venv-management

Release 2.2.0

Sixty North

Aug 08, 2023

CONTENTS:

CHAPTER

ONE

INSTALLATION

Install from PyPI using `pip`:

```
$ pip install venv-management
```

CHAPTER

TWO

CONFIGURATION

2.1 Shell selection

This `venv-management` package delegates most operations to one of the `virtualenvwrapper` or equivalent tools, which are implemented using shell scripts and shell functions. In order to invoke these scripts and functions successfully the shell environment mush have been correctly configured. By default `venv-management` attempts to use the current user's preferred shell by examining the `$SHELL` environment variable. This can be overridden by setting the `$VENV_MANAGEMENT_SHELL` variable with a shell executable name or the path to a shell executable, for example:

```
export VENV_MANAGEMENT_SHELL=zsh
```

If neither `$SHELL` nor `$VENV_MANAGEMENT_SHELL` are set, an attempt to use `bash` will be made.

2.2 Shell configuration

The selected shell must be configured to make the `virtualenvwrapper` commands available. By default, `venv-management` will source the `rc` file corresponding to the selected shell, for example `.bashrc` for `bash`, `.zshrc` for `zsh`, and so on, on the basis that `virtualenvwrapper` initialization is often performed from these files. If the `rc` file for the selected shell can only be usefully sourced in an interactive shell, set `VENV_MANAGEMENT_INTERACTIVE_SHELL` to `yes`:

```
export VENV_MANAGEMENT_INTERACTIVE_SHELL=yes
```

Should you wish to specify a different file for shell configuration, provide its path in the `VENV_MANAGEMENT_SETUP_FILEPATH` environment variable. For example, since `.bashrc` may return immediately in non-interactive shells, and only login shells source `.profile` on start-up, you may want to set up virtualenvwrapper or an equivalent in in a separate file, in this example called `.venvwraprc`:

```
# .venvwraprc
source /usr/local/bin/virtualenvwrapper.sh
```

and then source this file in turn from, say, `.bashrc`.

If the `VENV_MANAGEMENT_USE_SETUP` variable is set to `yes`, the script whose filepath is specified in the `VENV_MANAGEMENT_SETUP_FILEPATH` variable will be as necessary before executing the commands run by this package:

```
export VENV_MANAGEMENT_USE_SETUP=yes
export VENV_MANAGEMENT_SETUP_FILEPATH=$HOME/.venvwraprc
```

You can also source this custom config file in a shell-specific `rc` file using the `source` or `.` command, so that `virtualenvwrapper` could be used in interactive shells.

2.3 Driver preference

If you have multiple virtualenv wrapper implementations installed, you can specify the order in which they will be tried with the `VENV_MANAGEMENT_PREFERRED_DRIVERS` environment variable. The first working implementation will be used:

```
export VENV_MANAGEMENT_PREFERRED_DRIVERS="virtualenvwrapper,virtualenv-sh"
```

2.4 Configuring for use with *pyenv* and *pyenv-virtualenvwrapper*

The pyenv tool helps with managing multiple Python versions simultaneously. The pyenv-virtualenvwrapper plugin for *pyenv*, helps with making *virtualenvwrapper* available to Python environments made using *pyenv*.

Make a file called `.venvwraprc` containing command to initialize both *pyenv* and *pyenv-virtualenvwrapper* according to your needs:

```
export PYENV_ROOT=$HOME/.pyenv
export PATH=$PYENV_ROOT/bin:$PATH
eval "$(pyenv init -)"
eval "$(pyenv init --path)"
pyenv virtualenvwrapper
```

This file can be placed in, say, your home directory. You can source this file from your preferred shell's `rc` file (*e.g.* `.bashrc`) in order ensure both *pyenv* and *virtualenvwrapper* are available in your interactive shell sessions.

In the environment of the Python program which uses this `venv-management` package set these environment variables:

```
export VENV_MANAGEMENT_USE_SETUP=yes
export VENV_MANAGEMENT_SETUP_FILEPATH=$HOME/.venvwraprc
```

This will allow `venv-management` to ensure that *virtualenvwrapper* is properly initialized before its commands are executed.

CHAPTER

THREE

VENV_MANAGEMENT PACKAGE

Submodules:

3.1 venv_management.api module

The public API.

3.1.1 Summary

Functions:

check_environment	Returns: A 2-tuple containing the status output of the setup command, and text output
discard_virtual_env	Discard a virtual environment.
driver_name	Get the name of the driver.
ensure_virtual_env	Ensure a virtualenv with the given name and version exists.
has_virtualenvwrapper	Determine whether virtualenvwrapper available and working.
list_virtual_envs	A list of virtualenv names.
make_virtual_env	Make a virtual env.
python_executable_path	Find the Python executable for a virtual environment.
python_name	Find the name of the Python in a virtual environment.
python_version	Find the version of the Python in virtual environment.
remove_virtual_env	Remove a virtual environment.
resolve_virtual_env	Given the name of a virtual environment, get its path.
virtual_env	A context manager that ensures a virtualenv with the given name and version exists.

3.1.2 Reference

venv_management.api.**check_environment**() → Tuple[int, str]
Returns: A 2-tuple containing the status output of the setup command, and text output

venv_management.api.**has_virtualenvwrapper**()
Determine whether virtualenvwrapper available and working.

Returns True if virtualenvwrapper is available and working, otherwise False.

venv_management.api.**list_virtual_envs**() → List[str]
A list of virtualenv names.

Returns A list of string names in case-sensitive alphanumeric order.

Raises *ImplementationNotFound* – If no virtualenvwrapper implementation could be found.

venv_management.api.**make_virtual_env**(*name*, *, *python=None*, *project_path=None*, *packages=None*, *requirements_file=None*, *system_site_packages=False*, *pip=True*, *setuptools=True*, *wheel=True*)
Make a virtual env.

Parameters

- **name** – The name of the virtual environment.
- **python** – The target interpreter for which to create a virtual environment, either the name of the executable, or full path.
- **project_path** – An optional path to a project which will be associated with the new virtual environment.
- **packages** – An optional sequence of package names for packages to be installed.
- **requirements_file** – An optional path to a requirements file to be installed.
- **system_site_packages** – If True, give access to the system site packages.
- **pip** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **setuptools** – If True, or 'latest' the latest pip will be installed. If False, setuptools will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **wheel** – If True, or 'latest' the latest pip will be installed. If False, wheel will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.

Returns The Path to the root of the virtualenv, or None if the path could not be determined.

Raises RuntimeError – If the virtualenv could not be created.

venv_management.api.**resolve_virtual_env**(*name*)
Given the name of a virtual environment, get its path.

Parameters **environment.** (*The name of a virtual*) –

Returns The path to the virtual environment directory.

Raises

- ValueError – If the virtual environment name is not known.
- RuntimeError – If the path could not be determined.

venv_management.api.**virtual_env**(*name*, *expected_version=None*, ***, *force=False*, ***kwargs*)
A context manager that ensures a virtualenv with the given name and version exists.

Irrespective of whether the virtual environment already exists, it will be removed when the context manager exits.

Parameters

- **name** – The name of the environment to check for.
- **expected_version** – An optional required version as a string. "3.8" will match "3.8.2"
- **force** – Force replacement of an existing virtual environment which has the wrong version.
- ****kwargs** – Arguments which will be forwarded to mkvirtualenv if the environment needs to be created.

Returns A context manager that manages the lifecycle of the virtual environment.

Raises RuntimeError – If the virtual environment couldn't be created or replaced.

venv_management.api.**ensure_virtual_env**(*name*, *expected_version=None*, ***, *force=False*, ***kwargs*)
Ensure a virtualenv with the given name and version exists.

Parameters

- **name** – The name of the environment to check for.
- **expected_version** – An optional required version as a string. "3.8" will match "3.8.2"
- **force** – Force replacement of an existing virtual environment which has the wrong version.
- ****kwargs** – Arguments which will be forwarded to mkvirtualenv if the environment needs to be created.

Returns The path to the virtual environment.

Raises RuntimeError – If the virtual environment couldn't be created or replaced.

venv_management.api.**remove_virtual_env**(*name: str*)
Remove a virtual environment.

Parameters **name** – The name of the virtual environment to remove.

Raises

- ValueError – If there is no environment with the given name.
- RuntimeError – If the virtualenv could not be removed.

venv_management.api.**discard_virtual_env**(*name: str*)
Discard a virtual environment.

Parameters **name** – The name of the virtual environment to remove.

Raises

- RuntimeError – If the virtualenv could not be removed.
- ValueError – If the name is empty.

venv_management.api.**python_executable_path**(*env_dirpath: Union[pathlib.Path, str]*) → pathlib.Path
Find the Python executable for a virtual environment.

Parameters **env_dirpath** – The path to the root of a virtual environment (Path or str).

Returns A Path object to the executable.

Raises ValueError – If the env_dirpath is not a virtual environment.

venv_management.api.**python_name**(*env_dirpath: Union[pathlib.Path, str]*) → str
Find the name of the Python in a virtual environment.

Parameters **env_dirpath** – The path to the root of a virtual environment (Path or str).

Returns A descriptive string.

Raises **ValueError** – If the env_dirpath is not a virtual environment.

venv_management.api.**python_version**(*env_dirpath: Union[pathlib.Path, str]*) → str
Find the version of the Python in virtual environment.

Parameters **env_dirpath** – The path to the root of a virtual environment (Path or str).

Returns A version string, such as "3.8.1"

Raises **ValueError** – If the env_dirpath is not a virtual environment.

venv_management.api.**driver_name**() → str
Get the name of the driver.

3.2 venv_management.driver module

The virtualenvwrapper driver interface and factories.

3.2.1 Summary

Exceptions:

DriverExtensionError	Indicates that an error specific to a driver extension occurred.

Classes:

Driver	Defines the interface for a virtualenvwrapper-equivalent driver.

Functions:

create_driver	Create a driver
driver	Obtain a Driver instance.
driver_names	A list of available driver extensions.

3.2.2 Reference

class venv_management.driver.**Driver**(*name*)
Bases: *venv_management.extension.Extension*

Defines the interface for a virtualenvwrapper-equivalent driver.

abstract list_virtual_envs() → List[str]
The virtual environments available to this package.

Returns A list of virtual environment names which can be manipulated by this package.

abstract make_virtual_env(*name*, ***, *python=None*, *project_path=None*, *packages=None*, *requirements_file=None*, *system_site_packages=False*, *pip=True*, *setuptools=True*, *wheel=True*)

Make a virtual env.

Parameters

- **name** – The name of the virtual environment.
- **project_path** – An optional path to a project which will be associated with the new virtual environment.
- **packages** – An optional sequence of package names for packages to be installed.
- **requirements_file** – An optional path to a requirements file to be installed.
- **python** – The target interpreter for which to create a virtual environment, either the name of the executable, or full path.
- **system_site_packages** – If True, give access to the system site packages.
- **pip** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **setuptools** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **wheel** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.

Returns The Path to the root of the virtualenv, or None if the path could not be determined.

Raises

- *PythonNotFound* – If the requested Python version could not be found.
- *CommandNotFound* – If the required command could not be found.
- RuntimeError – If the virtualenv could not be created.

abstract remove_virtual_env(*name: str*)

Remove a virtual environment.

Parameters **name** – The name of the virtual environment to be removed.

Raises ValueError – If the name is empty.

abstract resolve_virtual_env(*name: str*) → pathlib.Path

Obtain a path to the virtual environment directory.

Parameters **name** – The name of the virtual environment.

Returns The path of the named virtual environment.

Raises ValueError – If there is no virtual environment with the supplied name.

exception venv_management.driver.**DriverExtensionError**

Bases: *venv_management.extension.ExtensionError*

Indicates that an error specific to a driver extension occurred.

venv_management.driver.**create_driver**(*driver_name*) → *venv_management.driver.Driver*

Create a driver

Parameters **driver_name** – The name of the driver to create.

Returns A Driver instance.

Raises **ImplementationNotFoundError** – If a driver could be created but the driver could not find a working virtualenvwrapper (or equivalent) implementation.

venv_management.driver.**driver_names**() → List[*venv_management.driver.Driver*]
A list of available driver extensions.

There is no guarantee that the listed drivers are backed by functioning virtualenvwrapper implementations.

venv_management.driver.**driver**() → *venv_management.driver.Driver*
Obtain a Driver instance.

Returns A Driver corresponding to an available virtualenvwrapper implementation.

Raises ***ImplementationNotFound*** – If no suitable virtualenvwrapper installation was found

3.3 venv_management.environment module

3.3.1 Summary

Functions:

preferred_drivers	The preferred drivers from optionally controlled by VENV_MANAGEMENT_PREFERRED_DRIVERS.
preferred_shell	The preferred shell from the VENV_MANAGEMENT_SHELL environment variable.
shell_is_interactive	True if the shell has been set to interactive.

3.3.2 Reference

venv_management.environment.**preferred_shell**(*preferred_shell_name*)
The preferred shell from the VENV_MANAGEMENT_SHELL environment variable.

Parameters **preferred_shell_name** – The name of the preferred shell to use if the environment variable VENV_MANAGEMENT_SHELL is not set.

Returns The name of the preferred shell.

venv_management.environment.**shell_is_interactive**()
True if the shell has been set to interactive.

Control the setting with the VENV_MANAGEMENT_INTERACTIVE_SHELL environment variable by setting it to 'yes' or 'no'.

Returns True if the shell is interactive, otherwise False.

venv_management.environment.**preferred_drivers**(*available_driver_names*)
The preferred drivers from optionally controlled by VENV_MANAGEMENT_PREFERRED_DRIVERS.

Parameters **available_driver_names** – A list of available driver names.

Returns A list of available driver names, with the preferred drivers first.

3.4 venv_management.errors module

Exception classes.

3.4.1 Summary

Exceptions:

CommandNotFound	Raised when a shell command could not be found.
ImplementationNotFound	Raised when a virtual environment driver is not backed by a working implementation.
PythonNotFound	Raised when the requested Python version is not found.

3.4.2 Reference

exception venv_management.errors.**CommandNotFound**
Bases: Exception

Raised when a shell command could not be found.

exception venv_management.errors.**ImplementationNotFound**
Bases: Exception

Raised when a virtual environment driver is not backed by a working implementation.

exception venv_management.errors.**PythonNotFound**
Bases: Exception

Raised when the requested Python version is not found.

3.5 venv_management.extension module

The plug-in extension mechanism.

3.5.1 Summary

Exceptions:

ExtensionError	Raised if there is an error in an extension.

Classes:

Extension	A generic extension point for plug-ins.

Functions:

create_extension	Create an instance of a named extension.

continues on next page

Table 9 – continued from previous page

list_dirpaths	A mapping of extension names to extension package paths.
list_extensions	List the names of the extensions available in a given namespace.

3.5.2 Reference

exception venv_management.extension.**ExtensionError**
Bases: Exception

Raised if there is an error in an extension.

venv_management.extension.**list_extensions**(*namespace*)
List the names of the extensions available in a given namespace.

venv_management.extension.**list_dirpaths**(*namespace*)
A mapping of extension names to extension package paths.

class venv_management.extension.**Extension**(*name*)
Bases: abc.ABC

A generic extension point for plug-ins.

property kind: str
The kind of extension.

property name: str
The name of the extension.

The name used to create the extension.

classmethod dirpath()
The directory path to the extension package.

property version
The version of the extension.

venv_management.extension.**create_extension**(*kind*, *namespace*, *name*, *exception_type*, **args*, ***kwargs*) → *venv_management.extension.Extension*
Create an instance of a named extension.

Parameters

- **kind** – The kind of extension to create.
- **namespace** – The namespace within which the extension is a member.
- **name** – The name of the extension.
- **exception_type** – The type of exception to be raised if an extension could not be created.
- ***args** – Positional arguments to forward to the extensions constructor.
- ****kwargs** – Keyword arguments to forward to the extensions constructor.

Returns An extension instance.

Raises **exception_type** – If the requested extension could not be located.

3.6 venv_management.shell module

3.6.1 Summary

Functions:

get_status_output	Return (status, output) of executing cmd in a shell.
has_interactive_warning	Determine whether a line of text contains a warning emitted by a shell.
remove_interactive_shell_warnings	Remove shell warnings from lines of text.
sub_shell_command	Build a command to run a given command in an interactive subshell.

3.6.2 Reference

venv_management.shell.**sub_shell_command**(*command*, *suppress_setup_output=True*)
Build a command to run a given command in an interactive subshell.

Parameters

- **command** – The command for the subshell.
- **suppress_setup_output** – Suppress output from the environment setup command if True, (the default), otherwise capture it.

Returns A string which can be used with the subprocess module.

Raises **ValueError** – If the subshell command could not be determined.

venv_management.shell.**get_status_output**(*cmd: List[str]*, *success_statuses=None*) → Tuple[int, str]
Return (status, output) of executing cmd in a shell.

Parameters

- **cmd** – A list of command arguments to be executed.
- **success_statuses** – A container of integer status codes which indicate success.

Execute the string 'cmd' in a shell with 'check_output' and return a 2-tuple (status, output). Universal newlines mode is used, meaning that the result with be decoded to a string.

A trailing newline is stripped from the output. The exit status for the command can be interpreted according to the rules for the function 'wait'.

venv_management.shell.**has_interactive_warning**(*line: str*)
Determine whether a line of text contains a warning emitted by a shell.

The shell program itself (e.g. bash) can emit warnings under certain circumstances which clutter output; for example, when running a shell in interactive mode without a connected terminal. This predicate can identify lines of text containing such warnings.

Parameters **line** – A string which may contain a shell warning.

Returns True if the line contains a shell warning, otherwise False.

venv_management.shell.**remove_interactive_shell_warnings**(*lines: str*) → str
Remove shell warnings from lines of text.

The shell program itself (e.g. bash) can emit warnings under certain circumstances which clutter output; for example, when running a shell in interactive mode without a connected terminal. This predicate can identify lines of text containing such warnings.

Parameters **lines** – A string (possibly multiline) which may contain lines which have shell warnings.

Returns The argument string without any lines containing matching shell warnings.

3.7 venv_management.utilities module

Utility functions.

3.7.1 Summary

Functions:

compatible_versions	Determine whether two versions are equal.
parse_package_arg	Make a command-line argument string specifing whether and which verison of a package to install.
str_to_bool	Convert a string representation of truth to true (1) or false (0).

3.7.2 Reference

venv_management.utilities.**compatible_versions**(*actual_version: str*, *required_version: str*) → bool
Determine whether two versions are equal.

Only the dot separated elements in common are taken into account, so actual "3.7.4" compared with "3.7" will return True.

Parameters

- **actual_version** – A dot separated version.
- **required_version** – A dot separated version.

Returns True if the actual_version is compatible with the required_version, otherwise False.

venv_management.utilities.**parse_package_arg**(*name*, *arg*)
Make a command-line argument string specifing whether and which verison of a package to install.

Parameters

- **name** – The name of the package.
- **arg** – True if the package is required, False if the package is not required, or a string containing a version number if a specific version of the package is required.

Returns A string which can be used as an argument to the virtualenv command.

venv_management.utilities.**str_to_bool**(*val*)
Convert a string representation of truth to true (1) or false (0).

True values are 'y', 'yes', 't', 'true', 'on', and '1'; false values are 'n', 'no', 'f', 'false', 'off', and '0'. Raises ValueError if 'val' is anything else.

3.8 venv_management.version module

Version information.

Subpackages:

3.9 venv_management.ext package

Subpackages:

3.9.1 venv_management.ext.drivers package

Subpackages:

venv_management.ext.drivers.pyenv_virtualenv package

Submodules:

venv_management.ext.drivers.pyenv_virtualenv.driver module

Summary

Classes:

PyEnvVirtualEnvDriver

Reference

class venv_management.ext.drivers.pyenv_virtualenv.driver.**PyEnvVirtualEnvDriver**(*name*)

Bases: *venv_management.driver.Driver*

list_virtual_envs() → List[str]

A list of virtualenv names.

Returns A list of string names in case-sensitive alphanumeric order.

Raises **FileNotFoundError** – If virtualenvwrapper.sh could not be located.

make_virtual_env(*name*, *, *python=None*, *project_path=None*, *packages=None*, *requirements_file=None*, *system_site_packages=False*, *pip=True*, *setuptools=True*, *wheel=True*)

Make a virtual env.

Parameters

- **name** – The name of the virtual environment.
- **project_path** – An optional path to a project which will be associated with the new virtual environment. (not supported by pyenv-virtualenv)

- **packages** – An optional sequence of package names for packages to be installed. (not supported by pyenv-virtualenv)
- **requirements_file** – An optional path to a requirements file to be installed. (not supported by pyenv-virtualenv)
- **python** – The target interpreter for which to create a virtual environment, either the name of the executable, or full path.
- **system_site_packages** – If True, give access to the system site packages.
- **pip** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **setuptools** – If True, or 'latest' the latest setuptools will be installed. If False, setuptools will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **wheel** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.

Returns The Path to the root of the virtualenv, or None if the path could not be determined.

Raises

- *CommandNotFound* – If the required command could not be found.
- RuntimeError – If the virtualenv could not be created.

remove_virtual_env(*name*)
Remove a virtual environment.

Parameters **name** – The name of the virtual environment to remove.

Raises

- ValueError – If there is no environment with the given name.
- RuntimeError – If the virtualenv could not be removed.

resolve_virtual_env(*name: str*) → pathlib.Path
Resolve the path to a virtual environment.

Parameters **name** – The name of the virtual environment.

Returns The path to the virtual environment in the $HOME/.pyenv/versions/<virtual_env_name> format.

venv_management.ext.drivers.virtualenv_sh package

Submodules:

venv_management.ext.drivers.virtualenv_sh.driver module

Summary

Classes:

VirtualEnvShDriver

Reference

class venv_management.ext.drivers.virtualenv_sh.driver.**VirtualEnvShDriver**(*name*)

Bases: *venv_management.driver.Driver*

list_virtual_envs() → List[str]

A list of virtualenv names.

Returns A list of string names in case-sensitive alphanumeric order.

Raises **FileNotFoundError** – If virtualenvwrapper.sh could not be located.

make_virtual_env(*name*, *, *python=None*, *project_path=None*, *packages=None*, *requirements_file=None*, *system_site_packages=False*, *pip=True*, *setuptools=True*, *wheel=True*)

Make a virtual env.

Parameters

- **name** – The name of the virtual environment.
- **project_path** – An optional path to a project which will be associated with the new virtual environment.
- **packages** – An optional sequence of package names for packages to be installed.
- **requirements_file** – An optional path to a requirements file to be installed.
- **python** – The target interpreter for which to create a virtual environment, either the name of the executable, or full path.
- **system_site_packages** – If True, give access to the system site packages.
- **pip** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **setuptools** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **wheel** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.

Returns The Path to the root of the virtualenv, or None if the path could not be determined.

Raises

- *PythonNotFound* – If the requested Python version could not be found.
- *CommandNotFound* – If the required command could not be found.

- **RuntimeError** – If the virtualenv could not be created.

remove_virtual_env(*name*)
Remove a virtual environment.

Parameters **name** – The name of the virtual environment to be removed.

Raises **ValueError** – If the name is empty.

resolve_virtual_env(*name: str*) → pathlib.Path
Obtain a path to the virtual environment directory.

Parameters **name** – The name of the virtual environment.

Returns The path of the named virtual environment.

Raises **ValueError** – If there is no virtual environment with the supplied name.

venv_management.ext.drivers.virtualenvwrapper package

Submodules:

venv_management.ext.drivers.virtualenvwrapper.driver module

Summary

Classes:

VirtualEnvWrapperDriver

Reference

class venv_management.ext.drivers.virtualenvwrapper.driver.**VirtualEnvWrapperDriver**(*name*)
Bases: *venv_management.driver.Driver*

list_virtual_envs() → List[str]
A list of virtualenv names.

Returns A list of string names in case-sensitive alphanumeric order.

Raises **FileNotFoundError** – If virtualenvwrapper.sh could not be located.

make_virtual_env(*name*, *, *python=None*, *project_path=None*, *packages=None*, *requirements_file=None*, *system_site_packages=False*, *pip=True*, *setuptools=True*, *wheel=True*)
Make a virtual env.

Parameters

- **name** – The name of the virtual environment.
- **project_path** – An optional path to a project which will be associated with the new virtual environment.
- **packages** – An optional sequence of package names for packages to be installed.
- **requirements_file** – An optional path to a requirements file to be installed.

- **python** – The target interpreter for which to create a virtual environment, either the name of the executable, or full path.
- **system_site_packages** – If True, give access to the system site packages.
- **pip** – If True, or 'latest' the latest pip will be installed. If False, pip will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **setuptools** – If True, or 'latest' the latest pip will be installed. If False, setuptools will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.
- **wheel** – If True, or 'latest' the latest pip will be installed. If False, wheel will not be installed. If 'bundled', the bundled version will be installed. If a specific version string is given, that version will be installed.

Returns The Path to the root of the virtualenv, or None if the path could not be determined.

Raises

- *PythonNotFound* – If the requested Python version could not be found.
- RuntimeError – If the virtualenv could not be created.

remove_virtual_env(*name*)

Remove a virtual environment.

Parameters **name** – The name of the virtual environment to remove.

Raises

- ValueError – If there is no environment with the given name.
- RuntimeError – If the virtualenv could not be removed.

resolve_virtual_env(*name: str*) → pathlib.Path

Obtain a path to the virtual environment directory.

Parameters **name** – The name of the virtual environment.

Returns The path of the named virtual environment.

Raises ValueError – If there is no virtual environment with the supplied name.

3.10 Summary

__all__ Exceptions:

CommandNotFound	Raised when a shell command could not be found.
ImplementationNotFound	Raised when a virtual environment driver is not backed by a working implementation.
PythonNotFound	Raised when the requested Python version is not found.

__all__ Functions:

check_environment	Returns: A 2-tuple containing the status output of the setup command, and text output
discard_virtual_env	Discard a virtual environment.
ensure_virtual_env	Ensure a virtualenv with the given name and version exists.
has_virtualenvwrapper	Determine whether virtualenvwrapper available and working.
list_virtual_envs	A list of virtualenv names.
make_virtual_env	Make a virtual env.
python_executable_path	Find the Python executable for a virtual environment.
python_name	Find the name of the Python in a virtual environment.
python_version	Find the version of the Python in virtual environment.
remove_virtual_env	Remove a virtual environment.
resolve_virtual_env	Given the name of a virtual environment, get its path.
virtual_env	A context manager that ensures a virtualenv with the given name and version exists.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

V

INDEX

N

P

R

S

V