invirtualenv Documentation

Release 0.0.0

Yahoo Inc.

Feb 01, 2019

Contents

The invirtualenv package contains scripts for deploying applications written in Python inside Python virtualenv.

These included scripts can:

- Create native packages via plugins that contain a Python application installed inside. Currently there are plugins to create rpms and docker containers.
- Create a specified operating system platform using an RPM package list.
- Create a Python virtualenv, when running as root the virtualenv can be deployed so that it can be manipulated by another user.
- Deploy Python packages into the virtualenv from a pip requirements file.

The script can be configured using command line arguments or all configuration can be defined in a single deploy.conf file.

Contents:

CHAPTER 1

Install

The invirtualenv utilities are part of a python package which can be installed using pip.

1.1 Quickstart

Install the package using the python pip package utility:

```
pip install -U invirtualenv
```

1.2 Platform

Some of the scripts provided by the invirtualenv package have enhanced functionality depending on the platform they are run on.

On systems that support rpm, the following rpm packages are required to utilize the rpm package support in the *create_package* script.:

```
rpm-build
```

The Docker container build requires the docker command has been installed and working.

CHAPTER 2

Usage

The *deploy_virtualenv* and *create_package* scripts have a number of common use cases.

CHAPTER 3

Scripts

Invirtualenv command line utilities.

3.1 invirtualenv

The *invirtualenv* command line tool is used to perform invirtualenv operations. This script is meant to replace the *create_package* and *deploy_virtualenv* scripts that are still provided for backwards compatibility.

3.1.1 invirtualenv usage

The invirtualenv command takes a number of subcommands used to specify the invirtualenv function to perform:

```
usage: invirtualenv [-h] [--deploy_conf DEPLOY_CONF]
                    {list_plugins,create_package_config,create_package,get_setting}
                    ...

optional arguments:
  -h, --help            show this help message and exit
  --deploy_conf DEPLOY_CONF
                        Deploy configuration filename or url (default:
                        deploy.conf)

command:
  {list_plugins,create_package_config,create_package,get_setting}
    list_plugins        List the installed invirtualenv plugins
    create_package_config
                        Generate the packaging configuration file
    create_package      Generate a package from a deployment configuration
    get_setting         Get a setting value from the configuration
```

3.2 deploy_virtualenv

This script will create a new python virtual environment based on a *deploy.conf - Deployment configuration file* or *requirements.txt* file. The script will default to looking for a *deploy.conf - Deployment configuration file* file in the current directory for it's configuration settings.

3.2.1 Usage

The *deploy_virtualenv* script has the following usage:

```
usage: deploy_virtualenv [-h] [--python PYTHON] [--requirement REQUIREMENT]
                         [--virtualenvdir VIRTUALENVDIR]
                         [--virtualenvuser VIRTUALENVUSER]
                         [--virtualenvgroup VIRTUALENVGROUP]
                         [--virtualenvversion_package VIRTUALENVVERSION_PACKAGE]
                         [--install_os_packages INSTALL_OS_PACKAGES]
                         [name]

Deploy a python application into a virtualenv

positional arguments:
  name                  VirtualEnv name (default: public_mirror)

optional arguments:
  -h, --help            show this help message and exit
  --python PYTHON, -p PYTHON
                        The Python interpreter to use, e.g.,
                        --python=python3.5 will use the python3.5 interpreter
                        to create the new environment. The default is the
                        interpreter that virtualenv was installed with.
                        (default: python2.7)
  --requirement REQUIREMENT, -r REQUIREMENT
                        Install from the given requirements file. This option
                        can be used multiple times. (default: [])
  --virtualenvdir VIRTUALENVDIR
                        Directory to build the virtualenv (default:
                        /var/tmp/virtualenv)
  --virtualenvuser VIRTUALENVUSER
                        The user to create the virtualenv (default: dhubbard)
  --virtualenvgroup VIRTUALENVGROUP
                        The group to create the virtualenv (default: )
  --virtualenvversion_package VIRTUALENVVERSION_PACKAGE
                        Version the virtualenv based on the version of a
                        package (default: public_mirror)
  --install_os_packages INSTALL_OS_PACKAGES
                        Install OS packages (default: False)
```

3.2.2 Examples

Deploying using deploy.conf

The following example uses a *deploy.conf - Deployment configuration file* file to do the following:

- **Creates a new python virtualenv with the following characteristics:**

- It is created in the */var/tmp/virtualenv* directory
- Uses a *python2.7* python interpreter
- It has a base name of *public_mirror*
- It has a version appended to the name based on the latest version of the *public_mirror* package in the python repo
- The virtualenv is owned by unix user *pypimirror*

The example uses the following *deploy.conf - Deployment configuration file* file:

```
[global]
;#####################################################################
; Global settings
;#####################################################################
; The name of the virtualenv to create
name = public_mirror

; The python interpreter to use for the virtualenv
; this will default to the python interpreter running the virtualenv
; command if it is not specified.
basepython = python2.7

; Base directory to create virtualenv in
virtualenv_dir = /var/tmp/virtualenv

; Use the version of a python package to determine the version component
; of the virtualenv.
; If no versions is found or specified the virtualenv will not have a
; version component in the name.
virtualenv_version_package = public_mirror

; The user that should own the virtualenv.
virtualenv_user = pypimirror

; When package manifest(s) to install into the virtualenv
; If none are specified all manifest will be deployed.
; Note:
;     It is generally a bad idea to use a deb and rpm manifest together.
install_manifest = pip, rpm

[pip]
;#####################################################################
; PIP package settings
;#####################################################################
; deps contains a list of python packages to install.
; It is recommended this be a concrete list such as what is returned
; using the 'pip freeze' command.
; Each line must be indented.
deps:
    astroid==1.4.4
    colorama==0.3.6
    eventlet==0.18.2
    future==0.14.3
    greenlet==0.4.9
    IPy==0.83
    keyring==8.4
    lazy-object-proxy==1.2.1
```

(continues on next page)

(continued from previous page)

```
    mccabe==0.4.0
    pkginfo==1.2.1
    pluggy==0.3.1
    py==1.4.31
    pycrypto==2.6.1
    pylint==1.5.4
    PyYAML==3.11
    requests==2.9.1
    six==1.10.0
    wrapt==1.10.6

[rpm]
;########################################################################
; rpm package settings
;########################################################################
deps:
    libcrypto-dev
```

The resulting output from running the *deploy_virtualenv* command in the same directory as the *deploy.conf - Deployment configuration file* is:

```
# deploy_virtualenv

**********************************************************************
Parsing the configuration
**********************************************************************

**********************************************************************
Getting version based on package 'public_mirror' from the repo
**********************************************************************
Using version: 0.0.13

**********************************************************************
Installing rpm packages
**********************************************************************
libcrypto

**********************************************************************
Building virtualenv
**********************************************************************
You are using pip version 7.1.2, however version 8.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
New python executable in /var/tmp/virtualenv/public_mirror_0.0.13/bin/python2.7
Also creating executable in /var/tmp/virtualenv/public_mirror_0.0.13/bin/python
Installing setuptools, pip, wheel...done.
Creating /var/tmp/virtualenv/public_mirror_0.0.13/conf directory
Creating /var/tmp/virtualenv/public_mirror_0.0.13/logs directory

**********************************************************************
Installing python package dependencies
**********************************************************************
Installing requirements from requirements file: /tmp/tmphBEO0g into virtualenv /var/
↪tmp/virtualenv/public_mirror_0.0.13 as user None
Current user is: root
Current uid: 0, Effective uid: 0
```

(continues on next page)

(continued from previous page)

```
**********************************************************************
Fixing file ownership
**********************************************************************
```

This is the virtualenv that got created by the last command:

```
# ls -lh /var/tmp/virtualenv/
total 0
drwxrwxr-x 1 pypimirror pypimirror 88 Feb 19 00:38 public_mirror_0.0.13
[root@6b7d38db3855 dhubbard]#
```

Creating a virtualenv using CLI arguments

The following example creates a new python virtualenv with the following characteristics:

- It is created in the /tmp directory
- It has a base name of invirtualenv
- It has a version appended to the virtualenv based on the latest version of the *invirtualenv* python package (1.1.62)
- The virtualenv is owned by the unix user dhubbard

The following requirements.txt file is used:

```
cov-core==1.15.0
coverage==4.0.1
future==0.15.2
nose==1.3.7
nose-cov==1.6
requests==2.8.1
virtualenv==13.1.2
wheel==0.24.0
```

This is what this example looks like:

```
airreport-lm:invirtualenv dhubbard$ deploy_virtualenv.py --virtualenvdir /tmp --
↪virtualenvversion_package invirtualenv --virtualenvuser dhubbard -r requirements.
↪txt invirtualenv
**********************************************************************
Building virtualenv
**********************************************************************
You are using pip version 7.0.3, however version 8.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Using real prefix '/Library/Frameworks/Python.framework/Versions/2.7'
New python executable in invirtualenv_1.1.62/bin/python
Installing setuptools, pip, wheel...done.
Creating /tmp/invirtualenv_1.1.62/conf directory
Creating /tmp/invirtualenv_1.1.62/logs directory

**********************************************************************
Installing python package dependencies
**********************************************************************
Installing requirements from requirements file: ['requirements.txt'] into virtualenv /
↪tmp/invirtualenv_1.1.62 as user None
Current user is: dhubbard
Current uid: 58157, Effective uid: 58157
```

(continues on next page)

(continued from previous page)

```
The directory '~/.cache' or its parent directory is not owned by the current user and␣
↪caching wheels has been disabled. check the permissions and owner of that directory.
↪ If executing pip with sudo, you may want sudo's -H flag.

**********************************************************************
Fixing file ownership
**********************************************************************
```

The resulting virtualenv directory has been created, owned by the specified user:

```
drwxrwxr-x  10 dhubbard  wheel  340 Feb 11 14:58 /tmp/invirtualenv_1.1.62
```

3.3 create_package

The *create_package* script creates packages of various types that contain the python application deployed within a virtualenv.

3.3.1 Usage

The *create_package* script has the following usage:

```
usage: create_package [-h] [--package_type {rpm, tar}]

optional arguments:
  -h, --help            show this help message and exit
  --package_type {rpm}
                        Type of package to create
```

CHAPTER 4

Configuration Settings

The *deploy_virtualenv* script can use two different configuration files: *deploy.conf - Deployment configuration file* and *requirements.txt* these files have different functionality and usage.

4.1 deploy.conf - Deployment configuration file

The *deploy.conf - Deployment configuration file* file allows for setting all configuration arguments in a single file. This allows running the script without any other command line arguments.

The same file can be used for deployment and generation of deployment packages in multiple packaging formats. Current supported deployment packaging formats are: rpm and docker containers.

The *deploy.conf - Deployment configuration file* file is a Python `ConfigParse` format configuration file. This file consists of sections which are deliminated by square brackets. Inside each section are key/value fields. Multi-Line values can be defined by indenting subsequent lines.

The *deploy.conf - Deployment configuration file* file has at least 3 possible sections, all of which are optional: global, pip, rpm.

The *deploy.conf - Deployment configuration file* supports jinja2 template format substitutions for values of environment variables. This allows setting values based on environment variable values. I.E. setting the version based on the BUILD_NUMBER environment variable provided by the CI pipeline for example.

4.1.1 global settings

The *global settings* section of the *deploy.conf - Deployment configuration file* defines a number of settings. All of which are optional. These settings act as the default values if they are not set in other sections.

name

The *name* setting is used to define the basename of the virtualenv to be created.

basepython

The *basepython* setting is used to define the python interpreter inside the virtualenv. If this setting is not specified it will default to the python interpreter that runs the virtualenv command.

install_manifest

The *install_manifest* setting specifies a comma seperated list of manifests to install. If this setting is not set all package manifests (lists of packages) that will be installed.

** NOTE: It is generally not a good idea to mix multiple types of platform package manifests. Such as using both rpm and tar **

install_os_packages

The *install_os_packages* setting causes the script to attempt to install the rpm package resources needed to install Python sdist format packages.

virtualenv_dir

The *virtualenv_dir* setting specifies the directory that will hold the virtualenv that is created. This directory needs to be writable by the user running the script.

version

This setting will set the version string the the value specified.

The example below the version string is specified to be based on the BUILD_NUMBER environment variable, defaulting to '0.0.0' if the BUILD_NUMBER is not set. In the example below if the BUILD_NUMBER was set to 15 the version would be '0.0.15':

```
version = 0.0.{{BUILD_NUMBER|default('0')}}
```

virtualenv_version_package

This setting will lookup the most recent version of the package specified and append that version number to the end of the virtualenv name. This allows versioning the virtualenvs based on a specific package version.

For example, if the most recent version of the public_mirror package is 0.39.0 and the virtualenv was created with the following settings:

```
name = mirror
virtualenv_version_package = public_mirror
```

The virtualenv will be named ** mirror_0.39.0 **

virtualenv_user

This setting specifies the unix username that should own the created virtualenv.

This setting requires the following:

- This script is being run by a user that has the privileges to change the

ownership to the user specified (generally root) * The user specified has been created on the system.

virtualenv_group

This setting specifies the unix group of the created virtualenv.

This setting requires the following:

- This script is being run by a user that has the privileges to change the group to the user specified (generally root)
- The group specified has been created on the system.

4.1.2 pip package manifest

This configuration section allows defining settings related to the installation of python packages that are installed using the *pip* tool. As part of a pip package manifest.

This section is used to define the python packages to install.

deps

The deps section contains a list of python packages to install. The format for the deps is the same as the format of a *pip* requirements file.

4.1.3 rpm package manifest

The [rpm] configuration section allows defining package manifests (list of packages) of rpm packages installed using the yum tool.

These is section defines a package manifest of rpm packages to install prior to installing the python packages.

Note:

> rpm package installations are not atomic and once installed some package dependencies can block uninstall or upgrade of certain packages. As a result a rpm install failure can leave the system in a different package state than when the script run started.

deps

This section is a list of rpm packages to install.

4.1.4 docker_container packaging (creation)

The [docker_container] section contains the settings used to create a docker container containing the python application in a virtualenv. This sectiion allows for adding settings used to create the docker container.

The docker plugin will generate basic sane container based on the defaults and values in the global section of the configuration.

This section can be used to set docker specific settings such as commands to use for healthchecks of the created docker container.

add

base_image

default=ubuntu:17.10

container_name

copy

deb_deps

A manifest of debian packages to install into the container prior to creating the python virtualenv.

entrypoint

env

expose

files

healthcheck

This is the healthcheck script to run in the container.

label

A list of labels to be applied to the container.

rpm_deps

A manifest of rpm packages to install into the container prior to creating the python virtualenv.

run_before

A list of shell commands to run before creating the python virtualenv inside the container.

run_after

A list of shell commands to run after creating the python virtualenv inside the container.

setenv

Environment variables to set when creating the container.

stopsignal

user

volume

A list of volume mappings to use with the container.

4.1.5 rpm packaging

The [rpm_package] section defines settings related to the creation of rpm packages. The settings in this section will overide the default values from the global settings for rpm package generation only.

deps

This is a manifest (list of packages) to be listed as dependencies of the created rpm package.

Note:

```
This should include the packages that contain the python interpreter
and virtualenv commands to use to deploy the virtualenv when the created package
is installed.
```

4.1.6 Example deploy.conf

```
; Copyright (c) 2016, Yahoo Inc.
; Copyrights licensed under the BSD License
; See the accompanying LICENSE.txt file for terms.

[global]
;#########################################################################
; Global settings
;#########################################################################

; The name of the virtualenv to create
name = public_mirror

; The python interpreter to use for the virtualenv
; this will default to the python interpreter running the virtualenv
; command if it is not specified.
basepython = python3

; Base directory to create virtualenv in
virtualenv_dir = /var/tmp/virtualenv

;#########################################################################
; Version settings
;#########################################################################
; There are several ways to specify a version for the virtualenv.
;
; The first way is to specify the version directly.  This file allows
; using environment variables using :ref:`jinja2` template syntax.
; In the example below the last digit of the version will be either the
```

(continues on next page)

(continued from previous page)

```
; value of the BUILD_NUMBER environment variable or 0 if it is not set.
; This method makes it easy to generate a unique version number in a CI
; pipeline that sets the BUILD_NUMBER environment variable.
version = 0.0.{{BUILD_NUMBER|default('0')}}

; The second method to specify a version is to use the latest version
; of a python package in the Python package repository to determine the
; version component of the virtualenv.
; If no versions is found or specified the virtualenv will not have a
; version component in the name.
; virtualenv_version_package = public_mirror

; The user and group that should own the virtualenv.  If not specified
; the current user will be used.
virtualenv_user =
virtualenv_group =


; When package manifest to install into the virtualenv
; If none are specified all manifest will be deployed.
; Note:
;     It is generally a bad idea to use tar and rpm manifest together.
install_manifest = pip, rpm

; If the user has root privileges setting this to True will
; attempt to install the python devel package resources using
; rpm.  This is here for backwards compatibility, it is
; recommended to specify the devel packages in the rpm
; package manifest instead.
install_os_packages = False

[pip]
;#####################################################################
; PIP package settings
;#####################################################################
; deps contains a list of python packages to install.
; It is recommended this be a concrete list such as what is returned
; using the 'pip freeze' command.
; Each line must be indented.
deps:
    public_mirror==0.0.39

[rpm]
;#####################################################################
; rpm package install settings
;#####################################################################
; deps contains a list of rpm packages to install.
; These dependencies are applied as a single rpm operation with
; the packages ordered as specified.
;
; Keep in mind rpm that rpm operations are not atomic so using
; a large number of packages with specific versions numbers can
; cause dependency conflicts with already installeed packages and
; leave the system in a state that cannot easily be reverted.
deps:

[rpm_package]
```

(continues on next page)

(continued from previous page)

```
;#####################################################################
; rpm package creation settings
;#####################################################################
# Deps listed here are added as rpm package dependencies of the created
# package.  This should at minimum include the python interpreter used
# to run the script contained in the package.
deps:
    python3
# These settings allow specifying these values in the spec file used to
# generate the rpm package.
; group=
; license=
; packager=


[docker_container]
;#####################################################################
; docker container creation settings
;#####################################################################
base_image=ubuntu:17.10
container_name=invirtualenvapp/publicmirror
; entrypoint=/bin/bash
; expose=
files:
setenv:
    DEBIAN_FRONTEND=noninteractive
```

4.2 requirements.txt

The *requirements.txt* file is a pip format *requirements.txt* file and only specifies the python packages to be installed in the virtualenv.

CHAPTER 5

Python invirtualenv module code documentation

Python module support functions for the invirtualenv scripts

InVirtualEnv Python Virtualenv Installer Module

5.1 Configuration Manipulation

Functions for handling the configuration settings

`invirtualenv.config.`**`cast_types`**(*configuration*, *types_dict=None*)
Update in place the configuration dictionary with inferred values if they aren't specified.

Parameters

- **`configuration`** (*`dict`*) –
- **`config_types`** (*`dict, optional`*) – Types dictionary, defaults to value from config_types()

`invirtualenv.config.`**`generate_parsed_config_file`**(*source=None*, *dest=None*)
Generate a conf file with the environment variables expanded

Parameters

- **`source`** (*`str, optional`*) – The path to the source file
- **`dest`** (*`str, optional`*) – The path to the desired destination file

Returns Path to the generated config file

Return type str

`invirtualenv.config.`**`get_configuration`**(*configuration=None*)
Parse a configuration file

Parameters **`configuration`** (*`str or list, optional`*) – A configuration file or list of configuration files to parse, defaults to the deploy_default.conf file in the package and deploy.conf in the current working directory.

Returns The parsed configuration

Return type configparser

invirtualenv.config.**get_configuration_dict**(*configuration=None*, *value_types=None*)
Parse the configuration files

Parameters

- **configuration** (*str or list, optional*) – A configuration file or list of configuration files to parse, defaults to the deploy_default.conf file in the package and deploy.conf in the current working directory.
- **value_types** (*dict, optional*) – Dictionary containing classes to apply to specific items

Returns Configuration dictionary

Return type dict

invirtualenv.config.**parse_arguments**(*configuration=None*)
Parse the command line arguments

Parameters **configuration** (*list, optional*) – A list of config files to parse, defaults to the packaged deploy_default.conf and the deploy.conf file in the current directory.

Returns With the configuration based on the passed command line arguments and defaults from the deploy_default.conf file in the package.

Return type argparse namespace object

5.2 Deployment

Module to create/deploy a virtualenv

invirtualenv.deploy.**build_deploy_virtualenv**(*arguments=None*, *configuration=None*, *update_existing=True*, *verbose=None*)
Build and deploy a python virtualenv

Parameters

- **arguments** (*argparse namespace, optional*) – An argparse namespace with all of the required command line arguments defaults to parsing the command line arguments if not provided.
- **configuration** (*str or list, optional*) – A configuration file or list of configuration files to parse, defaults to the deploy_default.conf file in the package and deploy.conf in the current working directory.
- **update_existing** (*bool*) – If True, allow updating an existing virtualenv, Otherwise generate an exception. Default=True
- **verbose** (*bool*) – If True, provides status output while running.

Raises

- `AlreadyExists` – The virtualenv already exists
- `BuildError` – The specified package manifest cannot be deployed to the virtualenv
- `InsufficientPermissions` – The current user lacks permissions to complete the specified operation

- `NoPackageVersions` – A virtualenv_version_package was specified in the configuration but no versions for that package where found on artifactory.

`invirtualenv.deploy.`**`deployed_bin_files`**(*venv*)
Gets files that where deployed to the bin directory of the virtualenv.

Parameters **venv** (*str*) – Path the to virtualenv to do this bin_file links for

Returns Key = filename Value = sha256 hash

Return type dict

`invirtualenv.deploy.`**`fix_file_ownership`**(*virtualenv*, *user*, *group*)
Fix the file ownership of a virtualenv

Parameters

- **virtualenv** (*str*) – Virtualenv path
- **user** (*str*) – User to change to
- **group** (*str*) – Group to change to

:param : :type : return:

`invirtualenv.deploy.`**`install_python_dependencies`**(*virtualenv*, *deps=None*, *requirements=None*, *upgrade=False*, *verbose=False*, *pip_version=None*, *use_index=True*)
Install python dependencies from a requirements file or deploy.conf manifest

Parameters

- **virtualenv** (*str*) – The virtualenv to install into
- **deps** (*list, optional*) – A list of python packages to install
- **requirements** (*str, optional*) – The requirements.txt file to install from
- **upgrade** (*bool, optional*) – Tell pip to upgrade packages when installing, Default=False
- **verbose** (*bool, optional*) – Display command output when running the dependency operations, Default=False
- **use_index** (*bool, optional*) – Allow pip to use an external index Default=True

Raises BuildException - If package installation fails

`invirtualenv.deploy.`**`install_rpm_dependencies`**(*deps=None*, *fail_missing=True*)
Install rpm dependencies from deploy.conf manifest

Parameters

- **deps** (*list, optional*) – A list of rpm packages to install
- **fail_missing** (*bool, optional*) – Generate an exception and fail if the "yum" command is not available. default = True

Raises BuildException - If package installation fails

`invirtualenv.deploy.`**`link_deployed_bin_files`**(*venv*, *destbin*)
Link the deployed files from the venv bin directory into the destbin directory.

Writes a json list of the created links to conf/created_links.json in the virtualenv.

Parameters

- **venv** (*str*) – Path to the python virtualenv
- **destbin** (*str*) – Destination directory when the links should be

Returns Full path to files linked

Return type list

invirtualenv.deploy.**unlink_deployed_bin_files**(*venv*)
Undo/delete the symlinks created by link_deployed_bin_files()

Parameters **venv** (*str*) – Path to the root of the virtualenv

5.3 Exceptions

Exceptions generated by the invirtualenv module

exception invirtualenv.exceptions.**AlreadyExists**
Package already Exists exception

exception invirtualenv.exceptions.**BuildException**
There was an error Building the VirtualEnv

exception invirtualenv.exceptions.**CommandNotFound**
The command specified wasn't found

exception invirtualenv.exceptions.**InsufficientPermissions**
The current user has insufficient permissions to complete the specified operation.

exception invirtualenv.exceptions.**InvirtualenvError**
An invirtualenv error occurred.

exception invirtualenv.exceptions.**NoPackageVersions**
Query of the Python PyPi repo returned no packages with the specified name.

exception invirtualenv.exceptions.**PackageConfigFailure**
Unable to create a package configuration file

exception invirtualenv.exceptions.**PackageGenerationFailure**
Package generation failed

5.4 Package Manipulation

Functions for managing packaging

invirtualenv.package.**install_prereq_packages**(*test=False*)
Install packages required to build python

Parameters **test** (*bool, optional*) – Don't run the actual command, if True

invirtualenv.package.**latest_package_version**(*package*)
Get the latest version number for a package

Parameters **package** (*str*) – Package to get the latest version of

Returns Latest package version or an empty string if no versions are found

Return type str

invirtualenv.package.**package_scripts_directory**()
Get the package scripts directory

Returns str

The path to the directory containing the package scripts

invirtualenv.package.**package_versions**(*package*, *pypi_url=None*)
Get all versions of a package from pypi

Parameters

- **package** (*str*) – The python package to search for
- **pypi_url** (*str, optional*) – The pypi URL to send the request to, defaults to the production pypi endpoint.

Returns A list of all packages found on pypi. The list will be empty if there were no versions found.

Return type list

invirtualenv.package.**strip_from_end**(*text*, *suffix*)
Strip a substring from the end of a string

Parameters

- **text** (*str*) – The string to be evaluated
- **suffix** (*str*) – The suffix or substring to remove from the end of the text string

Returns A string with the substring removed if it was found at the end of the string.

Return type str

5.5 Utility

General utility functionality module

invirtualenv.utility.**change_uid_gid**(*user_uid=None*, *user_gid=None*)
preexec_fn to change the uid/gid when using subprocess

Parameters

- **user_uid** (*int, optional*) – The user UID to set to, does not set user uid if this is not passed
- **user_gid** (*int, optional*) – The user GID to set to, does not set the gid if this is not passed

invirtualenv.utility.**chown_recursive**(*path*, *uid*, *gid*)
Change the ownership of all files and directories in path

Parameters

- **path** (*str*) – The root path to start changing the ownership at
- **uid,** (*int*) – The uid to change the ownership to
- **gid,** (*int*) – The gid to change the ownership to

invirtualenv.utility.**csv_list**(*value*)
Convert a comma separated string into a list

Parameters **value** (*str*) – The string object to convert to a list

Returns A list based on splitting the string on the ',' character

Return type list

invirtualenv.utility.**display_header**(*text=''*, *width=None*, *separator=None*, *outfile=None*, *collapse=False*)

Display a textual header message. :param text: The text to print/display :type text: str :param width: The width (text wrap) of the header message.

This will be the current terminal width as determined by the 'stty size' shell command if not specified.

Parameters

- **separator** (*str, optional*) – The character or string to use as a horizontal separator. Will use '=' if one is not specified.
- **outfile** (*File, optional*) – The File object to print the header text to. This will use sys.stdout if not specified.

:param : :type : return:

invirtualenv.utility.**get_terminal_size**()

Get the terminal rows and columns if we are running on an interactive terminal. :returns: * **rows** (*int*) – The number of rows on the current terminal.

- **columns** (*int*) – The number of columns on the current terminal.

invirtualenv.utility.**str_format_env**(*value*)

Substitute values with environment variables in a string

Parameters **value** (*str*) – The string object to be formatted and converted into a list

Returns With env variable values substituded

Return type str

invirtualenv.utility.**str_to_bool**(*value*)

Convert a string too a bool object

Parameters **value** (*str*) – The string object to convert to a bool. The following case insensitive strings evaluate to True ['true', 1', 'up', 'on']

Returns Boolean based on the string

Return type bool

invirtualenv.utility.**str_to_dict**(*value*)

Convert a newline terminated string of key=value into a dictionary.

Parameters **value** (*str*) – The string object to convert to a dictionary

Returns Dictionary based on the string

Return type dict

invirtualenv.utility.**str_to_list**(*value*)

Convert a newline terminated string into a list. Any empty lines will be removed from the result list.

Parameters **value** (*str*) – The string object to convert to a list

Returns List based on the string

Return type list

invirtualenv.utility.**update_recursive**(*basedict*, *updatedict*)

Recursively run update on a dictionary

Parameters

- **basedict** (*dict*) – First dictionary, which will contain the updated result
- **updatedict** (*dict*) – Second dictionary that will be be updated to basedict, conflicts will be replaced with the values from this dictionary.

Returns basedict updated to contain the values from updatedict

Return type dict

invirtualenv.utility.**update_recursive_generator**(*basedict*, *updatedict*)
Recursively run update on a dictionary

Parameters

- **basedict** (*dict*) – First dictionary, which will contain the updated result
- **updatedict** (*dict*) – Second dictionary that will be be updated to basedict, conflicts will be replaced with the values from this dictionary.

Returns basedict updated to contain the values from updatedict

Return type dict

invirtualenv.utility.**which**(*command*)
Function searches the path for the command executable

Parameters **command** (*str*) –

Returns Path to the command

Return type str

Raises CommandNotFound: – Command was not found

5.6 Static Standalone Binary Creation

5.7 Virtualenv Management

Functions for creating and managing python virtual environments

invirtualenv.virtualenv.**build_virtualenv**(*name*, *directory*, *python_interpreter=None*, *user=None*, *verbose=False*)
Build a virtualenv in a directory

Parameters

- **name** (*str*) – Name of the virtualenv to create
- **directory** (*str*) – Directory to create the virtualenv in
- **python_interpreter** (*str, optional*) – Python interpreter to provide in the virtualenv, defaults to the interpreter that is running the virtualenv command
- **verbose** (*bool*) – If True, provides status output while running.

Returns Full path to the root of the virtualenv directory

Return type str

Raises BuildException – The Virtualenv build failed

invirtualenv.virtualenv.**default_virtualenv_directory**()
Get the default virtualenv directory for the current system/platform

Returns The path to the default virtualenv directory

Return type str

invirtualenv.virtualenv.**install_requirements**(*requirements*, *virtualenv*, *user=None*, *upgrade=False*, *verbose=False*, *pip_version=None*, *use_index=True*)
Open one or more requirements files and run pip -r to install them

Parameters

- **requirements** (*str*) – Filename containing requirements
- **virtualenv** (*str*) – Full path to the virtualenv to install into
- **user** (*str, optional*) – The user:group to run the install as
- **upgrade** (*bool*) – If True, tell pip to upgrade when running the install. Default=False
- **verbose** (*bool*) – If True, provides status output while running.
- **pip_version** (*str, optional*) – Install the requirements with the specified version of pip
- **use_index** (*bool, optional*) – Allow pip to use an external index Default=True

invirtualenv.virtualenv.**remove_virtualenv**(*name*, *directory=None*)
Remove a virtualenv from a directory :param name: :param directory: :return:

invirtualenv.virtualenv.**upgrade_package_tools**(*virtualenv_directory*, *verbose=False*)
Upgrade the packages used to install/build packages in the virtualenv :param virtualenv_directory: The directory that contains the virtualenv :type virtualenv_directory: str

invirtualenv.virtualenv.**virtualenv_bin_file_hashes**(*virtualenv_dir*)
Calculate a hash of all files in the virtualenv bin directory :param virtualenv_dir: The root directory of the virtualenv to operate on :type virtualenv_dir: str

Returns Key = Filename, value = hash

Return type dict

invirtualenv.virtualenv.**virtualenv_command**(*install_virtualenv=False*)
Return the virtualenv command

Parameters **install_virtualenv** (*bool*) – If True, Install virtualenv if necessary. default=False

Returns Path to the virtualenv command

Return type str

CHAPTER 6

invirtualenv_plugins package

6.1 Subpackages

6.1.1 invirtualenv_plugins.rpm_scripts package

Submodules

invirtualenv_plugins.rpm_scripts.post_install module

invirtualenv_plugins.rpm_scripts.post_install.**get_config_flag**(*section*, *option*, *config_filename=None*)

invirtualenv_plugins.rpm_scripts.post_install.**update_config**(*venv_dir*, *config_filename=None*)

invirtualenv_plugins.rpm_scripts.pre_uninstall module

6.2 Submodules

6.2.1 invirtualenv_plugins.docker module

class invirtualenv_plugins.docker.**InvirtualenvDocker**(*config_file='deploy.conf'*)

Bases: invirtualenv.plugin_base.InvirtualenvPlugin

config_default = '[docker_container]\nadd=\nbase_image=ubuntu:17.10\ncmd=\ncontainer_n

config_types = {'docker_container': {'files': <type 'list'>, 'volume': <type 'list'

default_config_filename = 'Dockerfile.invirtualenv'

generate_wheel_archive(*filename=None*)

generate_wheel_packages(*wheeldir*)
Generate wheel packages for all dependencies

Parameters **wheeldir** (*str*) – The directory path to store the generated wheel packages

Returns

Return type dict of filename, pip requirements line

package_formats = ['docker']

package_template = 'FROM {{docker_container[\'base_image\']|default(\'ubuntu:17.10\')}

run_package_command(*package_hashes*, *wheel_dir='wheels'*)
Run the command to generate the package based on the hash

system_requirements_ok()
Check if all the system requirements for this plugin are met.

Returns True if requirements are met, False otherwise

Return type bool

write_command_scripts()

6.2.2 invirtualenv_plugins.parsedconfig module

class invirtualenv_plugins.parsedconfig.**InvirtualenvParsedConfig**(**args*, ***kwargs*)
Bases: invirtualenv.plugin_base.InvirtualenvPlugin

default_config_filename = 'deploy.conf.parsed'

package_formats = ['parsed_deploy_conf']

package_template = None

run_package_command(*package_hashes*, *wheel_dir='wheels'*)
Run the command to generate the package based on the hash

6.2.3 invirtualenv_plugins.rpm module

class invirtualenv_plugins.rpm.**InvirtualenvRPM**(*config_file='deploy.conf'*)
Bases: invirtualenv.plugin_base.InvirtualenvPlugin

config_default = '[rpm_package]\ndeps:\n'

config_types = {'rpm_package': {'deps': <type 'list'>}}

default_config_filename = 'invirtualenv.spec'

package_formats = ['rpm']

package_template = 'Summary: {{global[\'description\']|default(\'No summary available

run_package_command(*package_hashes*, *wheel_dir='wheels'*)
Run the command to generate the package based on the hash

system_requirements_ok()
Check if all the system requirements for this plugin are met.

Returns True if requirements are met, False otherwise

Return type bool

Python Module Index

i

Index

L

N

P

R

S

U

V

W