g-pypi Documentation

Release 0.3

Domen Kožar

January 20, 2014

Author Domen Kožar <domen@dev.si>

Source code Github.com source browser

Bug tracker Github.com issues

Generated January 20, 2014

Version 0.3

Features

- write ebuilds to overlay or stdout (formatted in ansi color, html and bbcode)
- install ebuilds through portage
- use *MY_P*, *MY_PN*, *MY_PV* when needed using Bash substitutions
- extract metadata from *PyPi* like *HOMEPAGE*, *DESCRIPTION*, *LICENSE*, *AUTHOR*, etc.
- determine *RDEPEND* / *DEPEND* from `setuptools`: `install_requires`, `tests_require`, `setup_requires` and `extras_require`
- determine *PYTHON_MODNAME* from `setuptools`: *packages*, *py_modules* and *package_dir*
- determine *S* by unpacking ebuild
- discovers Sphinx documentation
- discovers examples
- discovers `nosetests` and `setup.py test`
- generates ebuilds for dependencies
- uses Portage-alike colorful output
- offers *customizable configuration*
- support for `python setup.py sdist_ebuild`
- updates `Manifest` file
- generates `metadata.xml` file
- appends `ChangeLog` file

Overview

gpypi is a command line tool for creating Gentoo portage ebuilds from Python Package Index.
gpypi was started as part of Google Summer of Code 2010 by *Domen Kožar*, mentored by *Jesus Rivero*.
gpypi is not meant to be a Gentoo developer replacement. On the contrary, it's goal is to make his (hers) life easier.

CHAPTER 1

User Guide

1.1 Installation

1.1.1 Stable version

The most recommended way of installing **gpypi** is through portage. Portage will handle all of dependencies and make sure you install stable version.

```
# sudo emerge -av gpypi
```

Download and install directly through *PyPi*:

```
$ sudo easy_install gpypi
```

or:

```
$ sudo pip install gpypi
```

1.1.2 Development version

```
$ sudo pip install https://github.com/iElectric/g-pypi/zipball/master
```

or:

```
$ sudo easy_install https://github.com/iElectric/g-pypi/zipball/master
```

1.2 Getting started

gpypi is a command line tool with vision to make Gentoo developers life easier. To create an ebuild and its dependencies:

```
$ sudo gpypi create --overlay sunrise pylons
* Generating ebuild: Jinja2 2.5
* Your ebuild is here: /usr/local/portage/dev-python/jinja2/jinja2-2.5.ebuild
* Dependency needed: Babel
* Generating ebuild: Babel 0.9.5
* Your ebuild is here: /usr/local/portage/dev-python/babel/babel-0.9.5.ebuild
* Dependency needed: pytz
```

```
* Generating ebuild: pytz 2010h
* Your ebuild is here: /usr/local/portage/dev-python/pytz/pytz-2010h.ebuild
```

Warning: Root login must be used for populating overlays and unpacking ebuilds.

Usage should be pretty self explanatory through help:

```
$ sudo gpypi -h
usage: gpypi [-h] [-v] {create,sync,install,echo} ...

Builds ebuilds from PyPi.

optional arguments:
  -h, --help            show this help message and exit
  -v, --version

commands:
  {create,sync,install,echo}
    create              Write ebuild and it's dependencies to an overlay
    echo                Echo ebuild to stdout
    install             Install ebuild and it's dependencies
    sync                Populate all packages from pypi into an overlay
```

and most of the time one will use the **gpypi create** command:

```
$ sudo gpypi create -h
usage: gpypi create [-h] [-P PN] [-V PV] [--MY-PV MY_PV] [--MY-PN MY_PN]
                    [--MY-P MY_P] [--homepage HOMEPAGE] [--keywords KEYWORDS]
                    [--license LICENSE] [--description DESCRIPTION]
                    [--long-description LONG_DESCRIPTION] [-u URI]
                    [-i INDEX_URL] [--nocolors] [--config-file CONFIG_FILE]
                    [-q | -d] [-l OVERLAY_NAME] [-o] [--no-deps]
                    [-c CATEGORY] [--metadata-disable]
                    [--metadata-disable-echangelog-user]
                    [--metadata-herd METADATA_HERD]
                    [--metadata-maintainer-description METADATA_MAINTAINER_DESCRIPTION]
                    [--metadata-maintainer-email METADATA_MAINTAINER_EMAIL]
                    [--metadata-maintainer-name METADATA_MAINTAINER_NAME]
                    [--echangelog-disable]
                    [--echangelog-message ECHANGELOG_MESSAGE]
                    [--repoman-commands REPOMAN_COMMANDS]
                    package name [package version]

Write ebuild and it's dependencies to an overlay

positional arguments:
  package name
  package version

optional arguments:
  -h, --help            show this help message and exit
  -P PN, --PN PN        Specify PN to use when naming ebuild
  -V PV, --PV PV        Specify PV to use when naming ebuild
  --MY-PV MY_PV         Specify MY_PV used in ebuild
  --MY-PN MY_PN         Specify MY_PN used in ebuild
  --MY-P MY_P           Specify MY_P used in ebuild
  --homepage HOMEPAGE   Homepage of the package
  --keywords KEYWORDS   Portage keywords for ebuild masking
```

```
  --license LICENSE     Portage license for the ebuild
  --description DESCRIPTION
                        Short description of the package
  --long-description LONG_DESCRIPTION
                        Long description of the package
  -u URI, --uri URI     Specify SRC_URI of the package
  -i INDEX_URL, --index-url INDEX_URL
                        Base URL for PyPi
  --nocolors            Disable colorful output
  --config-file CONFIG_FILE
                        Absolute path to a config file
  -q, --quiet           Show less output.
  -d, --debug           Show debug information.
  -l OVERLAY_NAME, --overlay OVERLAY_NAME
                        Specify overlay to use by name (stored in
                        $OVERLAY/profiles/repo_name)
  -o, --overwrite       Overwrite existing ebuild
  --no-deps             Don't create ebuilds for any needed dependencies
  -c CATEGORY, --category CATEGORY
                        Specify portage category to use when creating ebuild

Workflow control:
  Generate metadata, manifest, changelog ...

  --metadata-disable    Disable metadata generation
  --metadata-disable-echangelog-user
                        Don't use ECHANGELOG_USER
  --metadata-herd METADATA_HERD
                        Herd for ebuild metadata
  --metadata-maintainer-description METADATA_MAINTAINER_DESCRIPTION
                        Maintainer descriptions for ebuild metadata (comma
                        separated)
  --metadata-maintainer-email METADATA_MAINTAINER_EMAIL
                        Maintainer emails for ebuild metadata (comma
                        separated)
  --metadata-maintainer-name METADATA_MAINTAINER_NAME
                        Maintainer names for ebuild metadata (comma separated)
  --echangelog-disable  Disable echangelog
  --echangelog-message ECHANGELOG_MESSAGE
                        Echangelog commit message
  --repoman-commands REPOMAN_COMMANDS
                        List of repoman commands to issue on each ebuild
                        (separated by space)
```

1.3 Creating ebuild from source of Python package with distutils

`gpypi` supports not also querying *PyPi* but also creating an ebuild with help of distutils. Configuration is done when you first run `gpypi`. `cd` to your package and just do:

```
python setup.py sdist_ebuild
```

1.4 Configuration

`gpypi` offers configuration based on multiple sources. Currently supported sources are: `Config.from_pypi()`,

`Config.from_setup_py()`, `Config.from_argparse()` and `Config.from_ini()`.

Configuration API lets you choose what source is used and what priority it has relative to other source providers. Here is a complete list of supported configuration options that `Config` can provide:

```
# 'config_name': ("doc", "type", "default_value"),
'up_pn': ('Upstream package name', str, ""),
'up_pv': ('Upstream package version', str, ""),
'pn': ('Specify PN to use when naming ebuild', str, ""),
'pv': ('Specify PV to use when naming ebuild', str, ""),
# TODO: move my_* stuff into config, make [] as default and make sure it handles lists from
'my_pv': ('Specify MY_PV used in ebuild', str, ""),
'my_pn': ('Specify MY_PN used in ebuild', str, ""),
'my_p': ('Specify MY_P used in ebuild', str, ""),
'uri': ('Specify SRC_URI of the package', str, ""),
'index_url': ('Base URL for PyPi', str, "http://pypi.python.org/pypi"),
'overlay': ('Specify overlay to use by name (stored in $OVERLAY/profiles/repo_name)', str, "
'overwrite': ('Overwrite existing ebuild', bool, False),
'no_deps': ("Don't create ebuilds for any needed dependencies", bool, False),
'category': ("Specify portage category to use when creating ebuild", str, ""),
'format': ("Format when printing to stdout (use pygments identifier)", str, "none"),
'command': ("Name of command that was invoked on CLI", str, ""),
'nocolors': ("Disable colorful output", bool, False),
'background': ("Background of terminal when using formatting", str, 'dark'),
#'pretend': ("Print ebuild to stdout, don't write ebuild file, don't download SRC_URI", bool,
'license': ("Portage license for the ebuild", str, ""),
'keywords': ("PyPI keywords", str, ""),
# TODO: homepage will be a list
'homepage': ("Homepage of the package", str, ""),
'description': ("Short description of the package", str, ""),
'long_description': ("Long description of the package", str, ""),
'gentoo_keywords': ("Portage keywords for ebuild masking", str, "~x86"),
# metadata
'metadata_disable': ("Disable metadata generation", bool, False),
'metadata_use_echangelog_user': ("Use ECHANGELOG_USER", bool, False),
'metadata_herd': ("Herd for ebuild metadata", str, ""),
'metadata_maintainer_description': ("Maintainer descriptions for ebuild metadata (comma sepa
'metadata_maintainer_email': ("Maintainer emails for ebuild metadata (comma separated)", str,
'metadata_maintainer_name': ("Maintainer names for ebuild metadata (comma separated)", str, '
# echangelog
'echangelog_disable': ("Disable echangelog", bool, False),
'echangelog_message': ("Echangelog commit message", str, "Initial ebuild generated by g-pypi'
# repoman
'repoman_commands': ("List of repoman commands to issue on each ebuild (separated by space)",
```

`Config` is basically a *dict* with few additional classmethods for validation and source processing. Each `Config` represents configuration values retrieved from specific source.

`ConfigManager` is a class that handles multiple `Config` instances. When a value is retrieved from `ConfigManager`, it is loaded from `Config` instances located in `ConfigManager.configs` *(dict)*. Order is specified as `use` parameter to `ConfigManager`.

When `gpypi` is first time used, it will create `.ini` configuration file at `/etc/gpypi`. Further usage will load the file with `ConfigManager.load_from_ini()`. Default configuration file will look something like this:

```
[config]
# main option defaults go here:
# overlay = Personal
# ...
```

```
[config_manager]
# list the order of configurations
use = argparse ini pypi setup_py
# list of what options will invoke interactive questions when missing
questionnaire_options = overlay
```

You will notice the `use` parameter in `config_manager` section. As already said, it specifies what `Config` sources are used and in what order. `config_manager` section is loaded on `ConfigManager.load_from_ini()` call, creating the `ConfigManager` instance.

`config` section is used as `ini` source provider, populated by `Config.from_ini` also called in `ConfigManager.load_from_ini`. Another non-foobared example of configuration file:

```
[config]
format = html
overlay = iElectric
index_url = http://eggs.mycompany.com

[config_manager]
use = pypi ini argparse
questionnaire_options = uri category
```

The last option not yet mentioned is `questionnaire_options`. The question is, what happens when none of `Config` sources provide the config value we need? The behavior is specified with `questionnaire_options`. If configuration option is listed in `questionnaire_options`, `Questionnaire` is used to interactively request developer for input through shell. Otherwise, default is used (specified in `Config.allowed_options` tuple).

Most of `ConfigManager.configs` are populated in `gpypi.cli` module.

Note: For example usage of classes, following linked API definition.

CHAPTER 2

Development

2.1 Workflow of creating an ebuild

1. PyPi is queried for an package with coresponding version (if no version is given, highest available is used)
2. PyPi metadata is collected, and `gpypi.enamer.Enamer.get_vars()` is used to collect common ebuild variables
3. Initial ebuild is written to overlay with *SRC_URI*
4. Ebuild in unpacked with Portage API through shell
5. Unpacked dir is inspected for setup.py information
6. Ebuild is rendered again and written to an overlay
7. Possible dependencies from setup.py are resolved and whole process is repeated for each one.

2.2 How are *PV*, *PN*, *MY_PV*, *MY_PN* and *SRC_URI* determined?

All the work is done by `gpypi.enamer.Enamer.get_vars()`. Specifically:

- *PV* and *MY_PV* in `gpypi.enamer.Enamer.parse_pv()`
- *PN* and *MY_PN* in `gpypi.enamer.Enamer.parse_pn()`
- *SRC_URI* and *HOMEPAGE* in `gpypi.enamer.SrcUriNamer`

2.3 Tests against live PyPi – `gpypi.tests.test_pypi`

This module runs numerous tests against whole PyPI. It should be run manually, to detect new possible issues. All failures MUST be first written as unittests and then fixed accordingly.

Important: Issues should not be closed until there are appropriate tests and documentation for the changeset.

2.4 TODO

List of features that may be implemented in no particular order:

- atomic actions (cleanup on traceback/error)
- migrate to simpleindex and xmlrpc API provided by distutils2
- issue HEAD request to homepages (warn on failure)
- finish SrcUriNamer implementation
- implement setuptools Feature class
- SVN/GIT/HG support as SRC_URI
- implement homepage/src_uri as list (also includes config work)
- decide what to do with configuartion on dependency ebuilds
- search command
- use rewriting system instead of regex for pn/pv parsing

CHAPTER 3

Python API

3.1 gpypi – Main package

3.1.1 gpypi.sdist_ebuild – Distutils command

class gpypi.sdist_ebuild.**sdist_ebuild**(*dist*)
Bases: distutils.cmd.Command

argparse_config = {'overwrite': True}

default_format = {'nt': '', 'posix': 'ebuild'}

description = 'create an ebuild file for Gentoo Linux'

finalize_options()

initialize_options()

path_to_distutils_conf = '/home/docs/checkouts/readthedocs.org/user_builds/g-pypi/envs/latest/lib/python2.7/dist

classmethod register()
Writes gpypi project into distutils commmand_packages settings.

run()

user_options = [('config-file=', 'c', 'GPyPi configuration file [default: /etc/gpypi]'), ('dist-dir=', 'd', 'directory to put t

3.1.2 gpypi.cli – Command line handling

3.1.3 gpypi.config – Configuration handling

3.1.4 gpypi.ebuild – Ebuild generation module

3.1.5 gpypi.enamer – Utilities for metadata conversion

3.1.6 gpypi.exc – gpypi specific Exceptions

Exceptions module

exception gpypi.exc.**GPyPiConfigurationError**
Bases: gpypi.exc.GPyPiException

exception gpypi.exc.**GPyPiCouldNotCreateEbuildPath**
Bases: gpypi.exc.GPyPiException

Raised when directory for an ebuild could not be created.

exception gpypi.exc.**GPyPiCouldNotUnpackEbuild**
Bases: gpypi.exc.GPyPiException

Raised if unpacking failed.

exception gpypi.exc.**GPyPiException**
Bases: exceptions.Exception

Core exception class, all exception inherit from this class.

exception gpypi.exc.**GPyPiInvalidAtom**
Bases: gpypi.exc.GPyPiException

Raised when determining Portage Atom did not succeed.

exception gpypi.exc.**GPyPiInvalidParameter**
Bases: gpypi.exc.GPyPiException

Raised CLI parameter is not valid.

exception gpypi.exc.**GPyPiNoDistribution**
Bases: gpypi.exc.GPyPiException

Raised if unpacked directory could not be found.

exception gpypi.exc.**GPyPiNoSetupFile**
Bases: gpypi.exc.GPyPiException

Raised if no setup.py was found.

exception gpypi.exc.**GPyPiOverlayDoesNotExist**
Bases: gpypi.exc.GPyPiException

exception gpypi.exc.**GPyPiValidationError**
Bases: gpypi.exc.GPyPiException

3.1.7 gpypi.portage_utils – Portage utilities

3.1.8 gpypi.workflow – Generate manifest, metadata, changelog ...

3.2 gpypi.tests – Unittests package

3.2.1 gpypi.tests.test_sdist_ebuild

3.2.2 gpypi.tests.test_cli

3.2.3 gpypi.tests.test_config

3.2.4 gpypi.tests.test_ebuild

3.2.5 gpypi.tests.test_enamer

3.2.6 gpypi.tests.test_portage_utils

3.2.7 gpypi.tests.test_pypi

3.2.8 gpypi.tests.test_workflow

CHAPTER 4

Changelog

4.1 0.4 (2014/01/17)

- Switch to EAPI=5, distutils-r1 [Noel Burton-Krahn <noel@pistoncloud.com>]

4.2 0.3.1 (2012/xx/xx)

- Switch to EAPI=4 [Domen Kožar]
- Use mirror://pypi/ instead of http link [Domen Kožar]

4.3 0.3 (2012/06/28)

- initial release after GSOC2010 work [Domen Kožar]

CHAPTER 5

Glossary

PN Package name in Gentoo syntax. Example: jinja2

MY_PN Package name used in ebuilds converted from *PN* to match upstream package name.

PV Package version in Gentoo syntax. Example: 0.1_beta3

MY_PV Package version used in ebuilds converted from *PV* to match upstream package version.

MY_P Package name and version in Gentoo syntax. Example: jinja2-0.1_beta3

SRC_URI URI string from where source code is downloaded.

HOMEPAGE Homepage address of a package.

LICENSE License which software is copyrighted under

DEPEND List of dependencies needed for compiling/setup

RDEPEND List of runtime dependencies

S Working directory while portage is emerging an ebuild

PYTHON_MODNAME List of python module names in this ebuild

PyPi Python Package Index (CPAN alike repository for packages) — http://pypi.python.org/pypi

DESCRIPTION Portage variable, short description of an ebuild program

AUTHOR Person who initially wrote the program

eclass Bash script provided by portage for handling ebuilds

CHAPTER 6

Indices and tables

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

g