related stacktraces.
- dev: don't test building wheel on travis...too much fuzzz.

3.11 2018-06-05: polyversion-v0.1.0a5

- Disable standalone-wheel hack from `pvlib/setup.py` and rely on *setuptools* plugin even for *polyversion* ONCE MORE. (but no need to update standalone, which is a wheel, unaffected by that)

3.12 2018-06-05: polyversion-v0.1.0a4

Bugfixing *polyversion* (and generate a non-buggy standalone wheel):

- FIX *polyversion* where it ignored `setup(default_version` keyword. (6519a1ba)
- fix: *polyversion* stop eating half of its own dog food: cannot reliably use *setuptools plugin* for its installation. (56a894cde)
- Monkeypatching *distutils* for *bdist-check* was failing in *PY2* due to being an "old class". (1f72baec)
- doc: fixed recommendation about how to bypass *bdist-check* to this:

  ... You may bypass this check and create a package with non-engraved sources (although it might not work correctly) by adding *skip_polyversion_check* option in your `$CWD/setup.cfg` file, like this:

  ```
  [global]
  skip_polyversion_check = true
  ...
  ```

3.13 2018-06-03: polyversion-v0.1.0a3: *setuptools*

- *v0.1.0a2'Canceled (like the previous 2), cannot release from r-tags because ''setup()'* reports version from v-tag.
  - Q: Is a new setup-keyword needed `--is-polyversion-release`?
  - A: no, just search both.
- *v0.1.0a0* had been canceled for the same reason, but somewhere down the road, the fix was reverted (*bdist-check* works for r-tag only).
- *v0.1.0a1* just marked that our `setup.py` files ate our dog food.

3.13.1 Breaking changes

- Dropped all positional-arguments from *polyversion.polyversion()*; was error-prone. They have all been converted to keyword-arguments.
- Renamed data in *polyversion* (also applied for *polyvers.pvproject.Project()*):

```
pvtag_frmt  --> pvtag_format
vtag_frmt   --> vtag_format
```

- Changed arguments in *polyversion.polyversion()* (affect also *polyvers.pvproject.Project()*):

```
default     --> default_version
tag_frmt    --> tag_format
            --> vprefixes   (new)
            --> is_release  (new)
```

- REVERTED again the *0.0.2a9* default logic to raise when it version/time cannot be derived. Now by default it raises, unless default-version or `no_raise` for *polyversion.polytime()*.
- Stopped engraving `setup.py` files ; clients should use *setuptools* plugin to derive version for those files (see new features, below)). For reference, this is the removed element from default `Project`'s configuration (in YAML):

```
globs: [setup.py]
grafts:
    - regex: -|
        (?xm)
            \bversion
            (\ *=\ *)
            .+?(,
            \ *[\n\r])+
```

- *polyversion* library searches both *v-tags* and *r-tags* (unless limited). Previously, even checked-out on an *r-tag*, both `polyversion` command and `polyvers bump` would ignore it, and report +1 from the *v-tag*!

3.13.2 Features

- The *polyversion* library function as a *setuptools* "plugin", and adds two new `setup()` keywords for deriving subproject versions from PKG-INFO or git tags (see `polyversion.init_plugin_kw()`):

  1. **keyword: `polyversion --> (bool | dict)`** When a dict, its keys roughly mimic those in *polyversion()*, and can be used like this:

     ```
     from setuptools import setup

     setup(
         project='myname',
         version=''              # omit (or None) to abort if cannot auto-
     ↪version
         polyversion={           # dict or bool
             'mono_project': True, # false by default
             ...  # See `polyversion.init_plugin_kw()` for more keys.
         },
         setup_requires=[..., 'polyversion'],
         ...
     )
     ```

  2. keyword: `skip_polyversion_check --> bool` When true, disable *bdist-check*, when false (default), any *bdist_** (e.g. `bdist_wheel`), commands will abort if not run from a *release tag*. You may bypass this check and create a package with non-engraved sources (although it might not work correctly) by invoking the setup-script from command-line like this:

```
$ python setup.py bdist_wheel --skip-polyversion-check
```

- ***bump* cmd: engrave also non-bumped projects with their `git describe`-derived** version (controlled by `--BumpCmd.engrave_bumped_only` flag).
- Assign names to engraves & grafts for readable printouts, and for refering to them from the new *Project.enabled_engarves* list. (namengraves)
- `polyversion -t` command-line tool prints the full tag (not the version) to make it easy to know if it is a v-tag or r-tag.

3.13.3 Documentation changes

- Adopt *towncrier* for compiling CHANGES. So now each code change can describe its change in the same commit, without conflicts. (towncrier)
- usage: explain how to set your projects **PEP 0518** `pyproject.toml` file & `setup_requires` keyword in `setup.py` in your script.
- add *pbr*, *incremental* and *Zest.release* in *Similar Tools* section as *setuptools* plugins.
- re-wrote and shrinked opening section using glossary terms.
- **Chore development:**
  - deps: don't pin *packaging==17.1*, any bigger +17 is fine for parsing version correctly.

3.14 2018-05-24: 0.0.2a10: polyvers

- fix: slight change of default engraving for `setup.py:version=...`.
- Remove default versions from the sources of our-own-dog-food (affects installations for developing this tool).
- refact: merged `` `pvlib.whl `` and `pvlib.run` into a single executable and importable standalone wheel in `bin/pvlib.run`, generated from `polyversion-0.0.2a9`, release below.
- doc: expand section for installing and contributing into this project.
- chore: tighten various test harnesses.

3.14.1 2018-05-24: 0.0.2a9: polyversion

2nd interim release to embed new `bin/pvlib.run`.

- INVERT by default `polyversion()`/`polytime()` functions not to raise if vtags missing.
- fix: *pvlib.run* shebang to use `#!/usr/bin/env python` to work on linux.

3.14.2 2018-05-23: 0.0.2a8: polyversion

Interim release to embed new `bin/pvlib.run`.

- FIX `polyversion` barebone command (a utility for when not installing the full *polyvers* tool).
- feat: make project-name optional in *`polyversion.polyversion()`*; if not given, defaults to caller's last segment of the module.

- doc: rudimentary explanation of how to use the lib on its own README.

3.15 2018-05-23: 0.0.2a9.post0: polyvers

- feat: add `-C` option to change project dir before running command.
- **`init` command:**
  - fix: were creating invalid `.polyvers.yaml` configuration-file unless `--monorepo`/`--mono-project` flags were given.
  - feat: include config-help in generated file only if the new `--doc` flag given.
  - feat: inform user of the projects auto-discovered and what type of config-file was generated.
- various fixes.

3.16 2018-05-19: 0.0.2a8: polyvers

- FIX(bump): was engraving all projects and not limiting to those specified in the command-line - command's syntax slightly changed.
- chore: Stop increasing *polyversion* version from now on.
- doc: fix all sphinx errors and API reference.

3.16.1 2018-05-18: 0.0.2a7

Interim release to embed re-LICENSED `pvlib/bin/pvlib.whl`, from EUPLv1.2–>MIT

3.17 2018-05-18: 0.0.2a6: `bump`!

- **`bump` command:**
  - feat: `--amend` now works
  - feat: `--engrave-only`.
  - feat: log `PRETEND` while doing actions.
  - feat: Log which files where engraved in the final message.
- fix(engrave): don't waste cycles/log-messages on empty-matches (minor).

3.18 2018-05-18: 0.0.2a5

Actually most changes happened in "interim" release *v0.0.2a2*, below.

- feat: make a standalone polyversion-lib wheel to facilitate bootstrap when installing & building from sources (and the lib is not yet installed).
- Add `bin/package.sh` that create the *pvlib* wheel as executable `dist/pvlib.run`.
- doc: fix rtd & pypi sites.

3.18.1 2018-05-18: 0.0.2a4

doc: bad PyPi landing page.

3.18.2 2018-05-17: 0.0.2a3

The *pvcmd* was actually broken so far; was missing *polyversion* lib dependency!

3.18.3 2018-05-17: 0.0.2a2

Interim release to produce executable wheel needed by next release.

3.19 2018-05-17: 0.0.2a1: *monorepo*!

- 2nd release, own "mono-project" splitted into 2-project "monorepo": - **polyvers:** cmdline tool - **polyversion:** library code for program-sources to derive version from git-tags
- *init*, *status*, *bump* and *config* commands work.
- Read/write YAML config file `.polyvers.yaml` at the git-root, and can automatically discover used configuration (from existing git *tags* or projects files).
- Support both `--monorepo` and `--mono-project` configurations.
- By default `__init__.py`, `setup.py` and `README.rst` files are engraved with bumped version.

3.19.1 2018-05-16: 0.0.2a0

broken

3.20 2018-01-29: 0.0.1a0: *mono-project*

- First release on PyPI as *mono-project*

CHAPTER 4

Contributing

Contributions and issue-reports are welcome, with no strings attached; project is at a very early stage.

4.1 Installation

Install both sub-projects in *develop* mode using `pip`

```
$ git clone  https://github.com/JRCSTU/polyvers  polyvers.git
$ cd polyvers.git
$ pip install  -e pvlib[test]  -e .[test]
```

Caution: Installing *egg* with `python setup.py install develop` is not tested, and may fail to work correctly.

4.2 Tests & integrations

test-suite pytest

linters flake8, mypy

integration-services

- travis (CI: linux),
- appveyor (CI: Windows),
- coveralls.io (coverage),
- codacy.com (code-quality)
- pyup.io (dependency-tracking)

commit-messages guidelines from *angular* repo: https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#-git-commit-guidelines

changes-aggregation towncrier

4.3 Similar Tools

Bumped across these projects while building it...

bumpversion The original **bumpversion** project; development stopped after 2015: (recomended also by *python guide*) https://github.com/peritus/bumpversion

bump2version active clone of the original: https://github.com/c4urself/bump2version

releash another *monorepo*s managing tool, that publishes also to PyPi: https://github.com/maartenbreddels/releash

Git Bump bump version using git-hooks: https://github.com/arrdem/git-bump

Lerna A tool for managing JavaScript projects with multiple packages. https://lernajs.io/

Pants a build system designed for codebases that: - Are large and/or growing rapidly. - Consist of many subprojects that share a significant amount of code. - Have complex dependencies on third-party libraries. - Use a variety of languages, code generators and frameworks. - https://www.pantsbuild.org/

pbr a `setup_requires` library that injects sensible default and behaviors into your *setuptools*. Crafted for *Semantic Versioning*, maintained for OpenStack projects. https://docs.openstack.org/pbr/

Zest.releaser easy releasing and tagging for Python packages; make easy, quick and neat releases of your Python packages. You need to change the version number, add a new heading in your changelog, record the release date, svn/git/bzr/hg tag your project, perhaps upload it to pypi... *zest.releaser* takes care of the boring bits for you. (recomended also by *python guide*) http://zestreleaser.readthedocs.io/

incremental a small *setuptools* plugin library that versions Python projects. https://github.com/twisted/incremental

changes Manages the release of a Python Library (intuitive logo, recomended also by *python guide*):

- Auto generates changelog entries from commit messages
- CLI that follows Semantic Versioning principles to auto-increment the library version
- Runs the library tests
- Checks the package installation from a tarball and PyPi
- Uploads the distribution to PyPi
- Tags the GitHub repository

https://github.com/michaeljoseph/changes

setuptools_scm managing versions in scm metadata instead of declaring them as the version argument or in a scm managed file, apart from handling file finders for the supported scm's. (recomended also by *python guide*) https://pypi.org/project/setuptools_scm/

python guide There is a dedicated guide for this problem in pythons docs: https://packaging.python.org/guides/single-sourcing-package-version/

Find more than 34 similar projects in GitHub: and in awesome: https://github.com/korfuri/awesome-monorepo.

4.4 Credits

- Contains a function from the BSD-tool :term‘pbr‘ to fetch version from *pkg-metadata* when invoked as a *setuptools plugin* from inside an egg.
- This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.
- Using towncrier for generating CHANGES.

CHAPTER 5

Reference

5.1 Polyversion (library)

polyversion	Python-2.7-safe, no-deps code to discover sub-project versions in Git *polyvers* monorepos.
polyversion.polyversion(**kw)	Report the *pvtag* of the *pname* in the git repo hosting the source-file calling this.
polyversion.polytime(**kw)	The timestamp of last commit in git repo hosting the source-file calling this.
polyversion.pkg_metadata_version(pname[, ...])	Get the version from package metadata if present.
polyversion.setuplugin	A *setuptools* plugin with x2 `setup()` kwds and monkeypatch all `bdist...` cmds.
polyversion.setuplugin.init_plugin_kw(dist, ...)	A *setuptools* kwd for deriving subproject versions from PKG-INFO or git tags.
polyversion.setuplugin.check_bdist_kw(dist, ...)	A *setuptools* kwd for aborting *bdist...* commands if not on r-tag.

5.1.1 Module: `polyversion`

Python-2.7-safe, no-deps code to discover sub-project versions in Git *polyvers* monorepos.

The *polyvers* version-configuration tool is generating **pvtags** like:

```
proj-foo-v0.1.0
```

And assuming *polyversion()* is invoked from within a Git repo, it may return either `0.1.0` or `0.1.0+2.gcaffe00`, if 2 commits have passed since last *pvtag*.

Also, this library function as a *setuptools* "plugin" (see `setuplugin`).

Finally, the wheel can be executed like that:

```
python polyversion-*.whl --help
```

polyversion.**polyversion**(***kw*)

Report the *pvtag* of the *pname* in the git repo hosting the source-file calling this.

Parameters

- **pname** (*str*) – The project-name, used as the prefix of pvtags when searching them. If not given, defaults to the *last segment of the module-name of the caller*.

  Attention: when calling it from `setup.py` files, auto-deduction above will not work; you must supply a project name.

- **default_version** (*str*) – What *version* to return if git cmd fails. Set it to *None* to raise if no *vtag* found.

  Tip: For cases where a shallow git-clone does not finds any *vtags* back in history, or simply because the project is new, and there are no *vtags*, we set default-version to empty-string, to facilitate pip-installing these projects from sources.

- **default_version_env_var** (*str*) – Override which env-var to read *version* from, if git cmd fails [Default: `<pname>_VERSION`]
- **mono_project** (*bool*) –
  - false: (default) *monorepo*, ie multiple sub-projects per git-repo. Tags formatted by *pvtags* *pvtag_format* & `pvtag_regex` (like `pname-v1.2.3`).
  - true: *mono-project*, ie only one project in git-repo Tags formatted as *vtags* *vtag_format* & *vtag_regex*. (like `v1.2.3`).
- **tag_format** (*str*) – The **PEP 3101** pattern for creating *pvtags* (or *vtags*).
  - It receives 3 parameters to interpolate: `{pname}, {vprefix}, {version} = '*'`.
  - It is used also to generate the match patterns for `git describe --match <pattern>` command.
  - It overrides *mono_project* arg.
  - See *pvtag_format* & *vtag_format*
- **tag_regex** (*regex*) – The regex pattern breaking apart *pvtags*, with 3 named capturing groups:
  - `pname`,
  - `version` (without the 'v'),
  - `descid` (optional) anything following the dash('-') after the version in `git-describe` result.
  - It is given 2 **PEP 3101** parameters `{pname}, {vprefix}` to interpolate.
  - It overrides *mono_project* arg.
  - See **PEP 0426** for project-name characters and format.
  - See `pvtag_regex` & *vtag_regex*

- **vprefixes** (*str*) – a 2-element array of str - `tag_vprefixes` assumed when not specified
- **is_release** – a 3-state boolean used as index into `tag_vprefixes`:
  - false: v-tags searched;
  - true: r-tags searched;
  - None: both tags searched.
- **basepath** (*str*) – The path of the outermost package inside the git repo hosting the project; if missing, assumed as the dirname of the calling code's package, or (if no module) cwd(.).
- **git_options** – a str or an iterator of (converted to str) options to pass to `git describe` command (empty by default). If a string, it is splitted by spaces.
- **return_all** – when true, return the 3-tuple (tag, version, desc-id) (not just version)

Returns The version-id (or 3-tuple) derived from the *pvtag*, or *default* if command failed/returned nothing, unless None, in which case, it raises.

Raises **CalledProcessError** – if it cannot find any vtag and *default_version* is None (e.g. no git cmd/repo, no valid tags)

Tip: It is to be used, for example, in package `__init__.py` files like this:

```
__version__ = polyversion()
```

Or from any other file:

```
__version__ = polyversion('myproj')
```

Note: This is a python==2.7 & python<3.6 safe function; there is also the similar function with elaborate error-handling `polyvers.pvtags.describe_project()` in the full-blown tool *polyvers*.

---

polyversion.**polytime**(***kw*)

The timestamp of last commit in git repo hosting the source-file calling this.

Parameters

- **no_raise** (*str*) – If true, never fail and return current-time. Assumed true if a *default version env-var* is found.
- **basepath** (*str*) – The path of the outermost package inside the git repo hosting the project; if missing, assumed as the dirname of the calling code's package.
- **pname** (*str*) – The project-name used only as the prefix for *default version env-var*. If not given, defaults to the *last segment of the module-name of the caller*. Another alternative is to use directly the *default_version_env_var* kwd.

  Attention: when calling it from `setup.py` files, auto-deduction above will not work; you must supply a project name.

- **default_version_env_var** (*str*) – Override which env-var to read *version* from, if git cmd fails [Default: `<pname>_VERSION`]

Returns the commit-date if in git repo, or now; **RFC 2822** formatted

`polyversion.`**`decide_vprefixes`**(*vprefixes*, *is_release*)
Decide v-tag, r-tag or both; no surprises params, return always an array.

`polyversion.`**`vtag_format = '{vprefix}{version}'`**
Like *`pvtag_format`* but for *mono-project* version-tags.

`polyversion.`**`vtag_regex = '(?xmi)\n ^(?P<pname>)\n {vprefix}(?P<version>\\d[^-]*)\n (?:-(?P<`**
Like *`pvtag_format`* but for *mono-project* version-tags.

`polyversion.`**`pvtag_format = '{pname}-{vprefix}{version}'`**
The default pattern for *monorepos* version-tags, receiving 3 **PEP 3101** interpolation parameters:

```
{pname}, {version} = '*', {vprefix} = tag_vprefixes[0 | 1]
```

The match patterns for `git describe --match <pattern>` are generated by this.

5.1.2 Module: `polyversion.setuplugin`

A *setuptools* plugin with x2 `setup()` kwds and monkeypatch all `bdist...` cmds.

Tip: Set *envvar[DISTUTILS_DEBUG]* to debug it. From https://docs.python.org/3.7/distutils/setupscript.html#debugging-the-setup-script

`polyversion.setuplugin.`**`init_plugin_kw`**(*dist*, *attr*, *kw_value*)
A *setuptools* kwd for deriving subproject versions from PKG-INFO or git tags.

Parameters

- **`dist`** – class:*distutils.Distribution*
- **`attr`** (*`str`*) – the name of the keyword
- **`kw_value`** – The content of the new `setup(polyversion=...)` keyword.

  SYNTAX: `'polyversion': (<bool> | <dict>)`

  When it is a dict, its keys roughly mimic those in *`polyversion()`* except those differences:

  param pname absent; derived from `setup(name=...)` keyword

  param default_version absent; derived from `setup(version=...)` keyword:

  - if *None*/not given, any problems will be raised, and `setup.py` script wil abort
  - if `version` is a (possibly empty) string, this will be used in case version cannot be auto-retrieved.

  param is_release absent; always *False* when deriving the version, and *True* when bdist-checking

  param basepath if not given, derived from `setup(package_dirs={...})` keyword or '.' (and never from caller-stack).

  See *`polyversion()`* for keyword-dict's content.

- It tries first to see if project contained in a distribution-archive (e.g. a "wheel"), and tries to derive the version from egg-infos. Then it falls through retrieving it from git tags.

> **Tip:** For cases where a shallow git-clone does not finds any *vtags* back in history, or simply because the project is new, and there are no *vtags*, we set default-version to empty-string, to facilitate pip-installing these projects from sources.

polyversion.setuplugin.**check_bdist_kw**(*dist*, *_attr*, *kw_value*)

A *setuptools* kwd for aborting *bdist...* commands if not on r-tag.

SYNTAX: `'polyversion_check_bdist_enabled': <any>`

When <*any*> evaluates to false (default), any *bdist...* (e.g. `bdist_wheel`), *setuptools* commands will abort if not run from a *release tag*.

By default it this check is bypassed. To enable it, without editing your sources add this in your `$CWD/setup.cfg` file:

```
[global]
polyversion_check_bdist_enabled = true
...
```

- Ignored, if *polyversion* kw is not enabled.
- Registered in *distutils.setup_keywords entry_point* of this project's `setup.py` file.

5.2 Polyvers (command-tool)

polyvers	Top-level package for *polyvers* version-configuration tool.
polyvers.cli	The code of *polyvers* shell-commands.
polyvers.bumpcmd	The command that actually bumps versions.
polyvers.pvproject	The main structures of *polyvers*.
polyvers.pvtags	Git code to make/inspect sub-project "(p)vtags" and respective commits in (mono)repos.
polyvers.engrave	Search and replace version-ids in files.
polyvers.vermath	Validate absolute versions or add relative ones on top of a base absolute.
polyvers.cmdlet.cfgcmd	Commands to inspect configurations and other cli infos.
polyvers.cmdlet.cmdlets	Utils for building elaborate Commands/Sub-commands with traitlets[1] Application.
polyvers.cmdlet.errlog	Suppress or ignore exceptions collected in a nested contexts.
polyvers.cmdlet.interpctxt	Enable Unicode-trait to pep3101-interpolate *{key}* patterns from "context" dicts.
polyvers.utils.mainpump	Utils pumping results out of yielding functions for *main()*.
polyvers.utils.logconfutils	Utils for configuring and using elaborate logs and handling *main()* failures.
polyvers.utils.oscmd	Utility to call OS commands through `subprocess.run()` with logging.
polyvers.utils.fileutil	Generic utils.

5.2.1 Module: `polyvers.cli`

The code of *polyvers* shell-commands.

class polyvers.cli.**InitCmd**(**args*, ***kw*)
Generate configurations based on directory contents.

initialize(*argv=None*)
Invoked after __init__() by *make_cmd()* to apply configs and build subapps.

Parameters **argv** – If undefined, they are replaced with `sys.argv[1:]`!

It parses cl-args before file-configs, to detect sub-commands and update any `config_paths`, then it reads all file-configs, and then re-apply cmd-line configs as overrides (trick copied from *jupyter-core*).

run(**args*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **args** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

class polyvers.cli.**LogconfCmd**(**args*, ***kw*)
Write a logging-configuration file that can filter logs selectively.

run(**args*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **args** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

class polyvers.cli.**PolyversCmd**(***kwargs*)
Bump independently PEP-440 versions of sub-project in Git monorepos.

SYNTAX: {cmd_chain} <sub-cmd> ...

bootstrapp_projects() → None
Ensure valid configuration exist for monorepo/mono-project(s).

Raises ***CmdException*** – if cwd not inside a git repo

collect_app_infos()
Provide extra infos to *config infos* subcommand.

parse_command_line(*argv=None*)
Parse the command line arguments.

class polyvers.cli.**StatusCmd**(**args*, ***kw*)
List the versions of project(s).

SYNTAX: {cmd_chain} [OPTIONS] [<project>]...

run(**pnames*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **args** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

polyvers.cli.**merge_dict**(*dct*, *merge_dct*)
Recursive dict merge. Inspired by :meth:`dict.update()`, instead of updating only top-level keys, dict_merge recurses down into dicts nested to an arbitrary depth, updating keys. The `merge_dct` is merged into `dct`. :param dct: dict onto which the merge is executed :param merge_dct: dct merged into dct :return: None

[1] http://traitlets.readthedocs.io/

Adapted from: https://gist.github.com/angstwad/bf22d1822c38a92ec0a9

`polyvers.cli.`**`run`**(*argv=()*, *cmd_consumer=None*, ***app_init_kwds*)
Handle some exceptions politely and return the exit-code.

Parameters

- **`argv`** – Cmd-line arguments, nothing assumed if nohing given.
- **`cmd_consumer`** – Specify a different main-mup, `mpu.PrintConsumer` by default. See `mpu.pump_cmd()`.

5.2.2 Module: `polyvers.bumpcmd`

The code of *polyvers* shell-commands.

`class` `polyvers.cli.`**`InitCmd`**(**args*, ***kw*)
Generate configurations based on directory contents.

`initialize`(*argv=None*)
Invoked after __init__() by *make_cmd()* to apply configs and build subapps.

Parameters **`argv`** – If undefined, they are replaced with `sys.argv[1:]`!

It parses cl-args before file-configs, to detect sub-commands and update any `config_paths`, then it reads all file-configs, and then re-apply cmd-line configs as overrides (trick copied from *jupyter-core*).

`run`(**args*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **`args`** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

`class` `polyvers.cli.`**`LogconfCmd`**(**args*, ***kw*)
Write a logging-configuration file that can filter logs selectively.

`run`(**args*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **`args`** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

`class` `polyvers.cli.`**`PolyversCmd`**(***kwargs*)
Bump independently PEP-440 versions of sub-project in Git monorepos.

SYNTAX: {cmd_chain} <sub-cmd> ...

`bootstrapp_projects`() → None
Ensure valid configuration exist for monorepo/mono-project(s).

Raises ***`CmdException`*** – if cwd not inside a git repo

`collect_app_infos`()
Provide extra infos to *config infos* subcommand.

`parse_command_line`(*argv=None*)
Parse the command line arguments.

`class` `polyvers.cli.`**`StatusCmd`**(**args*, ***kw*)
List the versions of project(s).

SYNTAX: {cmd_chain} [OPTIONS] [<project>]...

run (**pnames*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **args** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

`polyvers.cli.`**merge_dict** (*dct*, *merge_dct*)
Recursive dict merge. Inspired by :meth:`dict.update()`, instead of updating only top-level keys, dict_merge recurses down into dicts nested to an arbitrary depth, updating keys. The `merge_dct` is merged into `dct`. :param dct: dict onto which the merge is executed :param merge_dct: dct merged into dct :return: None

Adapted from: https://gist.github.com/angstwad/bf22d1822c38a92ec0a9

`polyvers.cli.`**run** (*argv=()*, *cmd_consumer=None*, ***app_init_kwds*)
Handle some exceptions politely and return the exit-code.

Parameters

- **argv** – Cmd-line arguments, nothing assumed if nohing given.
- **cmd_consumer** – Specify a different main-mup, `mpu.PrintConsumer` by default. See `mpu.pump_cmd()`.

5.2.3 Module: `polyvers.pvproject`

The main structures of *polyvers*:

```
Project 1-->* Engrave 1-->* Graft
```

class `polyvers.pvproject.`**Engrave** (***kwargs*)
Muliple file-patterns to search'n replace.

class `polyvers.pvproject.`**Graft** (***kwargs*)
Instructions on how to search'n replace some text.

collect_matches (*fbytes: bytes*, *project: polyvers.pvproject.Project*) → List[Match[~AnyStr]]

Returns all *hits*; use `sliced_matches()` to apply any slices

class `polyvers.pvproject.`**Project** (***kwargs*)
Configurations for projects, in general, and specifically for each one.

git_describe (**git_args*, *include_lightweight=False*, *is_release=None*, ***git_flags*)
Gets sub-project's version as derived from `git describe` on its *pvtag*.

Parameters

- **include_lightweight** – Consider also non-annotated tags when derriving description; equivalent to `git describe --tags` flag.
- **is_release** – a 3-state boolean used as index into `tag_vprefixes`:
  - false: v-tags searched;
  - true: r-tags searched;
  - None: both tags searched.
- **git_args** – CLI options passed to `git describe` command. See *oscmd.PopenCmd* on how to specify cli options using python functions, e.g. `('*-v*', '*-r*')`
- **git_flags** – CLI flags passed to `git describe` command.

- See *oscmd.PopenCmd* on how to specify cli flags using python functions, e.g. `(all=True)`.
- See https://git-scm.com/docs/git-describe

Returns the *pvtag* of the current project, or raise

Raises

- ***GitVoidError*** – if sub-project is not *pvtagged*.
- ***NoGitRepoError*** – if CWD not within a git repo.
- **sbp.CalledProcessError** – for any other error while executing *git*.

Tip: There is also the python==2.7 & python<3.6 safe `polyvers.polyversion()` `()` for extracting just the version part from a *pvtag*; use this one from within project sources.

Tip: Same results can be retrieved by this *git* command:

```
git describe --tags --match <PROJECT>-v*
```

where `--tags` is needed to consider also non-annotated tags, as `git tag` does.

---

is_good()
If format patterns are missing, spurious NPEs will happen when using project.

last_commit_tstamp()
Report the timestamp of the last commit of the git repo.

Returns last commit's timestamp in **RFC 2822** format

Raises

- ***GitVoidError*** – if there arn't any commits yet or CWD not within a git repo.
- **sbp.CalledProcessError** – for any other error while executing *git*.

Note: Same results can be retrieved by this *git* command:

```
git describe --tags --match <PROJECT>-v*
```

where `--tags` is needed to consider also unannotated tags, as `git tag` does.

---

pvtags_history
Return the full *pvtag* history for the project, if any.

Raises **AssertionError** – If used before `populate_pvtags_history()` applied on this project.

set_new_version(*version_bump: str = None*)

Parameters **version_bump** – relative or absolute

tag_fnmatch(*is_release=False*)
The glob-pattern finding *pvtags* with `git describe --match <pattern>` cmd.

Parameters **is_release** – *False* for version-tags, *True* for release-tags

By default, it is interpolated with two **PEP 3101** parameters:

```
{pname}   <-- this Project.pname
{version} <-- '*'
```

tag_regex(*is_release=False*) → Pattern[~AnyStr]
Interpolate and compile as regex.

Parameters **is_release** – *False* for version-tags, *True* for release-tags

tag_version_commit(*msg*, *, *is_release=False*, *amend=False*, *sign_tag=None*, *sign_user=None*)
Make a tag on current commit denoting a version-bump.

Parameters **is_release** – *False* for version-tags, *True* for release-tags

version_from_pvtag(*pvtag: str*, *is_release: Optional[bool] = None*) → Optional[str]
Extract the version from a *pvtag*.

5.2.4 Module: polyvers.pvtags

Git code to make/inspect sub-project "(p)vtags" and respective commits in (mono)repos.

There are 3 important methods/functions calling Git:

- `Project.git_describe()` that fetches the same version-id that *polyversion.polyversion()* would return, but with more options.
- `Project.last_commit_tstamp()`, same as above.
- *populate_pvtags_history()* that populates *pvtags* on the given project instances; certain pvtag-related Project methods would fail if this function has not been applies on a project instance.

exception polyvers.pvtags.**GitError**
A (maybe benign) git-related error

exception polyvers.pvtags.**GitVoidError**
Sub-project has not yet been version with a *pvtag*.

exception polyvers.pvtags.**NoGitRepoError**
Command needs a git repo in CWD.

polyvers.pvtags.**git_project_errors_handled**(*pname*)
Report *pname* involved to the user in case tags are missing.

polyvers.pvtags.**git_restore_point**(*restore_head=False*, *heads=True*, *tags=True*)
Restored checked out branch to previous state in case of errors (or if forced).

Parameters **restore** – if true, force restore at exit, otherwise, restore only on errors

polyvers.pvtags.**make_match_all_pvtags_project**(***project_kw*) → polyvers.pvproject.Project
Make a `Project` capturing any *pvtag*.

Useful as a "catch-all" last project in *populate_pvtags_history()*, to capture *pvtags* not captured by any other project.

polyvers.pvtags.**make_match_all_vtags_project**(***project_kw*) → polyvers.pvproject.Project
Make a `Project` capturing any simple *vtag* (e.g. `v0.1.0`).

Useful as a "catch-all" last project in *populate_pvtags_history()*, to capture *vtags* not captured by any other project.

polyvers.pvtags.**make_pvtag_project**(*pname: str = '<monorepo>'*, ***project_kw*) → polyvers.pvproject.Project

Make a `Project` for a subprojects hosted at git monorepos.

- Project versioned with *pvtags* like `foo-project-v0.1.0`.

polyvers.pvtags.**make_vtag_project**(*pname: str = '<mono-project>'*, ***project_kw*) → polyvers.pvproject.Project

Make a `Project` for a single project hosted at git repos root (not "monorepos").

- Project versioned with tags simple *vtags* (not *pvtags*) like `v0.1.0`.

polyvers.pvtags.**populate_pvtags_history**(**projects*, *include_lightweight=False*, *is_release=False*)

Updates `pvtags_history` on given *projects* (if any) in ascending order.

Parameters

- **projects** – the projects to search *pvtags* for
- **include_lightweight** – fetch also non annotated tags; note that by default, `git-describe` does consider lightweight tags unless `--tags` given.
- **is_release** – *False* for version-tags, *True* for release-tags

Raises **sbp.CalledProcessError** – if *git* executable not in PATH

Note: Internally, *pvtags* are populated in `_pvtags_collected` which by default it is `None`. After this call, it will be a (possibly empty) list. Any pre-existing *pvtags* are removed from all projects before collecting them anew.

Tip: To collect all *pvtags* OR *vtags* only, use pre-defined projects generated by `make_project_matching_*()` functions.

5.2.5 Module: polyvers.engrave

Search and replace version-ids in files.

class polyvers.engrave.**FileProcessor**(***kwargs*)

polyvers.engrave.**glob_files**(*patterns: List[str], mybase: Union[str, pathlib.Path] = '.', other_bases: Sequence[Union[str, pathlib.Path]] = None*) → List[pathlib.Path]

Glob files in *mybase* but not in *other_bases* (unless bases coincide).

- Supports exclude patterns: `!foo`.
- If *mybase* is in *other_bases*, it doesn't change the results.

polyvers.engrave.**overlapped_matches**(*matches: Sequence[Match[~AnyStr]], no_touch=False*) → Set[Match[~AnyStr]]

Parameters **no_touch** – if true, all three (0,1), (1,2) (2,3) overlap on 1 and 2.

5.2.6 Module: polyvers.vermath

Validate absolute versions or add relative ones on top of a base absolute.

class polyvers.vermath.**Pep440Version**(**args*, ***kwargs*)
A trait parsing text like python "slice" expression (ie `-10::2`).

cast(*value*)
Return a value that will pass validation.

This is only triggered if the value in question is `castable()`.

exception polyvers.vermath.**VersionError**

polyvers.vermath.**add_versions**(*v1: Union[str, packaging.version.Version]*, **rel_versions*) → packaging.version.Version
return the "sum" of the the given two versions.

5.3 Cmdlets

5.3.1 Module: `polyvers.cmdlet.cfgcmd`

Commands to inspect configurations and other cli infos.

class polyvers.cmdlet.cfgcmd.**ConfigCmd**(***kwds*)
Commands to manage configurations and other cli infos.

class polyvers.cmdlet.cfgcmd.**DescCmd**(***kwds*)
List and print help for configurable classes and parameters.

SYNTAX {cmd_chain} [-l] [-c] [-t] [-v] [<search-term> ...]

- If no search-terms provided, returns all.
- Search-terms are matched case-insensitively against '<class>.<param>', or against '<class>' if –class.
- Use –verbose (-v) to view config-params from the whole hierarchy, that is, including those from intermediate classes.
- Use –class (-c) to view just the help-text of classes.
- Results are sorted in "application order" (later configurations override previous ones); use –sort for alphabetical order.

run(**args*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **args** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

class polyvers.cmdlet.cfgcmd.**InfosCmd**(***kwargs*)
List paths and other intallation infos.

Some of the environment-variables affecting configurations:

HOME, USERPROFILE, [where configs & DICE projects are stored] (1st one defined wins)

<APPNAME>_CONFIG_PATHS : where to read configuration-files.

run(**args*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **args** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

class polyvers.cmdlet.cfgcmd.**ShowCmd**(***kwds*)
Print configurations (defaults | files | merged) before any validations.

SYNTAX {cmd_chain} [OPTIONS] [–source=(merged | default)] [<search-term-1> ...] {cmd_chain} [OPTIONS] –source file

- Search-terms are matched case-insensitively against '<class>.<param>'.
- Use –verbose to view values for config-params as they apply in the whole hierarchy (not
- Results are sorted in "application order" (later configurations override previous ones); use –sort for alphabetical order.
- Warning: Defaults/merged might not be always accurate!
- Tip: you may also add *–show-config* global option on any command to view configured values accurately on runtime.

initialize(*argv=None*)
Override to store file-configs separately (before merge).

run(**args*)
Leaf sub-commands must inherit this instead of `start()` without invoking `super()`.

Parameters **args** – Invoked by `start()` with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

polyvers.cmdlet.cfgcmd.**prepare_search_map**(*all_classes*, *own_traits*)

Parameters **own_traits** – bool or None (no traits)

Returns `{'ClassName.trait_name': (class, trait)` When *own_traits* not None, `{clsname: class}`) otherwise. Note: 1st case might contain None as trait!

5.3.2 Module: polyvers.cmdlet.cmdlets

Utils for building elaborate Commands/Sub-commands with traitlets[1] Application.

Examples:

To run a base command, use this code:

```
cd = MainCmd.make_cmd(argv, **app_init_kwds)  ## `sys.argv` used if `argv` is `None`!
cmd.start()
```

To run nested commands and print its output, use `baseapp.chain_cmds()` like that:

```
cmd = chain_cmds([MainCmd, Sub1Cmd, Sub2Cmd], argv)  ## `argv` without sub-cmds
sys.exit(baseapp.pump_cmd(cmd.start()) and 0)
```

Of course you can mix'n match.

Configuration and Initialization guidelines for *Spec* and *Cmd* classes

0. The configuration of `HasTraits` instance gets stored in its `config` attribute.
1. A `HasTraits` instance receives its configuration from 3 sources, in this order:

1. code specifying class-properties or running on constructors;
2. configuration files (*json* or `.py` files);

3. command-line arguments.
2. Constructors must allow for properties to be overwritten on construction; any class-defaults must function as defaults for any constructor `**kwds`.
3. Some utility code depends on trait-defaults (i.e. construction of help-messages), so for certain properties (e.g. description), it is preferable to set them as traits-with-defaults on class-properties.
4. Listen Good Bait after 1:43.

class polyvers.cmdlet.cmdlets.**CfgFilesRegistry**(*supported_cfg_extensions=['.json', '.py']*)

Locate and account extensioned files (by default `.json|.py`).

- Collects a Locate and (`.json|.py`) files present in the *path_list*, or
- Invoke this for every "manually" visited config-file, successful or not.
- Files collected earlier should override next ones.

collect_fpaths(*path_list*)

Collects all (`.json|.py`) files present in the *path_list*, (descending order).

Parameters **path_list** (*List[Text]*) – A list of paths (absolute, relative, dir or folders).

Returns fully-normalized paths, with ext

config_tuples

The consolidated list of loaded 2-tuples `(folder, fname(s))`.

Sorted in descending order (1st overrides later).

head_folder()

The *last* existing visited folder (if any), even if not containing files.

visit_file(*fpath*, *loaded*)

Invoke this in ascending order for every visited config-file.

Parameters **loaded** (*bool*) – Loaded successful?

class polyvers.cmdlet.cmdlets.**Cmd**(***kwargs*)

Common machinery for all (sub)commands.

classmethod clear_instance()

unset _instance for this class and singleton parents.

emit_description()

Yield lines with the application description.

emit_examples()

Yield lines with the usage and examples.

This usage string goes at the end of the command line help string and should contain examples of the application's usage.

emit_help_epilogue(*classes=None*)

Yield the very bottom lines of the help message.

If classes=False (the default), print *–help-all* msg.

emit_options_help()

Yield the lines for the options part of the help.

emit_subcommands_help()

Yield the lines for the subcommand part of the help.

initialize(*argv=None*)
Invoked after __init__() by *make_cmd()* to apply configs and build subapps.

Parameters **argv** – If undefined, they are replaced with `sys.argv[1:]`!

It parses cl-args before file-configs, to detect sub-commands and update any `config_paths`, then it reads all file-configs, and then re-apply cmd-line configs as overrides (trick copied from *jupyter-core*).

initialize_subcommand(*subc*, *argv=None*)
Initialize a subcommand with argv.

classmethod instance(**args*, ***kwargs*)
Returns a global instance of this class.

This method create a new instance if none have previously been created and returns a previously created instance is one already exists.

The arguments and keyword arguments passed to this method are passed on to the `__init__()` method of the class upon instantiation.

Create a singleton class using instance, and retrieve it:

```
>>> from traitlets.config.configurable import SingletonConfigurable
>>> class Foo(SingletonConfigurable): pass
>>> foo = Foo.instance()
>>> foo == Foo.instance()
True
```

Create a subclass that is retrived using the base class instance:

```
>>> class Bar(SingletonConfigurable): pass
>>> class Bam(Bar): pass
>>> bam = Bam.instance()
>>> bam == Bar.instance()
True
```

classmethod make_cmd(*argv=None*, ***kwargs*)
Instanciate, initialize and return application.

Parameters **argv** – Like *initialize()*, if undefined, replaced with `sys.argv[1:]`.

- Tip: Apply `pump_cmd()` on return values to process generators of *run()*.
- This functions is the 1st half of `launch_instance()` which invokes and discards *start()* results.

my_cmd_chain()
Return the chain of cmd-classes starting from my self or subapp.

read_config_files(*config_paths=None*)
Load `config_paths` and maintain `config_registry`.

Parameters **config_paths** – optional paths to override those in *config_paths* trait, in descending order (1st overrides the rest).

Returns the static_config loaded

- Configuration files are read and merged from `.json` and/or `.py` files in `config_paths`.
- Adapted from `load_config_file()` & `_load_config_files()` but without applying configs on the app, just returning them.

run(**args*)
Leaf sub-commands must inherit this instead of *start()* without invoking `super()`.

Parameters **args** – Invoked by *start()* with `extra_args`.

By default, screams about using sub-cmds, or about doing nothing!

start()
Dispatches into sub-cmds (if any), and then delegates to *run()*.

If overriden, better invoke `super()`, but even better to override *run()*.

exception polyvers.cmdlet.cmdlets.**CmdException**

class polyvers.cmdlet.cmdlets.**CmdletsInterpolation**(**args*, ***kw*)
Adds *cmdlets_map* into interp-manager for for help & cmd mechanics.

Client-code may add more dicts in *interpolation_context.maps* list.

class polyvers.cmdlet.cmdlets.**Forceable**
Mixin to facilitate "forcing" actions by ignoring/delaying their errors.

errlogged(**exceptions*, *token: Union[bool, str] = None*, *doing=None*, *raise_immediately=None*, *warn_log: Callable = None*, *info_log: Callable = None*)
A context-man for nesting `ErrLog` instances.

- See `ErrLog` for other params.
- The pre-existing *errlog* is searched in `_current_errlog` attribute.
- The *_current_errlog* on entering context, is restored on exit; original is *None*.
- The returned errlog always has its `ErrLog.parent` set to this enforceable.
- Example of using this method for multiple actions in a loop:

```
with self.errlogged(OSError,
                    doing="loading X-files",
                    token='fread'):
    for fpath in file_paths:
        with self.errlogged(doing="reading '%s'" % fpath):
            fbytes.append(fpath.read_bytes())

# Any errors collected above, will be raised/WARNed here.
```

is_forced(*token: Union[str, bool] = True*)
Whether some action ided by *token* is allowed to go thorugh in case of errors.

Parameters **token** – an optional string/bool to search for in `force` according to the following table:

```
                 token:
                       |NONE  |
                       |FALSE|TRUE|"str"|
force-element:         |-----|----|-----|
           [], FALSE|   X   |  X |   X   |
                TRUE|   X   |  O |   X   |
                  '*'|   X   |  X |   O   |
                "str"|   X   |  X |   =   |
```

- Rows above, win; columns to the left win.
- To catch all tokens, use `--force=true, --force='*'`

Tip: prefer using it via `ErrLog` contextman.

class polyvers.cmdlet.cmdlets.**PathList**(**args*, ***kwargs*)
Trait that splits unicode strings on *os.pathsep* to form a the list of paths.

validate(*obj*, *value*)
break all elements also into *os.pathsep* segments

class polyvers.cmdlet.cmdlets.**Printable**
A `HasTraits` mixin making `str()` return `class(trait=value, ...)` from traits.

Which traits to print are decided is decided by `TraitSelector`, for `printable_traits` class-property and `printable=True` tag.

class polyvers.cmdlet.cmdlets.**Replaceable**
A mixin to make `HasTraits` instances clone like namedtupple's `replace()`.

Parameters **changes** – a dict of values keyed be their trait-name.

Works nicely with *read-only* traits.

class polyvers.cmdlet.cmdlets.**Spec**(***kwargs*)
Common properties for all configurables.

classmethod class_get_trait_help(*trait*, *inst=None*, *helptext=None*)
Get the helptext string for a single trait.

Parameters

- **inst** – If given, it's current trait values will be used in place of the class default.
- **helptext** – If not given, uses the *help* attribute of the current trait.

ikeys(**maps*, ***kwds*) → ContextManager[polyvers.cmdlet.cmdlets.CmdletsInterpolation]
Temporarily place self before the given maps and kwds in interpolation-cntxt.

- Self has the least priority, kwds the most.
- For params, see `interp.InterpolationContext.interp()`.

Attention: Must use `str.format_map()` when *_stub_keys* is true; otherwise, `format()` will clone all existing keys in a static map.

interp(*text: Optional[str], *maps, **kwds*) → Optional[str]
Interpolate text with self attributes before maps and kwds given.

Parameters **text** – the text to interplate; None/empty returned as is

- For params, see `interp.InterpolationContext.interp()`.
- Self has the least priority, kwds the most.

polyvers.cmdlet.cmdlets.**build_sub_cmds**(**subapp_classes*)
Builds an ordered-dictionary of `cmd-name --> (cmd-class, help-msg)`.

polyvers.cmdlet.cmdlets.**chain_cmds**(*app_classes*, *argv=None*, ***root_kwds*)
Instantiate a list of `[cmd, subcmd, ...]`, linking children to parents.

Parameters

- **app_classes** – A list of cmd-classes: `[root, sub1, sub2, app]` Note: you have to "know" the correct nesting-order of the commands ;-)
- **argv** – cmdline args are passed to all cmds; make sure they do not contain any sub-cmds, or chain will be broken. Like `initialize()`, if undefined, replaced with `sys.argv[1:]`.

Returns The root(1st) cmd to invoke `Aplication.start()`

Apply the `pump_cmd()` or *collect_cmd()* on the return instance.

- Normally *argv* contain any sub-commands, and it is enough to invoke `initialize(argv)` on the root cmd. This function shortcuts arg-parsing for subcmds with explict cmd-chaining in code.
- This functions is the 1st half of `Cmd.launch_instance()`.

polyvers.cmdlet.cmdlets.**class_config_yaml**(*cls*, *outer_cfg*, *classes=None*, *config: polyvers._vendor.traitlets.config.loader.Config = None*)

Get the config section for this Configurable.

Parameters

- **classes** (*list*) – (optional) The list of other classes in the config file, used to reduce redundant help descriptions. If given, only params from these classes reported.
- **config** – If given, only what is contained there is included in generated yaml, with help-descriptions from classes, only where class default-values differ from the values contained in this dictionary.

polyvers.cmdlet.cmdlets.**class_help_description_lines**(*app_class*)

"Note: Reverse doc/description order bc classes do not have dynamic default `_desc()`, below.

polyvers.cmdlet.cmdlets.**cmd_line_chain**(*cmd*)

Utility returning the cmd-line(str) that launched a *Cmd*.

polyvers.cmdlet.cmdlets.**cmdlets_interpolations = CmdletsInterpolation({}, {'now': <polyver**

That's the singleton interp-manager used by all cmdlet configurables.

polyvers.cmdlet.cmdlets.**generate_config_file_yaml**(*self*, *classes=None*, *config: polyvers._vendor.traitlets.config.loader.Config = None*)

generate default config file from Configurables

Parameters **config** – If given, only what is contained there is included in generated yaml, with help-descriptions from classes, only where class default-values differ from the values contained in this dictionary.

5.3.3 Module: polyvers.cmdlet.errlog

Suppress or ignore exceptions collected in a nested contexts.

To get a "nested" *ErrLog* instance either use *nesterrlog()*, or call `ErrLog.__call__()` on the enclosing one.

FIXME: possible to have different node-hierarchies in contextvars-nesting!! (unless *ErrLog()* constructor is never called)

exception polyvers.cmdlet.errlog.**CollectedErrors**

class polyvers.cmdlet.errlog.**ErrLog**(*parent: polyvers.cmdlet.cmdlets.Forceable*, **exceptions*, *token: Union[bool, str, None] = None*, *doing=None*, *raise_immediately=None*, *warn_log: Callable = None*, *info_log: Callable = None*)

Collects errors in "stacked" contexts and delays or ignores ("forces") them.

Note: To nest errlogs, prefer *nesterrlog()* instead of this constructor, or else you must keep a reference on the last enclosing errlog and explicitly call `ErrLog.__call__()` on it.

- Unknown errors (not in *exceptions*) always bubble up immediately.
- Any "forced" errors are collected and logged in *warn_log* on context-exit, forcing is enabled when the *token* given exists in *spec*'s `force` attribute.
- Non-"forced" errors are either *raise_immediately*, or raised collectively in a *CollectedErrors*, on context-exit.
- Collected are always logged on DEBUG immediately.
- Instances of this class are callable, and the call will return a *clone* with provided properties updated.
- A *clone* is also returned when acquiring the context in a *with* statement.

Variables

- **spec** – the spec instance to search in its `Spec.force` for the token
- **exceptions** – the exceptions to delay or forced; others are left to bubble immediately If none given, `Exception` is assumed.
- **token** – the `force` token to respect, like `Spec.is_forced()`. Resets on each new instance from `__call__()`. Possible values:
  - **false: (default) completely ignore *force* trait** collected are just delayed);
  - <a string>: "force" if this token is in *force* trait;
  - *True*: "force" if *True* is in `force``force`.
- **doing** – A description of the running activity for the current stacked-context, in present continuous tense, e.g. "readind X-files".

  Resets on each new instance from `__call__()`. Assuming *doing = "having fun"*, it may generate one of those 3 error messages:

```
Failed having fun due to: EX
...and collected 21 errors (3 ignored):
    - Err 1: ...

Collected 21 errors (3 ignored) while having fun:
    - Err 1: ...

LOG: Ignored 9 errors while having fun:
    - Err 1: ...
```

- **raise_immediately** – if not forced, do not wait for *report()* call to raise them; suggested use when a function decorator. Also when –debug. Resets on each new instance from `__call__()`.

- **warn_log** – the logging method to report forced errors; if none given, use searched the log ov the *parent* or falls back to this's modules log. Note that all failures are always reported immediately on DEBUG.
- **info_log** – the logging method to report completed "doing" tasks; if none given does not report them.

PRIVATE FIELDS:

Variables

- **_root_node** – The root of the tree of nodes, populated when entering contexts recursively.
- **_anchor** – the parent node in the tree where on `__enter__()` a new *_active* child-node is attached, and the tree grows.
- **_active** – the node created in *_anchor*

Example of using this method for multiple actions in a loop:

```
with ErrLog(enforceable, OSError,
            doing="loading X-files",
            token='fread') as erl1:
    for fpath in file_paths:
        with erl1(doing="reading '%s'" % fpath) as erl2:
            fbytes.append(fpath.read_bytes())

# Any errors collected will raise/WARN here (root-context exit).
```

exception ErrLogException
A pass-through for critical ErrLog, e.g. context re-enter.

coords
Return my anchor's coordinate-indices from the root of the tree.

is_armed
Is context `__enter__()` currently under process?

is_forced
Try *force* in *parent* first.

is_good
An errlog is "good" if its anchor has not captured any exception yet.

If it does, it cannot be used anymore.

pdebug
Search *debug* property in *parent*, or False.

plog
Search *log* property in *parent*, or use this module's logger.

report_root(*ex_raised: Optional[Exception]*) → Optional[polyvers.cmdlet.errlog.CollectedErrors]
Raise or log the errors collected from

Parameters **ex_raised** – the cntxt-body exception `__exit__()` is about to raise

Returns a *CollectedErrors* in case catured errors contain non-forced errors BUT *ex_raised* given.

Raises ***CollectedErrors*** – any non-forced exceptions captured in tree (if any), unless *ex_raised* given

polyvers.cmdlet.errlog.**errlogged**(**errlog_args*, ***errlog_kw*)
Decorate functions/methods with a *ErrLog* instance.

The errlog-contextman is attached on the wrapped function/method as the *errlog* attribute.

polyvers.cmdlet.errlog.**nesterrlog**(*parent*, **exceptions*, *token: Union[bool, str] = None*, *doing=None*, *raise_immediately=None*, *warn_log: Callable = None*, *info_log: Callable = None*)
To nest errlogs, prefer this function instead of this *ErrLog()* constructor, or else you must keep a reference on the last enclosing errlog and explicitly call `ErrLog.__call__()` on it.

5.3.4 Module: polyvers.cmdlet.interpctxt

Enable Unicode-trait to pep3101-interpolate *{key}* patterns from "context" dicts.

class polyvers.cmdlet.interpctxt.**InterpolationContext**
A stack of 4 dics to be used as interpolation context.

The 3 stacked dicts are:

0. user-map: writes affect permanently this dict only;
1. time: ('now', 'utcnow'), always updated on access;
2. env-vars, *$*-prefixed.

Append more dicts in `self.maps` list if you wish.

ikeys(**maps*, *_stub_keys: Union[str, bool, None] = False*, *_escaped_for: Union[Callable, str] = None*, ***kv_pairs*) → ContextManager[InterpolationContext]
Temporarily place maps and kwds immediately after user-map (2nd position).

- For params, see *interp()*.

> **Attention:** Must use `str.format_map()` when *_stub_keys* is true; otherwise, `format()` will clone all existing keys in a static map.

interp(*text: Optional[str], *maps, _stub_keys=False, _escaped_for: Union[Callable, str] = None, _suppress_errors: bool = None, **kv_pairs*) → Optional[str]
Interpolate text with values from maps and kwds given.

Parameters

- **text** – the text to interpolate; if null/empty, returned as is
- **maps** – a list of dictionaries/objects/HasTraits from which to draw items/attributes/trait-values, all in increasing priority. Nulls ignored.
- **_stub_keys** –
  - If false, missing keys raise KeyError.
  - If *True*, any missing *key* gets replaced by `{key}` (practically remain unchanged).
  - If callable, the *key* is passed to it as a the only arg, and the result gets replaced.
  - Any other non-false value is returned for every *key*.
- **_suppress_errors** – ignore any interpolation errors and return original string
- **_escaped_for** – a callable or ('glob'|'regex') to escape object's attribute values

Later maps take precedence over earlier ones; *kv_pairs* have the highest, but *_stub_keys* the lowest (if true).

polyvers.cmdlet.interpctxt.**dictize_object**(*obj*, *_escaped_for: Union[Callable, str] = None*)
Make an object appear as a dict for *InterpolationContext.ikeys()*.

Parameters **_escaped_for** – one of 'glob', 'regex' or a callable to escape object's attribute values

5.4 Utilities

5.4.1 Module: polyvers.utils.mainpump

Utils pumping results out of yielding functions for *main()*.

class polyvers.utils.mainpump.**ConsumerBase**
Checks if all boolean items (if any) are True, to decide final bool state.

class polyvers.utils.mainpump.**ListConsumer**
Collect all items in a list, while checking if all boolean ok.

class polyvers.utils.mainpump.**PrintConsumer**
Prints any text-items while checking if all boolean ok.

polyvers.utils.mainpump.**collect_cmd**(*cmd_res*, *dont_coalesce=False*, *assert_ok=False*)
Pumps cmd-result in a new list.

Parameters

- **cmd_res** – A list of items returned by a `Cmd.start()`/*Cmd.run()*. If it is a sole item, it is returned alone without a list.
- **assert_ok** – if true, checks *ListConsumer*'s exit-code is not false.

polyvers.utils.mainpump.**pump_cmd**(*cmd_res*, *consumer=None*)
Sends (possibly lazy) cmd-results to a consumer (by default to STDOUT).

Parameters

- **cmd_res** – Whatever is returnened by a `Cmd.start()`/*Cmd.run()*.
- **consumer** – A callable consuming items and deciding if everything was ok; defaults to *PrintConsumer*

Returns `bool(consumer)`

- Remember to have logging setup properly before invoking this.
- This the 2nd half of the replacement for `Application.launch_instance()`.

5.4.2 Module: polyvers.utils.logconfutils

Utils for configuring and using elaborate logs and handling *main()* failures.

polyvers.utils.logconfutils.**exit_with_pride**(*reason=None*, *warn_color='\x1b[31;1m'*, *err_color='\x1b[1m'*, *logger=None*)
Return an *exit-code* and logs error/fatal message for `main()` methods.

Parameters

- **reason** –
  - If reason is None, exit-code(0) signifying OK;
  - if exception, print colorful (if tty) stack-trace, and exit-code(-1);
  - otherwise, prints str(reason) colorfully (if tty) and exit-code(1),
- **warn_color** – ansi color sequence for stack-trace (default: red)
- **err_color** – ansi color sequence for stack-trace (default: white-on-red)
- **logger** – Which logger to use to log reason (must support info and fatal). if missing, derived from this module.

Returns (0, 1 -1), for reason == (None, str, Exception) respectively.

Note that returned string from `main()` are printed to stderr and exit-code set to bool(str) = 1, so print stderr separately and then set the exit-code.

For colors use `RainbowLoggingHandler.getColor()`, defaults: - '[33;1m': yellow+bold - '[31;1m': red+bold

Note: it's better to have initialized logging.

`polyvers.utils.logconfutils.`**`init_logging`**(*level=20*, *logconf=None*, *color=None*, *logger=None*, ***kwds*)

Parameters

- **level** – Root-logger's level; Overrides *logconf* if given, INFO otherwise.
- **logconf** (*None, str, seq[str]*) – File(s) to configure loggers; set *[]* to prohibit loading any logconf file. Allowed file-extensions:
  - '.conf' (implied if missing) .
  - '.yml'/'yaml'

  The ~ in the path expanded to $HOME. See https://docs.python.org/3/library/logging.config.html
- **color** – Whether to color log-messages; if undefined, true only in consoles.
- **logger** – Which logger to use to log logconf source(must support info and debug). if missing, derived from this module.
- **kwds** – Passed directly to `logging.basicConfig()` (e.g. *filename*); used only id default HOME `logconf.yaml` file is NOT read.

`polyvers.utils.logconfutils.`**`log_level_from_argv`**(*args*, *start_level: int*, *eliminate_verbose=False*, *eliminate_quiet=False*, *verbosity_step=10*)

Parameters **start_level_index** – some existing level

5.4.3 Module: `polyvers.utils.oscmd`

Utility to call OS commands through `subprocess.run()` with logging.

The *polyvers* version-configuration tool is generating tags like:

```
proj-foo-v0.1.0
```

On purpose python code here kept with as few dependencies as possible.

class polyvers.utils.oscmd.**PopenCmd**(*dry_run=False*, *check_stdout=True*, *check_stderr=True*, *check_returncode=True*, ***popen_kw*)

A function –> cmd-line builder for executing (mostly) git commands.

To run `git log -n1`:

```
out = cmd.git.log(n=1)
```

To launch a short python program with `python -c "print('a')"`:

```
out = cmd.python._(c=True)('print('a')')
```

Raises **sbp.CalledProcessError** – if check_returncode=true and exit code is non-zero.

Important: Flags written out are mostly for Git, bc flags are produced like that:

```
-f <value> --flag=<value>
```

---

polyvers.utils.oscmd.**exec_cmd**(*cmd*, *dry_run=False*, *check_stdout=True*, *check_stderr=True*, *check_returncode=True*, *encoding='utf-8'*, *encoding_errors='surrogateescape'*, ***popen_kws*)

Parameters **check_stdout** – None: Popen(stdout=None), printed False: Popen(stdout=sbp.DEVNULL), ignored True: Popen(stdout=sbp.PIPE), collected & returned

5.4.4 Module: polyvers.utils.fileutil

Generic utils.

polyvers.utils.fileutil.**abspath**(*path*)

Like `osp.abspath()`, but preserving last slash.

polyvers.utils.fileutil.**convpath**(*fpath*, *abs_path=True*, *exp_user=True*, *exp_vars=True*)

Without any flags, just pass through `osp.normpath()`.

polyvers.utils.fileutil.**ensure_dir_exists**(*path*, *mode=493*)

ensure that a directory exists

If it doesn't exist, try to create it and protect against a race condition if another process is doing the same.

The default permissions are 755, which differ from os.makedirs default of 777.

polyvers.utils.fileutil.**ensure_file_ext**(*fname*, *ext*, **exts*, *is_regex=False*)

Ensure that the filepath ends with the extension(s) specified.

Parameters

- **ext** (*str*) – The 1st extension to search & to append if none matches, so must not be a regex.
- **exts** (*str*) – Other extensions. These may be regexes, depending on *is_regex*; a *'$'* is always added at its end.
- **is_regex** (*bool*) – When true, the rest *exts* are parsed as case-insensitive regexes.

Example:

```
>>> ensure_file_ext('foo', '.bar')
'foo.bar'
>>> ensure_file_ext('foo.', '.bar')
'foo.bar'
>>> ensure_file_ext('foo.', 'bar')
'foo.bar'

>>> ensure_file_ext('foo.BAR', '.bar')
'foo.BAR'
>>> ensure_file_ext('foo.DDD', '.bar')
'foo.DDD.bar'
```

Note that omitting dot('.') from extension does affect the results:

```
>>> ensure_file_ext('foo', 'bar')
'foo.bar'
>>> ensure_file_ext('foo.BAR', 'bar')
'foo.BAR'
>>> ensure_file_ext('fooBAR', 'bar')  # File allowed without extension!
'fooBAR'
```

When more extensions are given, the 1st is appended if none matches:

```
>>> ensure_file_ext('foo.xlt', '.xlsx', '.XLT')
'foo.xlt'
>>> ensure_file_ext('foo.xlt', '.xlsx', '.xltx')
'foo.xlt.xlsx'
```

And when regexes:

```
>>> ensure_file_ext('foo.xlt', '.xlsx', r'\.xl\w{1,2}', is_regex=True)
'foo.xlt'
>>> ensure_file_ext('foo.xl^', '.xls', r'\.xl\w{1,2}', is_regex=True)
'foo.xl^.xls'
```

polyvers.utils.fileutil.**find_git_root**(*path=None*) → Optional[pathlib.Path]

Search dirs up for a Git-repo like `git rev-parse --show-toplevel`.

Parameters **path** – where to start searching from, *cwd* if not given.

Returns a *pathlib* native path, or None

polyvers.utils.fileutil.**normpath**(*path*)

Like `osp.normpath()`, but preserving last slash.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

polyvers.cli, 35
polyvers.cmdlet.cfgcmd, 40
polyvers.cmdlet.cmdlets, 41
polyvers.cmdlet.errlog, 46
polyvers.cmdlet.interpctxt, 49
polyvers.engrave, 39
polyvers.pvproject, 36
polyvers.pvtags, 38
polyvers.utils.fileutil, 52
polyvers.utils.logconfutils, 50
polyvers.utils.mainpump, 50
polyvers.utils.oscmd, 51
polyvers.vermath, 39
polyversion, 29
polyversion.setuplugin, 32

Index

Symbols

<pname>_VERSION, 19

A

abspath() (in module polyvers.utils.fileutil), 52
add_versions() (in module polyvers.vermath), 40

B

bdist-check, **9**
bootstrapp_projects() (polyvers.cli.PolyversCmd method), 34, 35
build_sub_cmds() (in module polyvers.cmdlet.cmdlets), 45
bump2version, **26**
bumpversion, **26**

C

cast() (polyvers.vermath.Pep440Version method), 40
CfgFilesRegistry (class in polyvers.cmdlet.cmdlets), 42
chain_cmds() (in module polyvers.cmdlet.cmdlets), 45
changes, **26**
check_bdist_kw() (in module polyversion.setuplugin), 33
class_config_yaml() (in module polyvers.cmdlet.cmdlets), 46
class_get_trait_help() (polyvers.cmdlet.cmdlets.Spec class method), 45
class_help_description_lines() (in module polyvers.cmdlet.cmdlets), 46
clear_instance() (polyvers.cmdlet.cmdlets.Cmd class method), 42
Cmd (class in polyvers.cmdlet.cmdlets), 42
cmd_line_chain() (in module polyvers.cmdlet.cmdlets), 46
CmdException, 44
cmdlets_interpolations (in module polyvers.cmdlet.cmdlets), 46
CmdletsInterpolation (class in polyvers.cmdlet.cmdlets), 44
collect_app_infos() (polyvers.cli.PolyversCmd method), 34, 35
collect_cmd() (in module polyvers.utils.mainpump), 50
collect_fpaths() (polyvers.cmdlet.cmdlets.CfgFilesRegistry method), 42
collect_matches() (polyvers.pvproject.Graft method), 36
CollectedErrors, 46
config_tuples (polyvers.cmdlet.cmdlets.CfgFilesRegistry attribute), 42
ConfigCmd (class in polyvers.cmdlet.cfgcmd), 40
ConsumerBase (class in polyvers.utils.mainpump), 50
convpath() (in module polyvers.utils.fileutil), 52
coords (polyvers.cmdlet.errlog.ErrLog attribute), 48

D

decide_vprefixes() (in module polyversion), 32
default version env-var, **9**
DescCmd (class in polyvers.cmdlet.cfgcmd), 40
dictize_object() (in module polyvers.cmdlet.interpctxt), 50

E

emit_description() (polyvers.cmdlet.cmdlets.Cmd method), 42
emit_examples() (polyvers.cmdlet.cmdlets.Cmd method), 42
emit_help_epilogue() (polyvers.cmdlet.cmdlets.Cmd method), 42
emit_options_help() (polyvers.cmdlet.cmdlets.Cmd method), 42
emit_subcommands_help() (polyvers.cmdlet.cmdlets.Cmd method), 42
engrave, **9**
Engrave (class in polyvers.pvproject), 36
engravings, **9**
ensure_dir_exists() (in module polyvers.utils.fileutil), 52
ensure_file_ext() (in module polyvers.utils.fileutil), 52
environment variable
    <pname>_VERSION, 19

POLYVERSION_LOG_LEVEL, 18
ErrLog (class in polyvers.cmdlet.errlog), 46
ErrLog.ErrLogException, 48
errlogged() (in module polyvers.cmdlet.errlog), 48
errlogged() (polyvers.cmdlet.cmdlets.Forceable method), 44
exec_cmd() (in module polyvers.utils.oscmd), 52
exit_with_pride() (in module polyvers.utils.logconfutils), 50

F

FileProcessor (class in polyvers.engrave), 39
find_git_root() (in module polyvers.utils.fileutil), 53
Forceable (class in polyvers.cmdlet.cmdlets), 44

G

generate_config_file_yaml() (in module polyvers.cmdlet.cmdlets), 46
Git Bump, **26**
git_describe() (polyvers.pvproject.Project method), 36
git_project_errors_handled() (in module polyvers.pvtags), 38
git_restore_point() (in module polyvers.pvtags), 38
GitError, 38
GitVoidError, 38
glob_files() (in module polyvers.engrave), 39
Graft (class in polyvers.pvproject), 36

H

head_folder() (polyvers.cmdlet.cmdlets.CfgFilesRegistry method), 42

I

ikeys() (polyvers.cmdlet.cmdlets.Spec method), 45
ikeys() (polyvers.cmdlet.interpctxt.InterpolationContext method), 49
incremental, **26**
InfosCmd (class in polyvers.cmdlet.cfgcmd), 40
init_logging() (in module polyvers.utils.logconfutils), 51
init_plugin_kw() (in module polyversion.setuplugin), 32
InitCmd (class in polyvers.cli), 34, 35
initialize() (polyvers.cli.InitCmd method), 34, 35
initialize() (polyvers.cmdlet.cfgcmd.ShowCmd method), 41
initialize() (polyvers.cmdlet.cmdlets.Cmd method), 42
initialize_subcommand() (polyvers.cmdlet.cmdlets.Cmd method), 43
instance() (polyvers.cmdlet.cmdlets.Cmd class method), 43
interp() (polyvers.cmdlet.cmdlets.Spec method), 45
interp() (polyvers.cmdlet.interpctxt.InterpolationContext method), 49
InterpolationContext (class in polyvers.cmdlet.interpctxt), 49
is_armed (polyvers.cmdlet.errlog.ErrLog attribute), 48
is_forced (polyvers.cmdlet.errlog.ErrLog attribute), 48
is_forced() (polyvers.cmdlet.cmdlets.Forceable method), 44
is_good (polyvers.cmdlet.errlog.ErrLog attribute), 48
is_good() (polyvers.pvproject.Project method), 37

L

last_commit_tstamp() (polyvers.pvproject.Project method), 37
leaf commit, **8**
Lerna, **26**
ListConsumer (class in polyvers.utils.mainpump), 50
Lock release trains as "developmental", **9**
log_level_from_argv() (in module polyvers.utils.logconfutils), 51
LogconfCmd (class in polyvers.cli), 34, 35

M

make_cmd() (polyvers.cmdlet.cmdlets.Cmd class method), 43
make_match_all_pvtags_project() (in module polyvers.pvtags), 38
make_match_all_vtags_project() (in module polyvers.pvtags), 38
make_pvtag_project() (in module polyvers.pvtags), 38
make_vtag_project() (in module polyvers.pvtags), 39
Marking dependent versions across sub-projects, **9**
merge_dict() (in module polyvers.cli), 34, 36
mono-project, **8**
monorepo, **8**
my_cmd_chain() (polyvers.cmdlet.cmdlets.Cmd method), 43

N

nesterrlog() (in module polyvers.cmdlet.errlog), 49
NoGitRepoError, 38
normpath() (in module polyvers.utils.fileutil), 53

O

Other Features, **9**
out-of-trunk commit, **8**
overlapped_matches() (in module polyvers.engrave), 39

P

Pants, **26**
parse_command_line() (polyvers.cli.PolyversCmd method), 34, 35
PathList (class in polyvers.cmdlet.cmdlets), 45
pbr, **26**
pdebug (polyvers.cmdlet.errlog.ErrLog attribute), 48
PEP 440 version ids, **7**
Pep440Version (class in polyvers.vermath), 39

plog (polyvers.cmdlet.errlog.ErrLog attribute), 48
polytime() (in module polyversion), 31
polyvers.cli (module), 34, 35
polyvers.cmdlet.cfgcmd (module), 40
polyvers.cmdlet.cmdlets (module), 41
polyvers.cmdlet.errlog (module), 46
polyvers.cmdlet.interpctxt (module), 49
polyvers.engrave (module), 39
polyvers.pvproject (module), 36
polyvers.pvtags (module), 38
polyvers.utils.fileutil (module), 52
polyvers.utils.logconfutils (module), 50
polyvers.utils.mainpump (module), 50
polyvers.utils.oscmd (module), 51
polyvers.vermath (module), 39
PolyversCmd (class in polyvers.cli), 34, 35
polyversion (module), 29
polyversion() (in module polyversion), 30
polyversion.setuplugin (module), 32
POLYVERSION_LOG_LEVEL, 18
PopenCmd (class in polyvers.utils.oscmd), 51
populate_pvtags_history() (in module polyvers.pvtags), 39
prepare_search_map() (in module polyvers.cmdlet.cfgcmd), 41
Printable (class in polyvers.cmdlet.cmdlets), 45
PrintConsumer (class in polyvers.utils.mainpump), 50
Project (class in polyvers.pvproject), 36
pump_cmd() (in module polyvers.utils.mainpump), 50
pvtag_format (in module polyversion), 32
pvtags_history (polyvers.pvproject.Project attribute), 37
Python Enhancement Proposals
    PEP 0426, 30
    PEP 0440, 8
    PEP 0518, 4, 22
    PEP 3101, 30, 32, 38
python guide, **26**

R

r-tag, **8**
read_config_files() (polyvers.cmdlet.cmdlets.Cmd method), 43
release scheme, **8**
release tag, **8**
releash, **26**
Replaceable (class in polyvers.cmdlet.cmdlets), 45
repo scheme, **8**
report_root() (polyvers.cmdlet.errlog.ErrLog method), 48
RFC
    RFC 2822, 32, 37
run() (in module polyvers.cli), 35, 36
run() (polyvers.cli.InitCmd method), 34, 35
run() (polyvers.cli.LogconfCmd method), 34, 35
run() (polyvers.cli.StatusCmd method), 34, 35
run() (polyvers.cmdlet.cfgcmd.DescCmd method), 40
run() (polyvers.cmdlet.cfgcmd.InfosCmd method), 40
run() (polyvers.cmdlet.cfgcmd.ShowCmd method), 41
run() (polyvers.cmdlet.cmdlets.Cmd method), 43

S

set_new_version() (polyvers.pvproject.Project method), 37
setuptools, **9**
setuptools integration, **9**
setuptools plugin, **9**
setuptools_scm, **26**
ShowCmd (class in polyvers.cmdlet.cfgcmd), 40
Spec (class in polyvers.cmdlet.cmdlets), 45
start() (polyvers.cmdlet.cmdlets.Cmd method), 44
StatusCmd (class in polyvers.cli), 34, 35

T

tag_fnmatch() (polyvers.pvproject.Project method), 37
tag_regex() (polyvers.pvproject.Project method), 38
tag_version_commit() (polyvers.pvproject.Project method), 38

V

v-tag, **9**
validate() (polyvers.cmdlet.cmdlets.PathList method), 45
version scheme, **8**
version tag, **8**
version-bump algebra, **8**
version_from_pvtag() (polyvers.pvproject.Project method), 38
VersionError, 40
visit_file() (polyvers.cmdlet.cmdlets.CfgFilesRegistry method), 42
vtag_format (in module polyversion), 32
vtag_regex (in module polyversion), 32

Z

Zest.releaser, **26**